A dissertation presented to the faculty of the Graduate School of Yale University in candidacy for the degree of Doctor of Philosophy.

Numerical Techniques for the Solution of the Time-dependent Schrödinger Equation and their Parallel Implementation

Faisal Saied

Research Report YALEU/DCS/RR-811
July 1990

The author was supported in part by the Office of Naval Research under contracts N00014-86-K-0310, N00014-88-K-0262 and N00014-86-K-1671.

ABSTRACT

Numerical Techniques for the Solution of the Time-dependent Schrödinger Equation and their Parallel Implementation

Faisal Saied

Yale University

1990

We investigate numerical techniques for the solution of the time-dependent Schrödinger equation in one and two space dimensions. We introduce a framework for constructing finite difference schemes based on Padé approximations for both the time and space discretization and apply this framework to construct high order finite difference schemes for Schrödinger's equation in conjunction with an operator splitting approach. We also discuss three level schemes as an alternative to operator splitting.

The accuracy and stability of these methods is studied, and their efficiencies are compared. The results of some numerical comparisons of the methods are presented.

For two space dimensions, some of the new techniques proposed include a split-step Crank-Nicolson scheme, where the implicit equations at each time step can be solved by a fast Poisson solver. The two-dimensional methods have ADI (alternating direction implicit) analogues which reduce the complexity of the computations.

We describe how a balanced form of cyclic reduction and a method that exploits the transpose operation on hypercubes lead to effective strategies for parallelizing the solution of multiple tridiagonal equations that arise in the solution of the Schrödinger equation in two space dimensions in the ADI approach. We show that in spite of the implicit nature of these schemes, they can be parallelized with high efficiencies on MIMD hypercubes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Schrödinger Equation

The Schrödinger equation is a fundamental equation in quantum mechanics. It also arises in underwater acoustics, where the Helmholtz equation for the acoustic pressure is transformed into an equation of the same form by applying the so-called "parabolic approximation" (see [6]). In this section, we introduce the time-dependent Schrödinger equation in one dimension.

The one-dimensional Schrödinger equation is (SE-1d)

$$i\frac{\partial u}{\partial t} = Hu \equiv \frac{\partial^2 u}{\partial x^2} + V(x,t)u, \qquad x \in (a,b), \quad t > 0, \tag{1.1}$$

$$u(a,t) = u(b,t) = 0, \quad t > 0,$$
$$u(x,0) = \phi(x),$$

where $V(x,t)$ is a given real-valued potential function and $\phi(x)$ is a complex-valued initial condition, and u is a complex-valued function of x and t.

Equation (1.1) has essentially the same structure as the corresponding equation in quantum mechanics and the standard parabolic equation in underwater acoustics and numerical methods for solving it can be applied to the quantum mechanics and underwater acoustics problems. The Schrödinger equation conserves the L_2-norm of the solution. Dissipation can be introduced by adding an appropriate imaginary part to the potential V. In this dissertation, we shall focus on numerical schemes for the solution equation and not attempt to attach a physical interpretation to the computed solution.

In Chapter 6, we will introduce the two-dimensional analogue of (SE-1d) and present methods for its numerical solution.

The time-dependent Schrödinger equation in quantum mechanics is formulated as a Cauchy problem (initial value problem) for $x \in \mathbb{R}$ with the requirement that the solution $u(x,t) \in L^2(\mathbb{R})$, for each t. In practical computations, the domain is usually truncated and the problem is converted to the initial/boundary-value form that we are considering. We will not discuss the approximations that are made in going from an unbounded to a bounded domain.

1.2 Outline of the Dissertation

In this dissertation, we investigate numerical techniques for solving the time-dependent Schrödinger equation in one and two space dimensions. In the first part (Chapters 2–6), we consider the numerical methods in the traditional setting of sequential computation, where the total number of arithmetic operations required to produce a result with a prescribed accuracy is a good measure of a method's performance. In the second part (Chapters 7 and 8), we study the parallel implementation of some of the methods on hypercube multiprocessors, the Intel iPSC/1 and iPSC/2, and evaluate the relative merits of different approaches to parallelizing the computations.

In the next section, we develop some notation and review some background material.

Chapter 2 introduces two well-known numerical methods, the Crank-Nicolson and the split-step Fourier methods and summarizes their properties. The purpose is to introduce both concepts and terminology that are needed to describe some new methods that are introduced in later chapters.

In Chapter 3, we introduce a Padé framework for constructing difference schemes for the Schrödinger equation. In this framework, we synthesize previous work on Padé methods for approximating operator exponentials and the second derivative operator and define a four-parameter family of difference methods based on Padé approximations in time and space. This framework reproduces many of the classical numerical methods and provides a simple mechanism for constructing and analyzing certain natural generalizations.

In Chapter 4, we introduce a number of new schemes for the time-dependent Schrödinger equation in one space dimension. One of these is a split-step implementation of an explicit scheme. We then combine the split-step approach and the Padé framework developed in earlier chapters to define a family of split-step finite difference schemes. Some issues relating to efficiency are discussed. Next, we introduce an analogous family of schemes that uses a three-level scheme instead of the split-step approach. We then discuss a few approaches to constructing schemes that have a higher order of accuracy in time. We conclude this chapter by briefly reviewing several methods that have been applied by other authors for solving the Schrödinger equation, and other partial differential equations with a related structure.

In Chapter 5, we present numerical results for several of the new methods proposed here and for some of the other methods we have discussed. In particular, the efficiency of these methods for a test problem is compared. We employ a heuristic that has been used by some other authors to select the time and space mesh sizes optimally. This applies to the finite difference methods in particular. The results in this chapter make clear the relative merits of the various methods as the accuracy requirements and the "energy" in the solution are varied.

In Chapter 6, we consider extensions of a number of these methods to two space dimensions. We also introduce split-step alternating direction methods that have no one-dimensional analogues.

In Chapter 7, we introduce the hypercube architecture . A simple and widely used model for analyzing the performance of parallel algorithms is described. We then consider various

approaches to solving a system of tridiagonal equations on a hypercube and extend these methods to the case of multiple tridiagonal systems. An empirical (based on experiments on the iPSC/1) and theoretical comparison of hypercube algorithms for multiple tridiagonal systems is presented.

In Chapter 8, we describe CUBE-ADI, a hypercube algorithm for the ADI schemes discussed in Chapter 6 for the Schrödinger equation in two space dimensions. Experimental results from the iPSC/1 and the iPSC/2 are presented that indicate that our approach to parallelizing ADI on a hypercube is an effective one. We conclude the chapter by showing how some of the methods could be modified for large (but not massively parallel) hypercubes.

1.3 Some Notation and Background

Let $x_1, \ldots, x_m$ be a uniform discretization of (a, b), with mesh width $h \equiv (b-a)/(m+1)$ and $x_j \equiv a + jh$. Let k be the time step and let $t_n \equiv nk$. The quantity

$$r \equiv \frac{k}{h^2} \tag{1.2}$$

is called the mesh ratio.

The approximate solution to (SE-1d) at (x_j, t_n) will be denoted by U_j^n. The expression U^n, without a subscript, will denote an m-vector, $(U_1^n, \ldots, U_m^n)^T$.

Let $V_j^n \equiv V(x_j, t_n)$ and let V^n be the $m \times m$ diagonal matrix whose j-th diagonal entry is $V(x_j, t_n)$.

In discussing finite difference schemes, we will use the following standard notation,

$$\delta_x v(x_j) \equiv v(x_j + \frac{h}{2}) - v(x_j - \frac{h}{2}) \tag{1.3}$$

whereby the dependence on h is not explicitly displayed. An immediate consequence of this definition is that

$$\delta_x^2 U_j^n = U_{j-1}^n - 2U_j^n + U_{j+1}^n, \tag{1.4}$$

which corresponds to the most common finite difference discretization of the second derivative operator (without the $1/h^2$ factor). When δ_x^2 is applied to U^n, it will be interpreted as the $m \times m$ real, symmetric, tridiagonal, Toeplitz matrix T defined by

$$T_{ij} \equiv \begin{cases} -2, & \text{for} \quad i = j \\ 1, & \text{for} \quad i = j-1 \quad \text{or} \quad i = j+1 \\ 0, & \text{for} \quad |i-j| > 1. \end{cases}$$

We will use the following notation for the operators denoting differentiation with respect to x,

$$D_x^2 \equiv \frac{\partial^2}{\partial x^2}.$$

The complex conjugate of a will be denoted by $\bar{a}$.

We will use the standard $O(\cdot)$ and $o(\cdot)$ notation (see, for example, [3]). We will say that a function of k and h is $O(k^n, h^m)$ if its dominant part has both $O(k^n)$ and $O(h^m)$ terms (in the appropriate limit).

A computation scheme for advancing a discrete approximation to the solution of a linear differential equation of the form $u_t = Hu$ from t_n to $t_{n+1} = t_n + k$ will be denoted by an $m \times m$ matrix $C(k)$:

$$U^{n+1} = C(k)U^n,$$

where m is the number of components in U^n. The spatial dependence has been suppressed in the above expression. $C(k)$ can be viewed as an approximation to the evolution operator, $exp(kH)$, where the operator exponential is defined by

$$e^{kH} \equiv \sum_{j=0}^{\infty} k^j \frac{H^j}{j!}.$$

We now recall some standard definitions.

Definition 1.3.1 *The local truncation error of a scheme $C(k)$ that approximates the Schrödinger equation is defined to be*

$$TE = \frac{1}{k}\{u(x,t+k) - C(k)u(x,t)\}, \tag{1.5}$$

where $u(x,t)$ satisfies (SE-1d).
The scheme $C(k)$ is called consistent if the truncation error is $O(k^p)$, for some $p > 0$.
The scheme $C(k)$ is stable if $C(k)^n$ is uniformly bounded for $k \leq k_{max}$ and $nk \leq T$, for a fixed T, as $n \to \infty$.

For linear, well-posed problems, the Lax-Richtmyer equivalence theorem [4] states that a consistent scheme is convergent if, and only if, it is stable.

For constant coefficient problems with periodic boundary conditions, the stability analysis can be carried out in the manner of von Neumann (see [2]), by examining the growth factor associated with the Fourier modes.

The Schrödinger equation conserves the 2-norm of the solution and many of the numerical schemes we will consider satisfy conditions that are stronger than stability. With this in mind, we introduce the following definitions.

Definition 1.3.2 *A scheme $C(k)$ is contractive if $||C(k)|| \leq 1$. $C(k)$ is unitary if $||C(k)|| = 1$.*

Clearly, unitary $\Rightarrow$ contractive $\Rightarrow$ stable. Note that like stability, contractivity and unitarity are properties of the *scheme* only. It is desirable that the stability properties of a scheme hold without any restriction on the mesh ratio r. When this is true, we refer to the scheme as being unconditionally stable (unconditionally contractive, unconditionally unitary).

Chapter 2

Some Standard Numerical Techniques for the Schrödinger Equation in One Space Dimension

In this chapter, we will discuss two widely used numerical methods for the Schrödinger equation in one space dimension (SE-1d), namely the Crank-Nicolson method and the split-step Fourier method.

2.1 The Crank-Nicolson Method

Consider the following one-parameter family of finite difference methods for (1.1).

$$\begin{aligned} \frac{i}{k}(U_j^{n+1} - U_j^n) &= \frac{1}{h^2}\delta_x^2 \left[\theta U_j^{n+1} + (1-\theta)U_j^n\right] \\ &+ \left[\theta V_j^{n+1} U_j^{n+1} + (1-\theta)V_j^n U_j^n\right], \end{aligned} \tag{2.1}$$

with $\theta \in [0,1]$. The choices $\theta = 0$ and $\theta = 1$ yield the forward and backward Euler methods, respectively.

For $\theta = 1/2$, we get the well-known Crank-Nicolson scheme, which can be written in matrix-vector notation as follows:

$$\left[I + \frac{ir}{2}\delta_x^2 + \frac{ik}{2}V^{n+1}\right] U^{n+1} = \left[I - \frac{ir}{2}\delta_x^2 - \frac{ik}{2}V^n\right] U^n. \tag{2.2}$$

This scheme is second-order accurate in time and space and is unconditionally stable. The time step is restricted in practice by the accuracy requirements. The local truncation error is

$$\mathrm{TE} = -\frac{1}{12}k^2\frac{\partial^3}{\partial t^3}u(x,t) + \frac{i}{12}h^2\frac{\partial^4}{\partial x^4}u(x,t) + o(k^2,h^2). \tag{2.3}$$

The properties of the Crank-Nicolson method for the Schrödinger equation are discussed in [11] in the context of underwater acoustics.

If V does not depend on t, then the Crank-Nicolson method for the Schrödinger equation is unitary. If V depends on t, then it can be shown to be unconditionally stable by the method of energy inequalities [14].

The matrix on the left-and side of (2.2) is diagonally dominant for sufficiently small r and k. If we neglect the kV term, this matrix is $I + \frac{ir}{2}T$, where T is defined in (1.5). This matrix is diagonally dominant for all r and the diagonal dominance increases as $|r|$ decreases. For example, if we scale $I + \frac{ir}{2}T$ to have ones on the off-diagonals, then the absolute value of the diagonal entry is ≈ 2.25, 2.81, and 4.4 for $r = 2, 1$, and 0.5, respectively.

The work per time step is a matrix-vector product to form the right-hand side and the solution of a tridiagonal system of equations by Gaussian elimination. If both sides of (2.2) are scaled so that the matrix on the left hand side has ones on the off-diagonals and there are m grid points in the x direction, then the cost of one time step is $26m$ (real) floating point operations, if V does not depend on t, and $40m$, if it does (and we have to update the matrices and recompute the LU factors). These counts can be reduced somewhat by using an implementation described in [66] that exploits the form of (2.2) to avoid the matrix-vector product. Instead of using (2.2), we can compute U^{n+1}as follows.

$$\begin{aligned} \left[\delta_x^2 + h^2 V - 2\frac{i}{r}I\right]\tilde{U}^{n+1} &= U^n \\ U^{n+1} &= -U^n - 4\frac{i}{r}\tilde{U}. \end{aligned}$$

2.2 The Split-step Approach

In this section, we review the split-step approach for solving a time-dependent partial differential equation

$$\frac{\partial u}{\partial t} = Lu,$$

where $L = A + B$. Split-step methods are also referred to as time-split methods [13] or methods of fractional steps [16]. Alternating direction methods and locally one-dimensional methods [1] are among the best known methods of this class.

In the following discussion, we will assume that L is time-independent. Then

$$u(t + k) = e^{k(A+B)}u(t). \tag{2.4}$$

Definition 2.2.1 *The commutator of A and B is $[A, B] \equiv AB - BA$.*

Definition 2.2.2 *A splitting SP(A,B,k) of $e^{k(A+B)}$ is an approximation to $e^{k(A+B)}$ consisting of a finite linear combination of finite products with factors of the form $e^{\alpha kA}$ and $e^{\beta kB}$.*

A splitting of $e^{k(A+B)}$ has an expansion of the form

$$SP(A,B,k) = \sum_{j=0}^{\infty} k^j p_j(A,B), \tag{2.5}$$

where the p_j are homogeneous polynomials of degree j in two non-commuting variables. A simple example of such a polynomial is $p(A,B) = (A+B)^2 = A^2 + AB + BA + B^2$, for non-commuting A and B.

Some well-known splittings are

$$SP_1(A,B,k) \equiv e^{kA}e^{kB} \tag{2.6}$$

$$SP_2(A,B,k) \equiv e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B} \tag{2.7}$$

$$SP_{2a}(A,B,k) \equiv \left(e^{kA}e^{kB} + e^{kB}e^{kA}\right)/2 \tag{2.8}$$

SP_2 and SP_{2a} are due to Strang [15].

We define the splitting error in a manner similar to the definition of the truncation error. The splitting error operator defined in [13] is closely related to the following definition.

Definition 2.2.3 *The splitting error for $SP(A,B,k)$ is defined to be*

$$SE(A,B,k)u(x,t) \equiv \frac{1}{k}\left\{e^{k(A+B)} - SP(A,B,k)\right\}u(x,t),$$

for an appropriate u. A splitting $SP(A,B,k)$ is ν-th order accurate if

$$p_j(A,B) = \frac{(A+B)^j}{j!}, \qquad \text{for } j = 0,\ldots,\nu.$$

Definition 2.2.4 *A consistent splitting is one that is at least first-order accurate and for which the splitting error vanishes when A and B commute.*

We will only consider consistent splittings. For such splittings, $p_0(A,B) = I$, $p_1(A,B) = A+B$, and for $p_j(A,B) = (A+B)^j/j!$ for all $j \geq 2$ when A and B commute. (In general, A and B do not commute and $p_j(A,B)$ can have up to 2^j terms.)

The splittings we will consider all have an expansion for the splitting error of the form

$$\frac{1}{k}\left\{e^{k(A+B)} - SP_*(A,B,k)\right\}u(x,t) = \sum_{j=0}^{\infty} k^j s_j^{(*)}(A,B)u(x,t), \tag{2.9}$$

where $s_j^{(*)}$ is a homogeneous polynomial of two non-commuting variables of degree $j+1$. For a given splitting, the p_j's and the s_j's clearly satisfy

$$p_j(A,B) + s_{j-1}(A,B) = (L^j)/j!. \tag{2.10}$$

We note that $SP_1(B,A,k)$, $SP_2(B,A,k)$ and $SP_{2a}(B,A,k)$ are also splittings of $e^{k(A+B)}$. If A and B commute, then the splitting error vanishes.

In the following three lemmas, we give the leading terms in the splitting errors for SP_1, SP_2 and SP_{2a}. The proofs of these lemmas are based on a direct computation and are omitted.

Lemma 2.2.1 *Let $SE_1(A,B,k)$ be the splitting error corresponding to SP_1. Then*

$$SE_1(A,B,k) = \sum_{j=0}^{\infty} k^j s_j^{(1)}(A,B),$$

where

$$\begin{aligned} s_0^{(1)} &\equiv 0, \\ s_1^{(1)}(A,B) &= \frac{1}{2}[B,A] \qquad \textit{and}, \end{aligned} \tag{2.11}$$

$$s_2^{(1)}(A,B) = \frac{1}{6}\{-2A^2B - 2AB^2 + ABA + BAB + BA^2 + B^2A\}. \tag{2.12}$$

Lemma 2.2.2 *Let $SE_2(A,B,k)$ be the splitting error corresponding to SP_2. Then*

$$SE_2(A,B,k) = \sum_{j=0}^{\infty} k^j s_j^{(2)}(A,B),$$

where

$$s_0^{(2)}, s_1^{(2)} \equiv 0,$$

$$s_2^{(2)}(A,B) = \frac{1}{24}\left\{[B,[B,A]] - 2[A,[A,B]]\right\}, \tag{2.13}$$

and

$$\begin{aligned} s_3^{(2)}(A,B) \quad &= \frac{1}{48}\Big\{AB^3 - A^2B^2 - BAB^2 - 2A^3B - 4BA^2B \\ &\quad -B^2AB - 2BA^3 - B^2A^2 + B^3A + 2A^2BA \\ &\quad +2ABA^2 + 2ABAB + 2AB^2A + 2BABA\Big\}. \end{aligned} \tag{2.14}$$

Lemma 2.2.3 *Let $SE_{2a}(A,B,k)$ be the splitting error corresponding to SP_{2a}. Then*

$$SE_{2a}(A,B,k) = \sum_{j=0}^{\infty} k^j s_j^{(2a)}(A,B),$$

where

$$s_0^{(2a)}, s_1^{(2a)} \equiv 0$$

and

$$s_2^{(2a)}(A,B) = -\frac{1}{12}\left\{[A,[A,B]] + [B,[B,A]]\right\}. \tag{2.15}$$

The statement that SP_1 converges, i.e., $(e^{kA}e^{kB})^n \to e^{A+B}$, as $n \to \infty$ with $nk = 1$, is the well-known Trotter product formula [9]. We now estimate the "global" splitting error for SP_1 and SP_2 in the following theorem.

Theorem 2.2.1 *Let A, B be $m \times m$ matrices and let $nk = 1$, for positive integers n. There exist constants C_1, C_2 such that*

$$||e^{A+B} - \left(e^{kA}e^{kB}\right)^n|| \leq C_1 k.$$

and

$$||e^{A+B} - \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^n|| \leq C_2 k^2.$$

Proof. Our proof for SP_2 closely follows the one given in [9] (page 60), for SP_1.

$$\begin{aligned} e^{A+B} - \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^n &= \left(e^{k(A+B)}\right)^n - \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^n \\ &= \sum_{j=0}^{n-1} \left(e^{k(A+B)}\right)^j \left(e^{k(A+B)} - e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right) \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^{n-1-j}. \end{aligned}$$

The second equality is due to the cancellation of consecutive pairs of terms in the expanded form of the sum, with the exception of the first and last terms. It follows that

$$\begin{aligned} ||e^{A+B} - \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^n|| &\leq \sum_{j=0}^{n-1} e^{jk||A+B||}||e^{k(A+B)} - e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}|| \left(e^{||A||+||B||}\right)^{k(n-1-j)} \\ &\leq n||e^{k(A+B)} - e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}||e^{(||A||+||B||)(1-k)}. \end{aligned}$$

But from Lemma 2.2.2 we know that

$$||e^{k(A+B)} - e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}|| \leq k^3||s_3^{(2)}(A,B)|| + O(k^4), \qquad \text{as} \quad n \to \infty,$$

and hence

$$||e^{A+B} - \left(e^{\frac{k}{2}B}e^{kA}e^{\frac{k}{2}B}\right)^n|| \leq C_2 k^2, \qquad \text{as} \quad n \to \infty,$$

for $C_2 > ||s_3^{(2)}(A,B)||e^{||A||+||B||}$. ■

For the Schrödinger equation, A and B can be chosen to be skew-Hermitian so that the $e^{||A||+||B||}$ factor drops out of the bound for C_2.

The motivation for using split-step methods is that e^{kA} and e^{kB} can often be approximated more efficiently than $e^{k(A+B)}$. Broadly speaking, the use of operator splittings is desirable when the gain in efficiency more than offsets the errors introduced due to the non-commutativity of the operators A and B.

2.3 The Split-step Approach for the Schrödinger Equation

In this section, we discuss some general properties of the split-step approach applied to the Schrödinger equation. These properties will be common to all split-step methods that we will discuss later.

We first compute the leading term of the splitting error when SP_2 is applied to the Schrödinger equation with

$$Au \equiv -i\frac{\partial^2 u}{\partial x^2} \qquad \text{and} \qquad Bu \equiv -iVu. \tag{2.16}$$

A similar computation for the parabolic wave equation of underwater acoustics is given in [12]. The proof of the following lemma is based on a direct computation and is omitted. In the following, $f^{(j)}$ denotes $D_x^j f$, the j-th partial derivative of a smooth function f, with respect to x.

Lemma 2.3.1 *Let $u(x)$ and $V(x)$ be two and four times differentiable with respect to x, respectively. Then*

$$[A,[A,B]]\,u = i\left(V^{(4)}u + 4V^{(3)}u^{(1)} + 4V^{(2)}u^{(2)}\right), \tag{2.17}$$

$$[B,[B,A]]\,u = 2i(V^{(1)})^2 u. \tag{2.18}$$

We substitute (2.17) and (2.18) into (2.13) to get to (SE-1d).

$$k^2 s_2^{(2)}(A,B)u = i\frac{k^2}{24}\left(2(V^{(1)})^2 u - 2V^{(4)}u - 8V^{(3)}u^{(1)} - 8V^{(2)}u^{(2)}\right). \tag{2.19}$$

One could implement the SP_2-based split-step schemes with the roles of A and B interchanged in (2.7), i.e.,

$$SP_2'(A,B,k) \;\equiv\; e^{\frac{k}{2}A}e^{kB}e^{\frac{k}{2}A}.$$

For the Schrödinger equation, this would mean that the leading term in the splitting error would be

$$i\frac{k^2}{24}\left(V^{(4)}u + 4V^{(3)}u^{(1)} + 4V^{(2)}u^{(2)} - 4(V^{(1)})^2 u\right).$$

If the expression in (2.17) is greater than that in (2.18), this approach could lead to a somewhat smaller splitting error.

We now describe how $e^{\frac{1}{2}kB}$ is approximated; that is, we specify the scheme $C_B(k)$ used for the equation

$$iu_t = V(x)u. \tag{2.20}$$

Once we have discretized the x variable, (2.20) represents a system of uncoupled ordinary differential equations. $C_B(k)$ is an $m \times m$ diagonal matrix, with

$$[C_B(k)]_{j,j} = e^{-ikV(x_j)}. \tag{2.21}$$

For any Hermitian matrix H, the matrix $\exp(iH)$ is unitary. Hence $C_B(k)$ is a unitary matrix and conserves the L_2-norm. The scheme defined by (2.21) is exact (there is no truncation error) and we can write $C_B(k) = e^{kB}$ if the B in the exponent is interpreted to be the discretized form of B.

Definition 2.3.1 *Let $C_A(k)$ be a scheme for*

$$u_t = Au = -iu_{xx} \tag{2.22}$$

and let $C_B(k)$ be the scheme defined in (2.21). Then

$$C(k) \equiv C_B(\frac{k}{2})\ C_A(k)\ C_B(\frac{k}{2}). \tag{2.23}$$

is the resulting split-step method for the Schrödinger equation based on SP_2. We refer to (2.23) as the split-step "M" method, where "M" is the name of the scheme $C_A(k)$.

The truncation error of the split-step scheme is a combination of the splitting error and the truncation errors of the schemes $C_A(k)$ and $C_B(k)$. The following theorem is similar to a statement in [13] for a hyperbolic partial differential equation.

Theorem 2.3.1 *Consider the split-step scheme (2.23) for the Schrödinger equation based on SP_2. If $u(x,t)$ is a sufficiently smooth function of x and t that satisfies the Schrödinger equation, then the truncation error is*

$$\begin{aligned} \frac{1}{k}\left\{e^{k(A+B)} - C_B(\frac{k}{2})C_A(k)C_B(\frac{k}{2})\right\}u(x,t) &= SE_2(A,B,k)u(x,t) \\ &+ e^{\frac{k}{2}B}TE_A(k)e^{\frac{k}{2}B}u(x,t). \end{aligned}$$

Proof. We replace $C_A(k)$ by $e^{kA} - kTE_A(k)$ and $C_B(k)$ by e^{kB}, since there is no truncation error for $C_B(k)$.

$$\begin{aligned} & \frac{1}{k}\left\{e^{k(A+B)} - C_B(\frac{k}{2})C_A(k)C_B(\frac{k}{2})\right\}u(x,t) \\ = & \frac{1}{k}\left\{e^{k(A+B)} - e^{\frac{k}{2}B}\left(e^{A} - k\mathrm{TE}_A(k)\right)e^{\frac{k}{2}B}\right\}u(x,t) \\ = & \frac{1}{k}\left\{e^{k(A+B)} - \left(e^{\frac{k}{2}B}e^{A}e^{\frac{k}{2}B}\right)\right\}u(x,t) + e^{\frac{k}{2}B}\mathrm{TE}_A(k)e^{\frac{k}{2}B}u(x,t). \end{aligned}$$

■

Since $C_B(k)$ is unitary, the stability analysis of a split-step method for the Schrödinger equation reduces to showing the stability of $C_A(k)$. Such an analysis is often easy to carry out because we are approximating the solution of an equation with constant coefficients.

Theorem 2.3.2 *Let $C(k)$ be a split-step method (2.23)for the Schrödinger equation based on SP_2. If $C_A(k)$ is unconditionally contractive, then so is $C(k)$. If $C_A(k)$ is unconditionally unitary, then so is $C(k)$.*

Proof: Suppose that $C_A(k)$ is contractive. Then

$$\begin{aligned} ||C(k)|| &= ||C_B(\frac{k}{2})\, C_A(k)\, C_B(\frac{k}{2})|| \\ &= ||C_B(\frac{k}{2})||\, ||C_A(k)||\, ||C_B(\frac{k}{2})|| \qquad (\text{since} \quad C_B(\frac{k}{2}) \quad \text{is unitary}) \\ &= ||C_A(k)|| \leq 1, \end{aligned}$$

i.e., $C(k)$ is contractive. When $C_A(k)$ is unitary, the inequality on the last line becomes an equality and the second statement of the theorem follows. ■

2.4 The Split-step Fourier Method

The split-step Fourier (SSF) method, due to Hardin and Tappert [10], is the split-step method obtained by using a spectral method to approximate $\exp(kA)$.

Let $u(x)$ be a complex-valued function defined on the interval $(0, b)$ that vanishes at the end-points. We can approximate u by a sine series, truncated after the first m terms:

$$u(x) \approx U(x) = \sum_{s=1}^{m} \tilde{U}_s \sin(s\theta x), \qquad \text{where} \quad \theta \equiv \frac{\pi}{b}. \tag{2.24}$$

If the periodic, odd extension of u(x) has $j-1$ continuous derivatives and $u^{(j)}$ is square integrable, then the error incurred by truncating the sine series after m terms is $O(m^{-j})$ (see [9]). We can differentiate (2.24) term by term to obtain

$$\frac{\partial^2 U(x)}{\partial x^2} = \sum_{s=1}^{m} \left(-s^2\theta^2\right) \tilde{U}_s \sin(s\theta x). \tag{2.25}$$

We can use the discrete sine transform (DST[1]) to compute the coefficients $\tilde{U}_s$, or, having $\tilde{U}$ (the m-vector consisting of the Fourier coefficients), to evaluate U at the equispaced grid points. We denote these operations by $\mathcal{F}$ and $\mathcal{F}^{-1}$.

$$U^n \underset{\mathcal{F}^{-1}}{\overset{\mathcal{F}}{\rightleftarrows}} \tilde{U}^n, \tag{2.26}$$

where the superscript n denotes the n-th time level ($t = nk$).

[1]The $m \times m$ DST matrix $\mathcal{S}$ is given by $\mathcal{S}_{j,k} = \sin(\frac{jk\pi}{m+1})$. The normalization factor is given by the relation $\mathcal{S}^2 = (\frac{m+1}{2})\mathrm{I}$, where I is the identity matrix. For suitably chosen m (e.g. $m = 2^r - 1$), the DST can be implemented efficiently with the help of the Fast Fourier Transform (FFT) [5].

If we approximate $A = -iD_x^2$ by $\mathcal{F}^{-1}E\mathcal{F}$, then the corresponding approximation to $\exp(kA)$ defines the scheme

$$C_A(k)U = \mathcal{F}^{-1}e^{kE}\mathcal{F}U = \mathcal{F}^{-1}D\mathcal{F}U. \tag{2.27}$$

Since E is a diagonal matrix with $E_{s,s} = is^2\theta^2$, it follows that $\mathcal{D}$ is also diagonal, with

$$\mathcal{D}_{s,s} = e^{iks^2\theta^2}. \tag{2.28}$$

This choice of C_A in (2.23) gives the split-step Fourier (SSF) method [10]. If we use a different splitting, the resulting split-step Fourier method will be referred to as SSF/SP_1, SSF/SP_{2a}, etc.

We now consider the stability of the split-step Fourier methods.

Theorem 2.4.1 *The split-step Fourier methods SSF/SP_1 and SSF applied to Schrödinger's equation are unconditionally unitary for all real k. In particular, they are unconditionally stable.*

Proof. Since $\mathcal{F}$, $\mathcal{F}^{-1}$ and D are all unitary, $C_A(k)$ is also unitary for all real k, and the assertion follows from Theorem 2.3.2. ■

The number of Fourier modes m is equal to the number of equispaced grid points and must be large enough to resolve both the solution $u(x,.)$ *and* the potential V. Thus, even if the solution is smooth enough to be accurately represented with a small m (i.e. if it is made up of eigenfunctions corresponding to low eigenvalues of H), the potential might have steep gradients and this would force the use of a larger m. When m is sufficiently large, the total error is dominated by the splitting error, which means second order accuracy in time for SP_2. If the splitting error is small, then the SSF method is an efficient method for the numerical solution of the Schrödinger equation.

With the FFT package we have used (FFTPACK, [5]), it is convenient to use the real sine transform routine separately for the real and imaginary parts of the complex vector U to implement $\mathcal{F}$ and $\mathcal{F}^{-1}$. The cost of applying $\mathcal{F}$ or $\mathcal{F}^{-1}$ to a real vector of length m is roughly $(5/2)m\log_2 m$ real floating-point operations. The cost of applying the matrix $C_B(k/2)$ twice and D once is $18m$ floating-point operations. Hence the cost of one step of the SSF method is roughly $10m\log_2 m + 18m$. As pointed out by Strang [15], one can combine the last stage of a given step of (2.23) with the first stage of the next step, provided that the value of the solution is not required at that step. This reduces the number of applications of $C_B(k/2)$ to one per time step.

Chapter 3

Padé Approximations and Difference Schemes for the Schrödinger Equation

In this chapter, we develop a framework for constructing difference schemes for the Schrödinger equation based on Padé approximation. The framework is a synthesis of two separate components which have been used to discretize the time and space variables, respectively. The "Padé in time" component is based on approximations to the exponential function. The "Padé in space" component is based on approximations to a function that expresses the second derivative operator in terms of the centered difference operator, δ_x^2. We will describe these two components in the first two sections and then show how they can be combined to give difference schemes for equations of the form $u_t = \alpha u_{xx}$.

3.1 Padé Approximations to the Exponential Function

In this section, we consider time discretizations of equations of the form

$$\frac{\partial u}{\partial t} = iHu, \tag{3.1}$$

where H is an Hermitian operator. Assuming that H does not depend on time, we can write the evolution operator as an operator exponential to get

$$u(t+k) = e^{ikH}u(t).$$

We consider numerical schemes for (3.1) based on Padé approximations to the exponential function. This approach was used in [35] for parabolic partial differential equations and by a number of authors in the context of ordinary differential equations (e.g., [24], [36] [29]). This approach is distinct from that of [31], where Padé approximations to $(1-z)\log(1-z)$ and $-\log(1-z)$ are used to introduce families of explicit and implicit multistep schemes for spatial semidiscretizations of parabolic partial differential equations.

We begin by defining Padé approximation [35].

Table 3.1:

Some Padé approximations to e^z in the notation of (3.3).

(n,m)	$P_{n,m}(z)$	$Q_{n,m}(z)$	C_T	Accuracy
$(0,1)$	$1+z$	1	$\frac{1}{2}$	$O(z)$
$(1,0)$	1	$1-z$	$-\frac{1}{2}$	$O(z)$
$(1,1)$	$1+\frac{1}{2}z$	$1-\frac{1}{2}z$	$-\frac{1}{12}$	$O(z^2)$
$(2,2)$	$1+\frac{1}{2}z+\frac{1}{12}z^2$	$1-\frac{1}{2}z+\frac{1}{12}z^2$	$\frac{1}{720}$	$O(z^4)$

Definition 3.1.1 *Let $f : C \to C$ be an analytic function. Let m and n be non-negative integers. Then the (n,m) Padé approximant to f is the rational function $R_{n,m}(z)$ given by*

$$R_{n,m}(z) \equiv \frac{P_{n,m}(z)}{Q_{n,m}(z)},$$

where $P_{n,m}$ and $Q_{n,m}$ are polynomials of degree m and n, respectively, such that

$$\frac{d^j}{dz^j}f(0) = \frac{d^j}{dz^j}R_{n,m}(0), \text{ for } j = 0,1,...,m+n \quad \text{and} \quad Q_{n,m}(0) = 1.$$

We will sometimes refer to $R_{n,m}(z)$ as the (n,m) entry in the Padé table for f.

From now on, we will use $R_{n,m}(z) = P_{n,m}(z)/Q_{n,m}(z)$ to denote the (n,m) Padé approximant to e^z. Padé (1892) has given explicit expressions for the polynomials $P_{n,m}$ and $Q_{n,m}$:

$$\begin{aligned} P_{n,m}(z) &= \sum_{k=0}^{m} \frac{(n+m-k)!m!}{(n+m)!k!(m-k)!} z^k, \qquad \text{and} \\ Q_{n,m}(z) &= \sum_{k=0}^{n} \frac{(n+m-k)!n!}{(n+m)!k!(n-k)!} (-z)^k, \end{aligned} \tag{3.2}$$

(see Varga [35]). The error in this approximation can be written as

$$e^z - R_{n,m}(z) = C_T z^{n+m+1} + O(z^{n+m+2}) \quad \text{as} \quad z \to 0, \tag{3.3}$$

where the error constant C_T depends on n and m (see Table 3.1).

We now introduce the notion of i-stability. The following definition is motivated by the fact that the Schrödinger equation conserves the L_2-norm.

Definition 3.1.2 *A Padé approximation $R_{n,m}$ to e^z is called i-stable if $|R_{n,m}(iy)| \leq 1$, for all real y. $R_{n,m}$ is called conditionally i-stable if there exists a $y_o > 0$, such that $|R_{n,m}(iy)| \leq 1$, for all real y with $|y| \leq y_0$.*

Remark. The notion of i-stability has been used by a number of authors. In [32], it is referred to as "stability on the imaginary axis" (and attributed to H.-O. Kreiss) and in [37], it is called I-stability.

Lemma 3.1.1 *For all non-negative m and n and all real y, we have*

$$|R_{n,m}(iy)||R_{m,n}(iy)| = 1.$$

Proof. From the definition of $P_{n,m}$ and $Q_{n,m}$, it follows that $Q_{m,n}(z) = P_{n,m}(-z)$ or, equivalently, $P_{m,n}(z) = Q_{n,m}(-z)$.

For all non-negative m and n and all real y,

$$|R_{n,m}(iy)| = |R_{n,m}(-iy)|.$$

This follows from the fact that the coefficients of the polynomials $P_{n,m}$ and $Q_{n,m}$ are real and hence $P_{n,m}(-iy) = \overline{(P_{n,m}(iy))}$ and similarly for $Q_{n,m}$, and the lemma follows. ■

Thus, for $n \neq m$, precisely one of $R_{m,n}$ and $R_{n,m}$ is unconditionally i-unstable. The other entry is i-stable or conditionally i-stable.

The diagonal entries in the Padé table for e^z are particularly attractive for the Schrödinger equation due to the following property.

Lemma 3.1.2 *The (n,m) Padé approximant to e^z has modulus one on the entire imaginary axis if and only if $n = m$.*

Proof. If $n = m$, Lemma 3.1.1 says that $|R_{n,n}(iy)|^2 = 1$, for all real y.

If $n \neq m$, then $|R_{n,m}(iy)| = O(|y|^{m-n})$, as $|y| \to \infty$ and hence $R_{n,m}$ cannot have modulus one on the entire imaginary axis. ■

The notion of A-acceptability of Padé approximants to the exponential function has been studied in the context of A-stable schemes for ordinary differential equations, e.g., [24], [36] and is related to the property of i-stability.

Definition 3.1.3 *$R_{n,m}$ is called A-acceptable if $|R_{n,m}(z)| \leq 1$ for all z with $Re(z) \leq 0$.*

Clearly, A-acceptability implies i-stability.

A useful tool in the study of Padé approximations to the exponential function is Nørsett's "E-polynomial" [36]: $E(y) \equiv |Q_{n,m}(iy)|^2 - |P_{n,m}(iy)|^2$. When $m < n$,

$$E(y) = \left(\frac{m!}{(m+n)!}\right)^2 \sum_{r=r_0}^{n} \frac{(-1)^{n-r}}{(n-r)!} \left[\prod_{s=1}^{n-r}(n-s+1)(m+s)(r-m-s)\right] y^{2r}, \tag{3.4}$$

$$\text{where} \quad r_0 = \left\lfloor \frac{m+n+2}{2} \right\rfloor,$$

(see [36]).

Remark. $R_{n,m}$ is i-stable (conditionally i-stable) if and only if the corresponding E-polynomial is non-negative for all real y (for all real y with $|y| \leq y_0$ for some $y_0 > 0$).

The following theorem was conjectured by Ehle [24] and proved by Wanner, Hairer and Nørsett [36].

Theorem 3.1.1 *$R_{n,m}$ is A-acceptable if and only if $m \leq n \leq m+2$.*

We will show that the converse is not true.

Theorem 3.1.2 *There exist i-stable entries in the Padé table for e^z that are not A-acceptable.*

Proof. We will show that $R_{6,0}$ is i-stable. From Theorem 3.1.1, we know that $R_{6,0}$ is not A-acceptable. Substituting $n = 6$ and $m = 0$ in (3.4), we get $r_0 = 4$ and

$$E(y) = \left[\frac{1}{6!}\right]^2 y^8 \left[180 - 24y^2 + y^4\right].$$

It is easily verified that the quadratic expression in y^2 in the brackets is positive for $y = 0$ and has no real roots. Thus, $E(y)$ is positive for all real y, and $R_{6,0}$ is i-stable. ■

From Theorem 3.1.1, we know that the main diagonal and the first two subdiagonals of the Padé table are i-stable. In [29], in a partial proof of the Ehle conjecture, it is shown that for $n \geq m+1$, $R_{n,m}$ is unconditionally i-unstable for $n - m \equiv 0 \pmod 4$ and $n - m \equiv 3 \pmod 4$ (and hence not A-acceptable). The following result is the "positive" form of this statement and includes the super-diagonals of the Padé table for e^z.

Theorem 3.1.3 *For $m \neq n$, $R_{n,m}$ is conditionally i-stable if, and only if, $n - m \equiv 1$ or $2 \pmod 4$, for $m < n$ and $n - m \equiv 0$ or $3 \pmod 4$, for $m > n$. (In this statement, the i-stable entries are regarded as a subset of the set of conditionally i-stable entries.)*

Proof. We will first prove the statement for the case $m < n$, using (3.4). We can write

$$E(y) = y^{2r_0}\left(\alpha_0 + \alpha_1 y^2 + \ldots + \alpha_{n-r_0} y^{2(n-r_0)}\right),$$

where

$$\alpha_0 = \left[\frac{m!}{(m+n)!}\right]^2 \frac{(-1)^{n-r_0}}{(n-r_0)!} \prod_{s=1}^{n-r_0} (n-s+1)(m+s)(r_0-m-s).$$

Clearly, $R_{n,m}$ is conditionally i-stable if and only if $\alpha_0 > 0$. From the definition of r_0, we have that $0 \leq m < r_0 \leq n$ and the three factors in the product are positive for all integers s in the specified range. Thus, $\text{sign}(\alpha_0) = (-1)^{n-r_0}$. Thus, $\alpha_0 > 0$, if and only if $n - r_0 = n - \lfloor \frac{n+m+2}{2} \rfloor$ is even. If $n + m = 2r$, then $n - \lfloor \frac{n+m+2}{2} \rfloor = \frac{n-m}{2} - 1$ is even if and only if $\frac{n-m}{2}$ is odd. If $n + m = 2r + 1$, then $n - \lfloor \frac{n+m+2}{2} \rfloor = \frac{n-m-1}{2}$ is even if and only if $\lceil \frac{n-m}{2} \rceil$ is odd. Thus, for $m < n$, $\alpha_0 > 0$ if and only if $\lceil \frac{n-m}{2} \rceil$ is odd. If we write $n - m \equiv j \pmod 4$, or $n - m = 4s + j$, for some non-negative integer s, then $\lceil \frac{n-m}{2} \rceil = \lceil 2s + \frac{j}{2} \rceil$ and this expression is odd, if and only if $j = 1$ or 2. This proves the assertion for $m < n$.

The result for the upper triangle in the Padé table follows from Lemma 3.1.1, which implies that precisely one of (n, m) and (m, n) is conditionally i-stable when $n \neq m$. ■

3.2 Padé Approximations to the Second Derivative Operator

In this section, we review an approach for constructing approximations to the second derivative operator (in space) using centered finite difference operators that is based on Padé approximations. This approach was used by Guardiola and Ros [26] to construct high-accuracy difference schemes for the time-*independent* Schrödinger equation. A similar approach was used by Henrici [27] to construct higher-order methods for Poisson's equation in a hypercube.

The finite difference operator δ_x defined in Chapter 1 can be written formally as

$$\delta_x u = \left(e^{\frac{h}{2}D_x} - e^{-\frac{h}{2}D_x}\right) u$$

or

$$\frac{\delta_x}{2} = \sinh\left(\frac{h}{2}D_x\right)$$

which implies that ([28])

$$D_x^2 = \frac{4}{h^2}\left[\sinh^{-1}\left(\frac{\delta_x}{2}\right)\right]^2. \tag{3.5}$$

Let $s_0(z) \equiv \sinh^{-1}(z)$ and let $s_1(z) \equiv 4\left[s_0(z/2)\right]^2$. The Taylor series for $s_0(z)$ about the origin is given by

$$s_0(z) = z\sum_{j=0}^{\infty} b_j z^{2j}, \quad \text{with} \quad b_0 = 1, \quad b_{j+1} = -b_j\frac{(2j+1)^2}{(2j+2)(2j+3)},$$

with radius of convergence equal to 1 [17]. Equation (3.5) can now be written as $h^2 D_x^2 = s_1(\delta_x)$. The key idea to using Padé approximations to approximate D_x^2 is to recognize that $s_1(z)$ is an even function, i.e. its Taylor series expansion about the origin contains only even powers of z. Let $s_1(z) = \sum_{j=1}^{\infty} c_j z^{2j}$ be the power series expansion of s_1. Let $S(z) \equiv \sum_{j=1}^{\infty} c_j z^j$, i.e., $S(z^2) = s_1(z)$. Then, (3.5) can be written as

$$h^2 D_x^2 = S(\delta_x^2). \tag{3.6}$$

The purpose of introducing the function $S(z)$ is to be able to consider Padé approximation in the usual sense. The first few terms in the Taylor series expansion for $S(z)$ are

$$S(z) = z - \frac{1}{12}z^2 + \frac{1}{90}z^3 - \frac{1}{560}z^4 + \frac{1}{3150}z^5 - \frac{1}{16632}z^6 + \frac{1}{84084}z^7 - \ldots \tag{3.7}$$

These coefficients were given in [26] in an expansion of D_x^2 in terms of δ_x^2.

Definition 3.2.1 *Let $\mu \geq 1$ and $\nu \geq 0$ be integers and let $\tilde{P}_{\nu,\mu}$ and $\tilde{Q}_{\nu,\mu}$ be polynomials of degree μ and ν, respectively, such that $\tilde{R}_{\nu,\mu} = \tilde{P}_{\nu,\mu}/\tilde{Q}_{\nu,\mu}$ is the (ν,μ) Padé approximation to $S(z)$, with $\tilde{Q}_{\nu,\mu}(0) = 1$ and $\tilde{P}_{\nu,\mu}(0) = 0$.*

Table 3.2:

Some Padé approximations to $S(z)$ in the notation of (3.8).

(ν,μ)	$\tilde{P}_{(\nu,\mu)}(z)$	$\tilde{Q}_{(\nu,\mu)}(z)$	C_S	Accuracy
$(0,1)*$	z	1	$-1/12$	$O(h^2)$
$(0,2)*$	$z-\frac{1}{12}z^2$	1	$1/90$	$O(h^4)$
$(0,3)*$	$z-\frac{1}{12}z^2+\frac{1}{90}z^3$	1	$-1/560$	$O(h^6)$
$(0,4)$	$z-\frac{1}{12}z^2+\frac{1}{90}z^3-\frac{1}{560}z^4$	1	$1/3150$	$O(h^8)$
$(1,1)*$	z	$1+\frac{1}{12}z$	$1/240$	$O(h^4)$
$(1,2)$	$z+\frac{1}{20}z^2$	$1+\frac{2}{15}z$	$-23/75600$	$O(h^6)$
$(2,1)*$	z	$1+\frac{1}{12}z-\frac{1}{240}z^2$	$-31/60480$	$O(h^6)$
$(2,2)*$	$z+\frac{31}{252}z^2$	$1+\frac{13}{63}z+\frac{23}{3780}z^2$	$79/4762800$	$O(h^8)$

The error in this approximation can be written as

$$S(z)-\frac{\tilde{P}_{\nu,\mu}(z)}{\tilde{Q}_{\nu,\mu}(z)}=C_S z^{\mu+\nu+1}+O(z^{\mu+\nu+2}),\quad \text{as}\quad z\to 0, \tag{3.8}$$

where the error constant C_S depends on μ and ν. From (3.5) and (3.8) and the fact that $\delta_x^2=h^2D_x^2+O(h^4)$, it follows that

$$h^2\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]D_x^2u(x)=\tilde{P}_{\nu,\mu}(\delta_x^2)u(x)+C_Sh^{2(\mu+\nu+1)}D_x^{2(\mu+\nu+1)}u(x)+O(h^{2(\mu+\nu+2)}). \tag{3.9}$$

Table 3.2 gives a number of Padé approximations to the second derivative operator of the form (3.9). These entries were given in [26]; the entries with asterisks were given earlier in [21] (Appendix, Table III).

The discretization that the $(1,1)$ Padé approximant leads to has been used by a number of authors to construct difference schemes for various partial or ordinary differential equations, but with a derivation other than Padé approximation.

- Numerov, 1933 (cited in [26]): $O(h^4)$ method (the Numerov method) for two-point boundary-value problems.
- Collatz, 1950 [21]: $O(h^4)$ Mehrstellen methods for differential equations.
- Crandall, 1955 [22] and Douglas, 1956 [23]: $O(k^2,h^4)$ method for the heat equation in one space dimension.
- Fairweather and Mitchell, 1964, 1965 [25], [34]: $O(k^2,h^4)$ ADI method for the heat equation in two space dimensions.

- Ciment and Leventhal, 1975 [20]: An operator compact implicit (OCI) method for the wave equation. The authors attribute the Padé $(1,1)$ approximant to H.-O. Kreiss.

- Lynch and Rice, 1978 [33], Boisvert 1981 [18]: The Higher Order Differences and Identity Expansions or HODIE method for elliptic partial differential equations. (Their "Identity Expansions" correspond to the $\tilde{Q}_{\nu,\mu}(\delta_x^2)$ in the Padé approximants.) The relation between the HODIE methods and the Mehrstellen methods is discussed in [33].

In [30], (diagonal) Padé approximations to $(ln(z))^2$ are used to construct finite difference approximations to the second derivative operator in terms of the shift operator E, given by $E \cdot u(x) = u(x+h)$. The approximations obtained by this procedure are equivalent to the Padé approximations to the second derivative operator in terms of the operator δ_x^2 that we have discussed in this section.

3.3 A General Framework for Constructing Difference Schemes Based on Padé Approximations in Time and Space

We now introduce a framework for constructing difference schemes for a time-dependent equation of the form

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2}, \quad \text{for} \quad x \in (0,b), \tag{3.10}$$

$$u(0,t) = u(b,t) = 0, \qquad u(x,0) = \phi(x).$$

where α is a constant. Equations of this type include the Schrödinger equation with zero potential ($\alpha = \pm i$) and the diffusion equation (α real, positive). This framework combines the techniques described in the preceding sections.

The idea is the following. We first use a Padé approximant $R_{n,m}(z)$ to e^z to discretize the time variable in (3.10), replacing $e^{\alpha k D_x^2}$ by $[Q_{n,m}(\alpha k D_x^2)]^{-1}[P_{n,m}(\alpha k D_x^2)]$ to get

$$Q_{n,m}(\alpha k D_x^2) U^{n+1} = P_{n,m}(\alpha k D_x^2) U^n. \tag{3.11}$$

We next pick a Padé approximant to D_x^2

$$D_x^2 \approx \frac{1}{h^2}\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{-1} \tilde{P}_{\nu,\mu}(\delta_x^2)$$

and use this as the space discretization in (3.11), which can now be written as

$$Q_{n,m}\left(\alpha \frac{k}{h^2}\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{-1} \tilde{P}_{\nu,\mu}(\delta_x^2)\right) U^{n+1} = P_{n,m}\left(\alpha \frac{k}{h^2}\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{-1} \tilde{P}_{\nu,\mu}(\delta_x^2)\right) U^n.$$

The final step is to multiply both sides by the highest power of $\tilde{Q}_{\nu,\mu}(\delta_x^2)$ occurring on either side to obtain

$$\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{\max(n,m)} Q_{n,m}\left(\alpha r\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{-1}\tilde{P}_{\nu,\mu}(\delta_x^2)\right)U^{n+1}$$

$$= \left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{\max(n,m)} P_{n,m}\left(\alpha r\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)\right]^{-1}\tilde{P}_{\nu,\mu}(\delta_x^2)\right)U^{n}. \tag{3.12}$$

Definition 3.3.1 *A Padé $(n,m;\nu,\mu)$ difference scheme for (3.10) is any member of the four parameter family of schemes defined by (3.12)*

The Padé $(n,m;\nu,\mu)$ scheme (3.12) can be written in the form

$$U^{n+1} = C(k)U^n, \quad \text{with} \quad C(k) = R_{n,m}\left(\alpha r\tilde{R}_{\nu,\mu}(\delta_x^2)\right). \tag{3.13}$$

An important property of Padé $(n,m;\nu,\mu)$ difference schemes for (3.10) is that the truncation error can be read off directly from the errors in the underlying Padé approximations.

Theorem 3.3.1 *Let C_T be the error constant for the Padé (n,m) approximation to e^z and let C_S be the error constant for the Padé (ν,μ) approximation to $S(z)$ (see (3.3) and (3.8)). Then the principal terms in the local truncation error of the Padé $(n,m;\nu,\mu)$ difference scheme (3.12) are*

$$\left[C_T k^{m+n}\left(\frac{\partial}{\partial t}\right)^{m+n+1} + \alpha C_S h^{2(\mu+\nu)}\left(\frac{\partial^2}{\partial x^2}\right)^{\mu+\nu+1}\right]u(x,t), \tag{3.14}$$

for a $u(x,t)$ that satisfies the partial differential equation (3.10) and is $m+n+1$ times differentiable with respect to t and $2(\mu+\nu+1)$ times differentiable with respect to x.

Proof. From (3.13), we have

$$C(k)u(x,t) = R_{n,m}\left(\alpha r\tilde{R}_{\nu,\mu}(\delta_x^2)\right)u(x,t).$$

Since

$$\tilde{R}_{\nu,\mu}(\delta_x^2) = h^2D_x^2 - C_S(h^2D_x^2)^{\nu+\mu+1} + O(h^{2(\nu+\mu+2)}),$$

we have

$$C(k)u(x,t) = R_{n,m}\left(\alpha rh^2D_x^2 - \alpha rC_S(h^2D_x^2)^{\nu+\mu+1} + O(rh^{2(\nu+\mu+2)})\right)u(x,t).$$

We now use the relation

$$R_{n,m}(X) = e^X - C_TX^{n+m+1} + O(X^{n+m+2})$$

and obtain

$$\begin{aligned} C(k)u(x,t) &= \left[e^{\{\alpha k D_x^2 - \alpha C_S k h^{2(\nu+\mu)} D_x^{2(\nu+\mu+1)} + O(k h^{2(\nu+\mu+1)})\}}\right] u(x,t) \\ &- C_T \left[\alpha k D_x^2 - \alpha C_S k h^{2(\nu+\mu)} D_x^{2(\nu+\mu+1)} + O(k h^{2(\nu+\mu+1)})\right]^{n+m+1} u(x,t) \\ &+ O(k^{n+m+2}) \\ &= \left[e^{\alpha k D_x^2} - \alpha C_S k h^{2(\nu+\mu)} D_x^{2(\nu+\mu+1)}\right] u(x,t) + O(k h^{2(\nu+\mu+1)}) \\ &- C_T k^{n+m+1} \left(\frac{\partial}{\partial t}\right)^{n+m+1} + O(k^{n+m+1} h^{2(\nu+\mu)}) \\ &+ O(k^{n+m+2}), \end{aligned}$$

for sufficiently differentiable $u(x,t)$ that satisfy (3.10). We re-write the above as

$$\begin{aligned} C(k)u(x,t) &= \left\{e^{\alpha k D_x^2} - \alpha C_S k h^{2(\nu+\mu)} D_x^{2(\nu+\mu+1)} - C_T k^{n+m+1} \left(\frac{\partial}{\partial t}\right)^{n+m+1}\right\} u(x,t) \\ &+ O(k^{n+m+2}, h^{2(\nu+\mu+1)}). \end{aligned}$$

Since $u(x,t)$ is assumed to satisfy the differential equation (3.1), the truncation error can be written as

$$\left(\frac{e^{\alpha k D_x^2} - C(k)}{k}\right) u(x,t)$$

and therefore the principal part of the local truncation error for the Padé $(n,m;\nu,\mu)$ difference scheme is (3.14). ■

In practice, it is clearly not feasible to use Padè schemes of arbitrarily high order. The use of very high order schemes may be restricted not only by the differentiability requirements but also by the boundary behavior of the solution. We now give some examples of Padé schemes that are suitable for practical computations.

Example 1. The Padé $(0,1;0,1)$ difference scheme with:

$$P_{0,1}(z) = 1 + z, \qquad Q_{0,1}(z) = 1,$$

$$\tilde{P}_{0,1}(z) = z, \qquad \tilde{Q}_{0,1}(z) = 1$$

reduces to

$$U^{n+1} = \left[I + \alpha r \delta_x^2\right] U^n, \tag{3.15}$$

which is the explicit Euler scheme with the standard second-order difference approximation to D_x^2. The work at each time step is a (tridiagonal) matrix-vector product. The principal part of the truncation error is

$$\left\{\frac{1}{2} k \left(\frac{\partial}{\partial t}\right)^2 - \alpha \frac{1}{12} h^2 \left(\frac{\partial}{\partial x}\right)^4\right\} u(x,t) = O(k, h^2).$$

Similarly, the Padé $(1,0;0,1)$ scheme is the backward Euler method for (3.10).

Example 2. The Padé $(1,1;0,1)$ difference scheme with:

$$P_{1,1}(z) = 1 + \frac{1}{2}z, \qquad Q_{1,1}(z) = 1 - \frac{1}{2}z,$$

$$\tilde{P}_{0,1}(z) = z, \qquad \tilde{Q}_{0,1}(z) = 1$$

reduces to

$$\left[I - \frac{\alpha}{2} r \delta_x^2\right] U^{n+1} = \left[I + \frac{\alpha}{2} r \delta_x^2\right] U^n. \tag{3.16}$$

This is the Crank-Nicolson scheme. The work at each time step is a (tridiagonal) matrix-vector product and a tridiagonal solve. The principal part of the truncation error is

$$\left\{ -\frac{1}{12}k^2 \left(\frac{\partial}{\partial t}\right)^3 - \alpha \frac{1}{12} h^2 \left(\frac{\partial}{\partial x}\right)^4 \right\} u(x,t) = O(k^2, h^2).$$

Example 3. The Padé $(1,1;1,1)$ difference scheme with:

$$P_{1,1}(z) = 1 + \frac{1}{2}z, \qquad Q_{1,1}(z) = 1 - \frac{1}{2}z,$$

$$\tilde{P}_{1,1}(z) = z, \qquad \tilde{Q}_{1,1}(z) = 1 + \frac{1}{12}z$$

gives

$$\left[I - \frac{\alpha}{2} r \left(I + \frac{1}{12}\delta_x^2\right)^{-1} \delta_x^2\right] U^{n+1} = \left[I + \frac{\alpha}{2} r \left(I + \frac{1}{12}\delta_x^2\right)^{-1} \delta_x^2\right] U^n,$$

which, after multiplying both sides by $\left(I + \frac{1}{12}\delta_x^2\right)$ simplifies to

$$\left[I + (\frac{1}{12} - \frac{\alpha}{2} r)\delta_x^2\right] U^{n+1} = \left[I + (\frac{1}{12} + \frac{\alpha}{2} r)\delta_x^2\right] U^n. \tag{3.17}$$

This scheme was devised independently by Crandall [22] and Douglas [23] for the diffusion equation, but with a different derivation. The work at each time step is the same as that for the Crank-Nicolson scheme. The principal part of the truncation error is

$$\left\{ -\frac{1}{12}k^2 \left(\frac{\partial}{\partial t}\right)^3 + \alpha \frac{1}{240} h^4 \left(\frac{\partial}{\partial x}\right)^6 \right\} u(x,t) = O(k^2, h^4).$$

An interesting property of the Crandall-Douglas scheme for real $\alpha > 0$ (the parabolic case) is that the choice $r = k/h^2 = \sqrt{\alpha/20})$ eliminates the k^2 and h^4 terms in the truncation error. This does not apply for $\alpha = \pm i$ (the Schrödinger equation).

Example 4. The Padé $(1,1;2,2)$ difference scheme with:

$$P_{1,1}(z) = 1 + \frac{1}{2}z, \qquad Q_{1,1}(z) = 1 - \frac{1}{2}z,$$

$$\tilde{P}_{2,2}(z) = z + az^2, \qquad \tilde{Q}_{2,2}(z) = 1 + bz + cz^2,$$

where the values of a, b and c are given in Table 3.2, reduces to

$$\left[I + (b - \frac{\alpha}{2}r)\delta_x^2 + (c - \frac{\alpha}{2}ra)\delta_x^4\right] U^{n+1} = \left[I + (b + \frac{\alpha}{2}r)\delta_x^2 + (c + \frac{\alpha}{2}ra)\delta_x^4\right] U^n \tag{3.18}$$

The work at each time step is a penta-diagonal matrix-vector product and the solution of a penta-diagonal system of linear equations. The principal part of the truncation error is

$$\left\{-\frac{1}{12}k^2\left(\frac{\partial}{\partial t}\right)^3 + \alpha\frac{79}{4,762,800}h^8\left(\frac{\partial}{\partial x}\right)^{10}\right\} u(x,t) = O(k^2, h^8).$$

We will refer to this scheme as FD(2,8), to reflect the values of the exponents of k and h, respectively.

Example 5. The Padé $(2,2;1,1)$ difference scheme with:

$$P_{2,2}(z) = 1 + \frac{1}{2}z + \frac{1}{12}z^2, \qquad Q_{2,2}(z) = 1 - \frac{1}{2}z + \frac{1}{12}z^2,$$

$$\tilde{P}_{1,1}(z) = z, \qquad \tilde{Q}_{1,1}(z) = 1 + \frac{1}{12}z$$

reduces to

$$\begin{aligned} &\left[I + (\frac{1}{6} + \frac{\alpha r}{2})\delta_x^2 + (\frac{1}{144} + \frac{\alpha r}{2} - \frac{r^2}{12})\delta_x^4\right] U^{n+1} \\ = &\left[I + (\frac{1}{6} - \frac{\alpha r}{2})\delta_x^2 + (\frac{1}{144} - \frac{\alpha r}{2} - \frac{r^2}{12})\delta_x^4\right] U^n. \end{aligned} \tag{3.19}$$

The work at each time step is the same as for the Padé $(1,1;2,2)$ scheme of the previous example. However, the temporal and spatial errors have different orders, since the principal part of the truncation error is

$$\left\{\frac{1}{720}k^4\left(\frac{\partial}{\partial t}\right)^5 + \alpha\frac{1}{240}h^4\left(\frac{\partial}{\partial x}\right)^6\right\} u(x,t) = O(k^4, h^4).$$

We will refer to this scheme as FD(4,4)

The stability properties of the family of schemes (3.12) for spatial semidiscretizations of the diffusion equation ($\alpha = 1$) were investigated by Varga [35]. His main stability result is that time-differencing schemes for the diffusion equation based on Padé approximations to the exponential function are unconditionally stable if and only if $m \leq n$. We now prove a stability theorem for the Schrödinger equation ($\alpha = \pm i$).

Theorem 3.3.2 *Let $\alpha = i$ in (3.10). Then the Padé $(n, m; \nu, \mu)$ difference scheme (3.12) is contractive if $R_{n,m}$ is i-stable. The scheme is unitary if $n = m$.*

Proof. If T is a Hermitian matrix, then so is $\tilde{R}_{\nu,\mu}(T)$. If $R_{n,m}$ is i-stable and H is a Hermitian matrix, then $||R_{n,m}(iH)|| \leq 1$. We have equality precisely for the diagonal entries (Lemma 3.1.2). Thus $||R_{n,m}(ir\tilde{R}_{\nu,\mu}(T))|| \leq 1$, where T is the matrix representing δ_x^2 and (n, m) is an i-stable entry in the Padé table for e^z. ■

The Padé framework we have developed here for (3.10) can also be applied to the wave equation,

$$\frac{\partial^2 u(x,t)}{\partial t^2} = c^2 \frac{\partial^2 u(x,t)}{\partial x^2}. \tag{3.20}$$

We proceed by selecting two Padé approximations, $\tilde{R}_{n,m}$ and $\tilde{R}_{\nu,\mu}$, to the function $S(x)$ and use them to approximate the second derivatives in (3.20) with respect to t and x, respectively. Thus,

$$u_{tt}(x_j, t_n) \equiv \frac{1}{k^2} \frac{\tilde{P}_{n,m}(\delta_t^2)}{\tilde{Q}_{n,m}(\delta_t^2)} U_j^n, \qquad u_{xx}(x_j, t_n) \equiv \frac{1}{k^2} \frac{\tilde{P}_{\nu,\mu}(\delta_x^2)}{\tilde{Q}_{\nu,\mu}(\delta_x^2)} U_j^n,$$

which yields the following Padé $(n, m; \nu, \mu)$ scheme for (3.20):

$$\tilde{Q}_{\nu,\mu}(\delta_x^2)\tilde{P}_{n,m}(\delta_t^2)U_j^n = \tilde{Q}_{n,m}(\delta_t^2)\tilde{P}_{\nu,\mu}(\delta_x^2)U_j^n. \tag{3.21}$$

In conclusion to this discussion of Padé methods, we note that Padé approximations have the important advantage that they maximize the order of accuracy of the numerical schemes. However, this is a statement about the asymptotic behavior of the schemes as k, $h \to 0$. Rational approximations other than Padé may lead to schemes that achieve a given accuracy at a lower cost (see, for example, the discussion in [35] on Chebyshev rational approximations).

Chapter 4

Some Methods for the Schrödinger Equation in One Space Dimension

In this chapter, we define a number of new schemes for the Schrödinger equation in one space dimension. We combine the split-step approach and the Padé framework that were discussed in the preceding chapters. We also introduce a split-step variant of an explicit scheme and a family of three-level schemes that use the Padé framework.

4.1 Split-step, Implicit Finite Difference Schemes

In this section, we will use the apparatus developed in earlier chapters to derive a family of split-step, implicit finite difference schemes for the Schrödinger equation. All the methods developed in this section are based on the split-step approach. The idea is to choose $C_A(k)$ in (2.23) to be a Padé $(n,m;\nu,\mu)$ difference scheme for

$$i\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}. \tag{4.1}$$

A method of this form (the split-step Crank-Nicolson method) was applied to the nonlinear Schrödinger equation in [53].

Definition 4.1.1 *A split-step Pade $(n,m;\nu,\mu)$ method is a split-step method (Definition 2.3.1, page 11), where the scheme $C_A(k)$ is chosen to be the Pade $(n,m;\nu,\mu)$ difference scheme (Definition 3.3.1, page 21). (The treatment of the potential is the same for all methods in this family).*

To illustrate this approach, consider the sub-family of split-step Padé $(1,1;\nu,\mu)$ schemes, i.e., split-step schemes that use Crank-Nicolson in time and Padé (ν,μ) in space for the $C_A(k)$ part. These schemes have the form

$$U^{n+1} = C_B(\frac{k}{2})\ C_A(k)\ C_B(\frac{k}{2})U^n,$$

where

$$\left(C_B(\frac{k}{2})U^n\right)_j = e^{-ikV(x_j)}U_j^n,$$

and where $C_A(k)$ can be expressed as

$$\left[\tilde{Q}_{\nu,\mu}(\delta_x^2) + i\frac{r}{2}\tilde{P}_{\nu,\mu}(\delta_x^2)\right]U^{n+1} = \left[\tilde{Q}_{\nu,\mu}(\delta_x^2) - i\frac{r}{2}\tilde{P}_{\nu,\mu}(\delta_x^2)\right]U^n. \tag{4.2}$$

Here $\tilde{P}_{\nu,\mu}/\tilde{Q}_{\nu,\mu}$ is the (ν,μ) Padé approximant to the second derivative operator. For $(\nu,\mu) = (0,1)$, $(1,1)$ and $(2,2)$, the resulting split-step, implicit finite difference schemes will be referred to as the split-step Crank-Nicolson (SS-CN), the split-step Crandall-Douglas (SS-CD) and the split-step FD(2,8) methods, respectively (see Examples 2, 3 and 4 in Section 3.3).

We now consider the stability of the proposed schemes. The main stability result is an immediate consequence of Theorem 2.3.2 and Theorem 3.3.2.

Theorem 4.1.1 *The split-step Padé $(n,m;\nu,\mu)$ difference schemes of Definition 4.1.1 are unconditionally contractive if and only if the (n,m) entry in the Padé table for e^z is i-stable. The scheme is unconditionally unitary if and only if $n = m$.*

The matrix on the left-hand side of the linear system (4.2) does not depend on the potential V and does not change with time. This simplifies the linear algebra since the matrix needs to be factored just once and at each time step we need to perform just the forward and backward solve. For an $m \times m$ banded matrix with semi-bandwidth b, the cost of the LU factorization without pivoting is $O(b^2m)$ and the cost of the forward- or back-solve is $O(bm)$, i.e., linear in the semi-bandwidth. Thus the cost of the solve phase for a pentadiagonal system is twice that for a tridiagonal system.

An alternative to solving a linear system of semi-bandwidth b is to express the matrix in (4.2) as the product of b tridiagonal matrices [52]. This will be referred to as the *factorization approach* and must be distinguished from the LU factorization performed in Gaussian elimination[1] . The factorization approach reduces the complexity of the computing the LU decomposition from $O(b^2m)$ to $O(bm)$ since we go from solving one system of semi-bandwidth b to solving b tridiagonal systems. However, since the split-step Padé $(n,m;\nu,\mu)$ methods only require a back-solve at each time step, the cost of implementing these schemes with and without factoring is essentially the same. For higher-order schemes, the factorization approach may be preferable due to its simplicity (no higher derivatives need to be discretized).

The width of the stencil of the operator on the left-hand side of (4.2) is $\max\{2\mu+1, 2\nu+1\}$. The order of the truncation error of the Padé (μ,ν) approximation is $2(\mu+\nu)$. Putting these facts together, we have

Theorem 4.1.2 *The Padé approximations of Definition 3.2.1 that give the schemes of the form (4.2) with the highest (local) order of accuracy for a given stencil width are the Padé (μ,μ) approximations.*

Table 4.1:

Some doubly-diagonal Padé methods for the Schrödinger equation. The order of accuracy of each method is given, along with the value of b, the semi-bandwidth of the matrices.

		Padé in space			
		$(1,1)$	$(2,2)$	$(3,3)$	$(4,4)$
Padé in time	$(1,1)$	$O(k^2,h^4)$ 1	$O(k^2,h^8)$ 2	$O(k^2,h^{12})$ 3	$O(k^2,h^{16})$ 4
	$(2,2)$	$O(k^4,h^4)$ 2	$O(k^4,h^8)$ 4	$O(k^4,h^{12})$ 6	$O(k^4,h^{16})$ 8
	$(3,3)$	$O(k^6,h^4)$ 3	$O(k^6,h^8)$ 6	$O(k^6,h^{12})$ 9	$O(k^6,h^{16})$ 12
	$(4,4)$	$O(k^8,h^4)$ 4	$O(k^8,h^8)$ 8	$O(k^8,h^{12})$ 12	$O(k^8,h^{16})$ 16

A similar statement holds for schemes based on other Padé approximations in time.

The accuracy and work requirements for the Padé schemes can be read off directly from the values of n, m, ν and μ. In Table 4.1, we have listed the accuracies of a special subset of the Padé difference methods, namely the *doubly-diagonal* schemes, i.e., schemes based on diagonal Padé approximations in both time and space (the Padé $(n,n;\mu,\mu)$ schemes). The importance of this subset is based on Theorems 4.1.1 and 4.1.2 which imply that Padé schemes that are diagonal in time are norm-conserving and schemes that also are diagonal in space maximize the spatial order of accuracy for a given stencil width. Table 4.1 also gives the semi-bandwidth of the matrices arising in each method. The table illustrates a general property of the Padé family of schemes, namely, that, for a given amount of work per time step, there is the option of trading off temporal accuracy for spatial accuracy. In the factorization approach of [52], each factor (a tridiagonal matrix) contributes two additional orders of accuracy in time. With our Padé in time and space framework, if we use the doubly-diagonal Padé schemes, the factorization approach can be extended so that each tridiagonal factor contributes either two additional orders of accuracy in time or four orders of accuracy in space.

We now discuss the condition numbers of the matrices in the Padé schemes for the Schrödinger equation. Let

$$A_{\nu,\mu}(T,r) = \tilde{Q}_{\nu,\mu}(T) + i\frac{r}{2}\tilde{P}_{\nu,\mu}(T). \tag{4.3}$$

[1]For the diffusion equation, the use of the factorization approach necessitates the use of complex arithmetic, partly off-setting the gains. For the Schrödinger equation, this is not an issue, since complex arithmetic is used anyway.

For each (ν,μ) pair shown in Table 3.2 and a fixed r, the spectrum of the matrix $A_{\nu,\mu}(T,r)$ lies outside a neighborhood of the origin that does not depend on m. This can be verified numerically by examining the image of the interval $[-4,0]$ under the mapping $z \mapsto A_{\nu,\mu}(z,r)$. As a result, asymptotically, the condition number of these matrices depends on their order m only indirectly, through the mesh ratio r. (We do not know if this property holds for all (μ,ν) pairs.)

In Figure 4.1, we have plotted the condition number of $\mathrm{cond}(A_{\nu,\mu}(T,r))$ as a function of r, for $m = 500$ and $(\nu,\mu) = (0,1)$, $(1,1)$ and $(2,2)$. For small r, $\mathrm{cond}(A_{\nu,\mu}(T,r)) \approx \mathrm{cond}(\tilde{Q}_{\nu,\mu})$. As r increases, the condition number eventually becomes linear in r. For a fixed m, this says that the condition number of $A_{\nu,\mu}(T,r)$ grows linearly with the step-size k. Interestingly, the higher-order method $(\nu,\mu) = (2,2)$ is somewhat better conditioned than Crank-Nicolson and Crandall-Douglas for $r > 1$. The truncation error for Crank-Nicolson time-differencing grows quadratically with k. Both these effects combine to limit the time step that can be used in practice, even though there is no stability limit on r.

The Padé $(1,1;\nu,\mu)$ schemes discussed above have the same order of accuracy in time (second) as the underlying splitting and it may appear that there is nothing to be gained by using a higher-order in time method for $C_A(k)$. However, if the potential is smooth, the constant in front of the k^2 term in the splitting error may be much smaller than the constant for Crank-Nicolson. If this is the case, it makes sense to use a higher-order in time method for (4.1), so that the overall error in time is determined by the splitting error. One possibility is to choose a member of the Padé $(2,2;\nu,\mu)$ family of difference schemes:

$$\begin{aligned}&\left[\tilde{Q}_{\nu,\mu}(\delta_x^2)^2 + i\frac{r}{2}\tilde{Q}_{\nu,\mu}(\delta_x^2)\tilde{P}_{\nu,\mu}(\delta_x^2) - i\frac{r^2}{12}\tilde{P}_{\nu,\mu}(\delta_x^2)^2\right]U^{n+1}\\ &= \left[\tilde{Q}_{\nu,\mu}(\delta_x^2)^2 - i\frac{r}{2}\tilde{Q}_{\nu,\mu}(\delta_x^2)\tilde{P}_{\nu,\mu}(\delta_x^2) - i\frac{r^2}{12}\tilde{P}_{\nu,\mu}(\delta_x^2)^2\right]U^{n}.\end{aligned} \tag{4.4}$$

The Padé $(2,2;\nu,\mu)$ schemes can be written in factored form as follows.

$$\begin{aligned}\left[\tilde{Q}_{\nu,\mu}(\delta_x^2) + i\frac{r}{2}z_1\tilde{P}_{\nu,\mu}(\delta_x^2)\right]\tilde{U}^{n+1} &= \left[\tilde{Q}_{\nu,\mu}(\delta_x^2) - i\frac{r}{2}z_1\tilde{P}_{\nu,\mu}(\delta_x^2)\right]U^{n},\\ \left[\tilde{Q}_{\nu,\mu}(\delta_x^2) + i\frac{r}{2}z_1^*\tilde{P}_{\nu,\mu}(\delta_x^2)\right]U^{n+1} &= \left[\tilde{Q}_{\nu,\mu}(\delta_x^2) - i\frac{r}{2}z_1^*\tilde{P}_{\nu,\mu}(\delta_x^2)\right]\tilde{U}^{n+1},\end{aligned} \tag{4.5}$$

where $z_1 = 3 + i\sqrt{3}$. These schemes are $O(k^4, h^{2(\nu+\mu)})$ accurate for (4.1) and the semi-bandwidth of the matrices is $2\max\{\nu,\mu\}$. For $(\nu,\mu) = (1,1)$, the resulting split-step, implicit finite difference scheme will be referred to as SS-FD(4,4) (see Example 5 in Section 3.3).

4.2 A Split-step Explicit Finite Difference Scheme

In this section, we describe a split-step implementation of an explicit finite difference scheme for the Schrödinger equation. The basic idea is to stabilize the (unconditionally unstable) explicit Euler method by applying a small ($O(h^2)$) complex artificial viscosity.

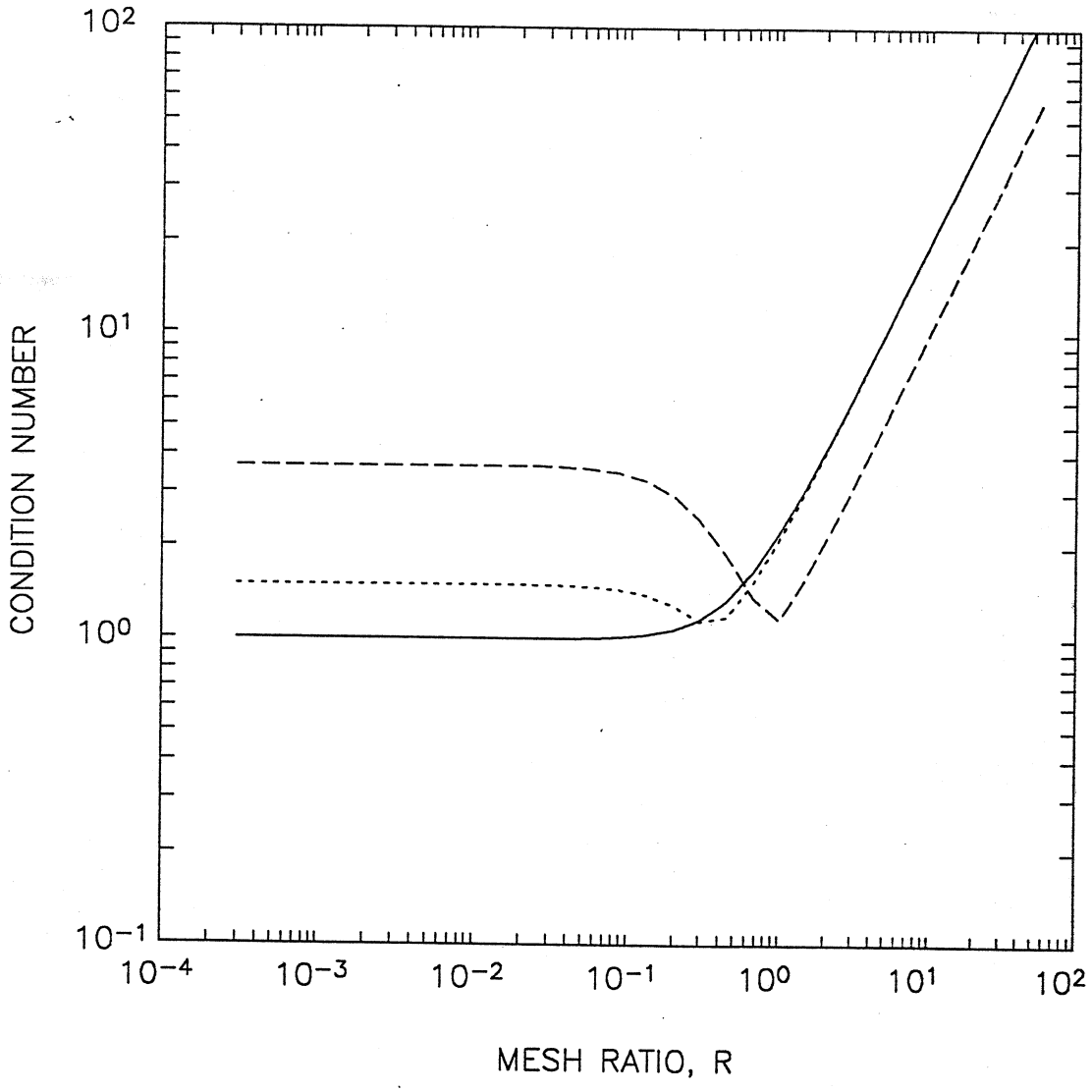


Figure 4.1: *The condition number of the matrix (4.3) of the Padé* $(1,1;\nu,\mu)$ *difference scheme vs. the mesh ratio* r. *The order of the matrices,* m, *is* 500. $(\nu,\mu) = (0,1)$ *(solid line),* $(1,1)$ *(dashed line) and* $(2,2)$ *(dotted line).*

In [41], a family of such schemes was introduced and the scheme in this family with the least restrictive stability condition was identified. We will use this scheme to approximate the operator $e^{-ikD_x^2}$ and refer to it as the 5P scheme. This scheme is

$$U^{n+1} = \left[I - ir\delta_x^2(1 + \frac{1}{4}(1-i))\delta_x^2\right] U^n. \tag{4.6}$$

With this choice of $C_A(k)$, the resulting split-step scheme of the form (2.23) will be referred to as the split-step five-point scheme (SS-5P). The accuracy of this scheme is $O(k, h^2)$ and the stability limit is $r = \frac{k}{h^2} \leq \frac{1}{2}$. The work at each time step is a (penta-diagonal) matrix-vector product.

An advantage of our split-step implementation of this explicit method over the original unsplit version is that the stability analysis can be done by Fourier techniques and is thus considerably simpler. Instead of a complicated application of the energy method to prove the stability of the scheme for the full Schrödinger equation [42], we can combine the stability result for the scheme for $iu_t = u_{xx}$ from [41] with Theorem 2.3.2 to get the following result:

Theorem 4.2.1 *The SS-5P scheme is contractive for* $r \leq \frac{1}{2}$.

On conventional (serial) computers, a first-order accurate method is generally not competitive with the higher-order methods that we will discuss below. On a parallel computer and in a situation where the accuracy requirements are modest, the SS-5P method may prove to be competitive. However, the use of this method is essentially ruled out on parallel computers for which the communication start-up cost is high relative to the arithmetic speed (e.g. the Intel iPSC/1).

4.3 Three-level, Implicit Finite Difference Schemes

We now propose a family of three-level schemes for the Schrödinger equation, all of which treat the x-derivative term implicitly and the Vu term explicitly. The motivation is to devise schemes that have the high accuracy of the Crandall-Douglas and the FD(2,8) schemes, without using the split-step approach. Unfortunately, these schemes are only conditionally stable as opposed to the unconditional stability of the split-step versions.

Let $\tilde{P}$ and $\tilde{Q}$ be polynomials of degree μ, ν, respectively, that define the (ν, μ) Padé approximation to D_x^2 of the form (3.9). Then the three-level scheme that we propose based on this Padé approximation is

$$\left[\tilde{Q}(\delta_x^2) + ir\tilde{P}(\delta_x^2)\right] U^{n+1} = \left[\tilde{Q}(\delta_x^2) - ir\tilde{P}(\delta_x^2)\right] U^{n-1} - 2ik\tilde{Q}(\delta_x^2)(VU^n). \tag{4.7}$$

As in Section 4.1, we are interested in the Padé (1,0), Padé (1,1) and Padé (2,2) methods, which we will refer to as 3L-CN (three-level Crank-Nicolson), 3L-CD (three-level Crandall-Douglas) and 3L-FD(2,8) (three-level FD(2,8)). Using the definition of order of accuracy for multi-level schemes proposed in [4], it is straightforward to show that the three-level schemes described above are second-order in time and have the same order of accuracy in space as the

underlying Padé approximant. Thus, the local truncation errors for 3L-CN, 3L-CD, 3L-FD(2,8) are $O(k^2, h^2)$, $O(k^2, h^4)$ and $O(k^2, h^8)$, respectively.

One difference between (4.2) and (4.7) is that the factor $r/2$ in (4.2) becomes r in (4.7). This reflects the fact that the "effective" time step for the three-level scheme is doubled since the implicit part involves the time levels $n-1$ and $n+1$. This results in a fourfold increase in the temporal component of the truncation error as compared to the corresponding split-step, finite difference schemes with the same time step k.

The characteristic polynomial of a difference scheme is defined in [46]. Assuming that V is a constant, the characteristic equation for a scheme of the form (4.7) is

$$(\tilde{Q}(\lambda) + ir\tilde{P}(\lambda))\xi^2 + 2ikV\tilde{Q}(\lambda)\xi - (\tilde{Q}(\lambda) - ir\tilde{P}(\lambda)) = 0, \tag{4.8}$$

for $\lambda \in [-4, 0]$. If all roots of the characteristic polynomial of a difference scheme of the form (4.7) have modulus one and are distinct, for all λ in the spectrum of the matrix that represents δ_x^2, the scheme is unitary. The roots of the characteristic polynomial (4.8) are

$$\xi = \frac{-ikV\tilde{Q}(\lambda) \pm \sqrt{\tilde{Q}(\lambda)^2(1 - k^2V^2) + r^2\tilde{P}(\lambda)^2}}{\tilde{Q}(\lambda) + ir\tilde{P}(\lambda)}$$

If $|kV| < 1$, then the square root is real and

$$\begin{aligned} |\xi|^2 &= \frac{k^2V^2\tilde{Q}(\lambda)^2 + \tilde{Q}(\lambda)^2(1 - k^2V^2) + r^2\tilde{P}(\lambda)^2}{\tilde{Q}(\lambda)^2 + r^2\tilde{P}(\lambda)^2} \\ &= 1. \end{aligned}$$

Since the expression under the square sign is non-zero, it follows that the two roots are distinct. This proves the following theorem.

Theorem 4.3.1 *If V is a constant and $|kV| < 1$, then the characteristic polynomial of the scheme (4.7) has distinct roots of modulus one. Hence, with this restriction on k, the scheme is unitary.*

Thus, the split-step approach can be dispensed with, at the cost of doubling the effective time step, and introducing a limit on the time step that depends on the potential V.

4.4 Higher Order in Time Schemes

In this section, we discuss three different approaches to constructing numerical schemes for the Schrödinger equation that have a higher order of accuracy (than two) in time. Richardson extrapolation and deferred corrections are two other devices for obtaining higher-order methods but we will not discuss them here.

We recall that the operator H was defined to be: $Hu = u_{xx} + Vu$.

4.4.1 Schemes based on a Taylor series approximation

A simple way to approximate the operator exponential $exp(-ikH)$ is by the truncated Taylor series. A typical member of this family is the third order scheme

$$C_A(k) = I - ikH - \frac{1}{2}k^2H^2 + \frac{i}{6}k^3H^3.$$

In the Padé terminology, these schemes are are Padé $(0, n)$ in time. In the context of ordinary differential equations, Henrici [45] distinguishes between a direct and an indirect (Runge-Kutta) implementation of these methods.

If we divide the stability limit by the order of the method (which is equal to the number of applications of the operator H (1.1)) we have a criterion for comparing efficiency. For the conditionally stable third- and fourth-order schemes, this works out to $\frac{\sqrt{3}}{3}$ and $\frac{\sqrt{8}}{4}$, respectively, implying that the fourth-order Taylor series method is more efficient than the third-order method. From Theorem 3.1.3 we know that the Taylor series methods of orders 1, 2, 5 and 6 are unconditionally unstable for the Schrödinger equation. Thus, if we were choosing a Taylor series method for the Schrödinger equation of order six or less, the fourth-order method would be best.

Taylor series methods lose efficiency due to the necessity of applying the the operator H several times at each time step. As pointed out in [45], these methods can be useful in computing the additional initial values required by multi-step methods which we will discuss below.

A spectral implementation involves applying the differential operator in H in Fourier space. For the numerical results to be presented in the next chapter, a spectral fourth-order Taylor series method was used to compute the additional initial values for some spectral multi-level methods.

4.4.2 Multi-level schemes

Linear multistep methods are used extensively in the numerical solution of ordinary differential equations [45], [44]. For the Schrödinger equation, these methods have the form

$$U^{n+1} = \sum_{r=1}^{\ell} \alpha_r U^{n+1-r} + kH \sum_{s=0}^{\ell} \beta_s U^{n+1-s}. \tag{4.9}$$

The leap-frog method is a widely used method of this form and will be discussed separately.

The Adams-Bashforth family of schemes has the property that $\alpha_1 = 1$, $\alpha_r = 0$, for $r > 1$ and $\beta_0 = 0$ (see [45]).

A spectral implementation of the Adams-Bashforth methods is mentioned in [9] for the heat equation, $u_t = u_{xx}$. We will refer to the spectral Adams-Bashforth methods as SAB(ℓ). The stability limits have the form $k\lambda < c_\ell$, where λ is the largest eigenvalue of the of the matrix that approximates the Hamiltonian. This can be written as $k||H||_2 < c_\ell$. The constants c_ℓ for the Schrödinger equation are different from those for the heat equation. The stability limits for the Adams-Bashforth methods are quite severe and limit the efficiency of the SAB(ℓ) family. We

will present numerical results for the SAB(ℓ) methods with $\ell = 2$, 3, 4 and 5. These methods may be of interest when the accuracy requirements are high.

The Adams-Moulton schemes have a number of advantages over the Adams-Bashforth schemes but their implicit nature makes them less suitable for a spectral implementation.

In this context, it should be mentioned that a result of Jeltsch [32] states that a linear multistep method whose stability region includes the entire imaginary axis is A-stable. A classical result of Dahlquist [43] implies that such a method must be implicit and cannot have order greater than two. In particular, for the Schrödinger equation, the order of an unconditionally contractive linear multistep method cannot exceed two.

4.5 Other schemes

For completeness, we give a brief review of several other numerical methods for the Schrödinger equation. Some of these methods will be included in the comparison of methods that will be presented in the next chapter.

4.5.1 Leap-frog schemes

The simplest variants of the leap-frog scheme for the Schrödinger equation is the LF scheme of [38]:

$$U^{n+1} = U^{n-1} - 2ir\delta_x^2 U^n - 2ikVU^n. \tag{4.10}$$

It is $O(k^2, h^2)$ accurate, and conditionally stable[2]. The stability analysis (for constant V) reduces to examining the roots of the characteristic polynomial

$$m^2 + 2i\alpha m - 1, \tag{4.11}$$

where $\alpha = k(\lambda + V)$ and $\lambda \in (-4/h^2, 0)$. In particular, α is real. The roots are given by

$$m_\pm = -i\alpha \pm \sqrt{1-\alpha^2}.$$

If $|\alpha| < 1$, the roots are simple and have modulus one. Thus, a sufficient condition for the leap-frog scheme to be unitary is $k||H|| < 1$.

In the Schrödinger equation the time derivative of the real part of the solution depends only on the imaginary part and vice versa. Thus, following [51], we can compute only the real or the imaginary part of the solution at each time level. If we write

$$U^n = X^n + iY^n,$$

where X^n and Y^n are real m-vectors, then this scheme, LF(R/I), can be written as

$$\begin{aligned} X^{n+1} &= X^{n-1} - 2ir\delta_x^2 Y^n - 2ikVY^n \\ Y^{n+2} &= Y^n - 2ir\delta_x^2 X^{n+1} - 2ikVX^{n+1}. \end{aligned} \tag{4.12}$$

[2]The corresponding scheme for the diffusion equation is unconditionally unstable.

The memory usage and computational requirements are about half those of the standard leap-frog implementation, while the accuracy and stability properties are unchanged.

Another possibility is to use Fourier transforms to evaluate the derivative term at the intermediate time level ([47]). This SLF or spectral leap-frog scheme, can be written as

$$U^{n+1} = U^{n-1} - 2ik\left[\mathcal{F}^{-1}\mathcal{D}\mathcal{F}U^n + VU^n\right], \tag{4.13}$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are defined in (2.26) and $\mathcal{D}$ is the diagonal matrix with entries

$$\mathcal{D}_{s,s} = -s^2\theta^2. \tag{4.14}$$

Although for band limited solutions U^n, the spatial error is almost negligible, the temporal error is still $O(k^2)$, as for the other leap-frog schemes. More importantly, the conditional stability requires that smaller time steps be taken to improve the spatial resolution, even though this may not be necessary for the overall accuracy. This is likely to limit the efficiency of the leap-frog schemes (and of any other conditionally stable scheme).

A combination of the LF(R/I) scheme with the spectral in space treatment of SLF would be more efficient than either one on its own. This is denoted by SLF(R/I).

4.5.2 An explicit scheme

In this section, we briefly state the explicit scheme of [41] for the Schrödinger equation. We have used this scheme in a split-step framework (4.6). Here we merely state the scheme in its original form where the treatment of the potential is incorporated into the basic scheme. The scheme for the full Schrödinger equation can be written as

$$U^{n+1} = \left[I - ir\delta_x^2(1 + \frac{1}{4}(1-i))\delta_x^2 - ikV\right]U^n. \tag{4.15}$$

This scheme is the one with the least restrictive stability condition of all schemes in a family of explicit methods introduced in [41].

4.5.3 The Chan-Kerkhoven scheme

Chan and Kerkhoven in [40] proposed a scheme for the Korteweg-de Vries equation and suggested that their analysis could be extended with minor modifications to the Schrödinger equation. The idea is to use a three-level scheme, march the solution out in Fourier space, and use the FFT at each time step to move to physical space to evaluate the potential and then transform back. This scheme for the Schrödinger equation can be written as

$$[I + ik\mathcal{D}]\,\tilde{U}^{n+1} = [I - ik\mathcal{D}]\,\tilde{U}^{n-1} - 2ik\mathcal{F}V\mathcal{F}^{-1}\tilde{U}^n, \tag{4.16}$$

where $\tilde{U}_j^n$ is the j-th coefficient of the sine series at time level n (see (2.24)) and $\mathcal{F}$ and $\mathcal{D}$ are defined by (2.26) and (4.14), respectively.

The cost of each time step is dominated by the cost of the two FFTs. The fact that the scheme is implicit turns out not affect the cost appreciably, since the matrix is diagonal and independent of time, and solving the system reduces to m complex multiplications.

4.5.4 Ordinary differential equation approaches

Perturbation methods

If the potential V can be written as the sum of two parts, $V = V_0 + V_1$, with V_0 time-independent, and if the Hermitian operator H_0 defined by $H_0 u = u_{xx} + V_0 u$ has eigenfunctions ϕ_n with $H_0\phi_n = E_n\phi_n$, then we can write

$$u(x,t) = \sum_{r=1}^{\infty} c_r(t)\phi_r(x). \tag{4.17}$$

Substituting into the Schrödinger equation, we get an infinite system of ordinary differential equations. Viewing $\mathbf{c}(t) = (c_1(t), c_2(t), \ldots)^T$ as an infinite vector, this system of ordinary differential equations can be written as

$$\dot{\mathbf{c}}(t) = E\mathbf{c} + C\mathbf{c},$$

where E is a diagonal matrix, with the eigenvalues of H_0 on the diagonal, and C is matrix whose entries are

$$C_{r,s} = \int \bar{\phi}_r(x) V_1 \phi_s(x) dx.$$

The effectiveness of this technique depends on how "close" V_0 is to V, which is reflected in the structure of C. In practice, one would use a finite number, N, of eigenfunctions, and solve the system of equations using an ODE package.

Method of lines

A different approach that also reduces the solution of the Schrödinger equation to the integration of a system of ordinary differential equations is the method of lines. In this method, one defines a spatial grid and uses some finite difference approximation to the derivatives. After this semi-discretization, the vector of approximate solutions at the grid points is viewed as a function of time and the partial differential equation is transformed into a system of ordinary differential equations. This approach has been applied to the parabolic wave equation in [48] and [49].

4.5.5 Overview of the methods

In Figure 4.2, we use a tree diagram to depict the connections between a number of methods for solving the Schrödinger equation in one space dimension. In the next chapter, we will present numerical results for a number of these methods.

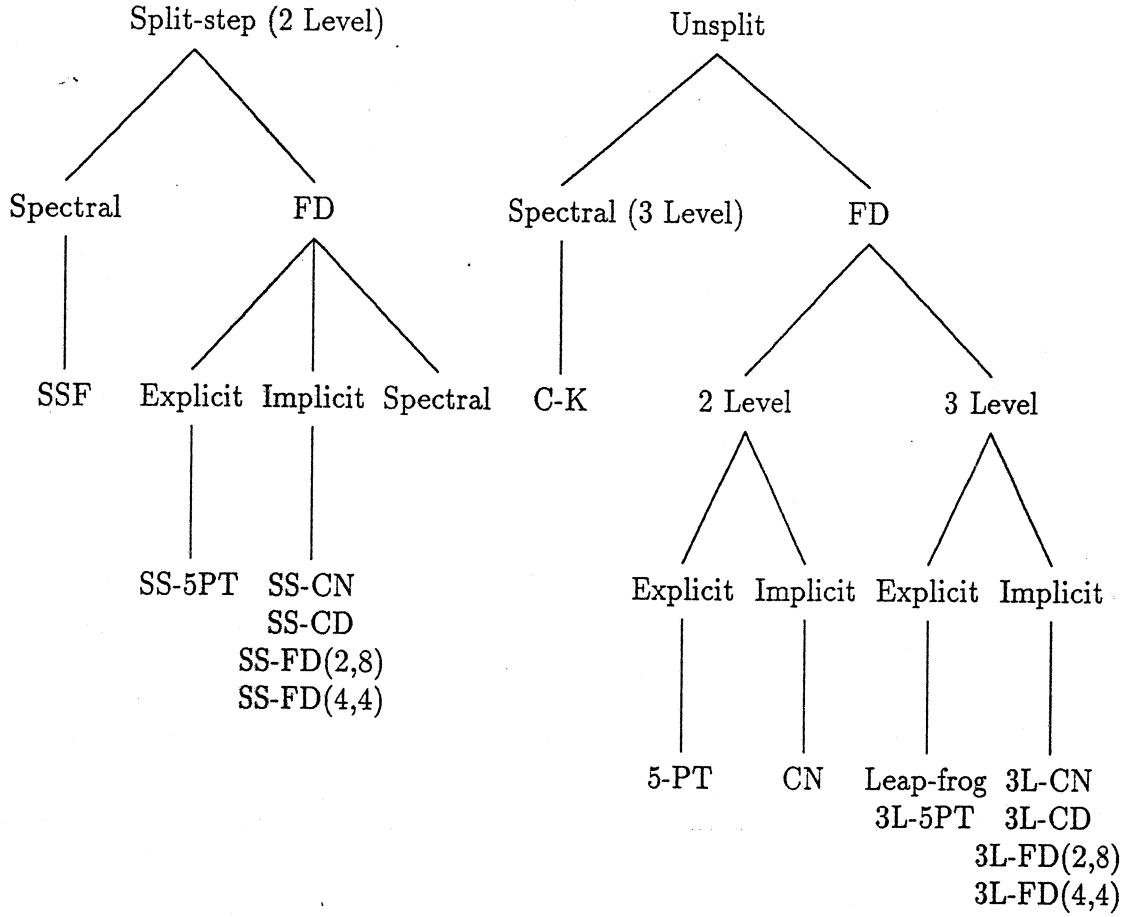


Figure 4.2: *Methods for the Schrödinger equation in one space dimension. (FD = finite difference).*

Chapter 5

Numerical Results for the One-dimensional Schrödinger Equation

In this chapter, we present the results of some experiments with the methods proposed in earlier chapters and compare these methods with some of the other methods we have discussed.

5.1 The Test Problem

We recall the form of the one-dimensional Schrödinger equation:

$$i\frac{\partial u}{\partial t} = Hu \equiv \frac{\partial^2 u}{\partial x^2} + V(x,t)u, \qquad x \in (a,b), \quad t > 0, \tag{5.1}$$

$$u(a,t) = u(b,t) = 0, \quad t > 0,$$
$$u(x,0) = \phi(x),$$

The test problem was adapted from the one-dimensional linear harmonic oscillator of quantum mechanics (see, for example, [57]). For (5.1), this means choosing $V(x) = -x^2$. For this potential, the solution is

$$u(x,t) = e^{-\frac{x^2}{2}} \sum_{n=0}^{\infty} c_n e^{-iE_n t} H_n(x), \qquad E_n = -(2n+1), \tag{5.2}$$

where $H_n(x)$ denotes the n-th Hermite polynomial. The largest n for which $c_n \neq 0$ will be referred to as the *mode number*. Thus, higher mode numbers[1] correspond to more oscillatory solutions. The computational domain was chosen to be the interval $[-10, 10]$ with zero boundary conditions. This is justified for the modes we deal with, because the true solution goes to zero rapidly as $|x|$

[1] The mode numbers n in (5.2) should not be confused with the number of (Fourier) modes, m, used in spectral methods to construct the approximate solution.

increases. This casts the problem into the form (5.1). For the splitting we will use in our tests, SP_2, the expression for the splitting error given in (2.19) simplifies to

$$i\frac{k^2}{3}\left[x^2u(x) + 2u_{xx}(x)\right] \tag{5.3}$$

for the potential in our test problem.

The computations were carried out in double precision (53 bit mantissa) arithmetic. All computations were performed using FORTRAN complex arithmetic, except for the the FFTs, which were performed in real arithmetic, using FFTPACK [5].

In discussing the accuracy of the various methods, we will use the quantity

$$\text{DIGITS} = \max\left\{-\log_{10}\left(\frac{||U_c - U_t||_2}{||U_t||_2}\right), 0\right\}, \tag{5.4}$$

which is a measure of the (average) number of correct digits in the computed solution, U_c. The true solution on the grid is denoted by U_t and $||.||_2$ is the discrete 2-norm.

For ease of reference, we have listed the methods that were included in the numerical tests in Table 5.1. The table also summarizes some properties of the methods.

5.2 Selection of the Mesh Parameters

The computational efficiency of a method depends, among other things, on the choice of the time and space mesh sizes. In this section, we discuss an approach to selecting the mesh parameters in an optimal manner, based on solving a minimization problem. This is important for a meaningful comparison of different methods.

Let $W(k,h)$ be the cost of computing an approximate solution at time T by a given method using a time step k and grid spacing h. Let $E(k,h)$ be the norm of the error in this approximate solution. For a given error tolerence ϵ, the optimal (k,h) pair is the solution of the following constrained minimization problem.

$$\text{(MP):}\quad \text{Minimize } W(k,h), \text{ subject to } E(k,h) \le \epsilon \text{ and } (k,h) \in \mathcal{S},$$

where $\mathcal{S}$ is the set of non-negative pairs (k,h) that satisfy the stability conditions (if any) associated with the method. We have included the stability limit in (MP) only to stress the fact that, for conditionally stable methods, we are constrained to remain in $\mathcal{S}$. In fact, the condition $E(k,h) \le \epsilon$ will ensure that $(k,h) \in \mathcal{S}$. (A violation of the stability limit leads to unbounded growth in the computed solution.) This approach of selecting the mesh parameters based on the solution of an optimization problem was used in [54]. Related ideas are found in [55], [58].

The optimization problem (MP) is solvable for an important special case, namely when $W(k,h) = C(kh^d)^{-1}$, where d is the number of space dimensions, and $Ak^\alpha + Bh^\beta$ is a good approximation to $E(k,h)$. Here, A, B and C are generic constants.

Table 5.1:

Methods used in the numerical tests. In the Stability Limit column, V refers to the operator that corresponds to multiplication by the potential function and H refers to the operator $D_x^2 + V$. The operation counts are for one-time step and m is the number of grid points (or modes for the spectral methods).

Method	Accuracy	Stability Limit	Operation Count
SS-Crank-Nicolson (SS-CN) (4.2), [53]	$O(k^2, h^2)$	none	$38m$
SS-Crandall-Douglas (SS-CD) (4.2)	$O(k^2, h^4)$	none	$38m$
SS-FD(2,8) (4.2)	$O(k^2, h^8)$	none	$80m$
SS-FD(4,4) (4.4)	$O(k^2, h^4)$	none	$80m$
SS-5P (4.6)	$O(k, h^2)$	$r \leq \frac{1}{2}$	$32m$
SS-Fourier (SSF) [10]	$O(k^2)$	none	$(10log_2 m + 12)m$
Chan-Kerkhoven (C-K) (4.16), [40]	$O(k^2)$	$k\|\|V\|\| < 1$	$(10log_2 m + 20)m$
Spectral leap-frog (SLF) (4.13), [47]	$O(k^2)$	$k\|\|H\|\| < 1$	$(10log_2 m + 16)m$
Spectral Adams-Bashforth			
SAB3	$O(k^3)$	$k\|\|H\|\| < c_3$	$(10log_2 m + 26)m$
SAB4	$O(k^4)$	$k\|\|H\|\| < c_4$	$(10log_2 m + 30)m$

This case includes many of the finite difference schemes we have discussed, for $d = 1$, and the ADI schemes for higher dimensions. The spectral methods do not fall into this group.

Even without knowing the error constants A and B, we can deduce that, with the optimal k and h,

$$W^* = O(\epsilon^{-\sigma}), \quad \text{as} \quad \epsilon \to 0, \quad \text{where} \quad \sigma = \frac{1}{\alpha} + \frac{d}{\beta}. \tag{5.5}$$

If $\alpha = 2$ and $d = 1$, then, $W^* = \mathrm{O}(\epsilon^{-\tilde{\sigma}})$ with $\tilde{\sigma} = \tilde{\sigma}(\beta) = \frac{1}{2} + \frac{1}{\beta}$, for all $\beta > 0$. Thus all finite difference methods that are second order accurate in time will have a value of $\sigma > \frac{1}{2}$, no matter how accurate they are in space. In particular, for SS-CN, SS-CD and SS-FD(2,8), the values of σ are 1, $\frac{3}{4}$ and $\frac{5}{8}$, respectively.

The ratio of the temporal to spatial error (Ak^α:Bh^β) is β:αd. This expresses the intuitively obvious fact that the temporal and spatial errors must be balanced at the optimal choice of k and h. The correct balance is seen to depend on the orders of accuracy and the dimensionality. For example, for a scheme with accuracy $\mathrm{O}(k^2, h^8)$, the optimal k and h result in a temporal error that is four times larger than the spatial error (with $d = 1$).

The ratio (k/h^γ) remains constant as the accuracy requirement is increased, for $\gamma = \frac{\beta}{\alpha}$. For Crank-Nicolson, $\gamma = 1$, which means that k/h should be kept constant for maximum efficiency[2]. The special mesh ratio, $r = k/h^2$, decreases for SS-CN, remains constant for SS-CD and increases for SS-FD(2,8) as the accuracy requirement is increased, with the optimal choice of k and h.

The minimization problem (MP) does not have such a simple solution for the spectral methods since the cost is $W(k,h) = C(kh^d)^{-1}\log(\frac{1}{h})$.

5.3 Numerical Results

In Figures 5.1 through 5.4, we have plotted (the logarithm of) the cost of marching from $t = 0$ to $t = 1$ as a function of number of correct digits in the final solution. This essentially corresponds to a plot of W^* versus ϵ^{-1} with logarithmic scaling. Thus, a smaller slope means greater efficiency. For completeness, we have tabulated the data that was used in the plots in Tables 5.2 through 5.5.

For all the finite difference methods except SS-5P, we first estimated the error constants A and B by solving the problem with a number of (k,h) pairs. The minimization problem (MP) was then solved for a number of values of the accuracy parameter ϵ. The results of the computations with these (near-) optimal parameters are displayed. For SS-5P, k and h were chosen to satisfy the stability limit. For the Split-step Fourier method, $m = 63$ was used for all the data displayed in Figure 5.1. (Using more Fourier modes did not make the solution more accurate in our examples.)

[2]Isaacson and Keller [56] reach the same conclusion for the Crank-Nicolson method applied to the heat equation by examining an upper bound for the global error.

In Figure 5.1, the solution consisted of the lowest mode only (n=0, in (5.2)). Figure 5.2 is the same as Figure 5.1, except that the solution consisted of the modes corresponding to $n = 3$ and 4 in (5.2). The results in Figures 5.1 and 5.2 demonstrate the effect of increasing the mode number (energy eigenvalue) of the solution. The splitting error grows approximately like $k^2|E_n|$ (due to the u_{xx} term in (5.3)), whereas the Crank-Nicolson time-differencing (Padé $(1,1)$ in time) error grows like $k^2|E_n|^3$ (see (2.3))[3]. For higher-order time-differencing schemes the temporal error behaves like $k^\alpha|E_n|^{\alpha+1}$. For example, the temporal error of a scheme based on Padé $(2,2)$ in time behaves like $k^4|E_n|^5$.

In both figures, the SS-5P, SS-CN, SS-CD and SS-FD(2,8) schemes have the same relative performance and the slopes agree well with the theoretical values of σ. The advantage of higher-order finite difference schemes over SS-5P and SS-CN is seen to increase with the accuracy requirement and mode number. For example, SS-CD is about $6\frac{1}{2}$ times more efficient than SS-CN for 3 digits of accuracy with the lowest mode ($n = 0$ in (5.2)). The factor increases to 10 for 4 digits of accuracy with the same mode number. This factor is also 10 for 3 digits of accuracy with higher modes ($n = 3, 4$). The improvement of the SS-Crandall-Douglas scheme over SS-Crank-Nicolson does not require additional smoothness of the solution. The other higher-order finite difference schemes do require the existence of higher derivatives.

The relative ranking of these two methods is reversed as we go from Figure 5.1 to Figure 5.2. SS-CD and SS-FD(4,4) are both $O(k^2, h^4)$ accurate, but the local truncation error of SS-FD(4,4) does not have an $O(k^2)$ Crank-Nicolson time-differencing error. This is because in the first case, the Crank-Nicolson error is comparable to the splitting error and the extra work done by SS-FD(4,4) to eliminate the Crank-Nicolson error results in a loss of efficiency. However, with the higher mode numbers, the Crank-Nicolson error is larger (by a factor of roughly 100) than the splitting error and SS-FD(4,4) is superior to SS-CD.

For the Split-step Fourier method, the overall error is essentially due to the splitting error. In Figure 5.1, we see that SS-FD(2,8) is more efficient than SSF up to an accuracy requirement of over 6 digits. In the case of higher modes, SS-FD(2,8) is less efficient than SSF due to the large Crank-Nicolson error. However, for low accuracy requirements, SS-FD(4,4) is more efficient than SSF.

For other potentials, we would expect to see the same qualitative behavior, with the higher-order finite difference methods being computationally more efficient for lower modes and moderate accuracy requirements and the SSF method becoming more efficient for higher modes. However, the cross-over point (in terms of mode number and/or accuracy requirements) depends on the splitting error which in turn depends on the potential. For potentials that have steep gradients, the splitting error will be larger and the cross-over point will be shifted in favor of the higher-order finite difference schemes.

In Figures 5.3 and 5.4 we compare the performance of several spectral methods, including the SSF method. The results for SS-CN and the best finite difference method are also included.

[3]This behavior has been pointed out in [12] in the context of the parabolic wave equation.

The spectral Adams-Bashforth, C-K and spectral leap-frog methods are conditionally stable; for these methods the left end of the curve corresponds very nearly to the smallest step size allowed by the stability limit. (Any reduction in the time step, k, results in an unbounded growth in the computed solution.) The number of modes, m, for all spectral methods was fixed at 63.

The efficiencies of the C-K and SLF schemes are very close. The C-K scheme has the advantage of an extended interval of stability (see the Stability Limit column in Table 5.1). The consequence is that the left end-point of the C-K curves lies further left than that of SLF). This difference grows with the number of modes m.

The SAB schemes have more restrictive stability limits but do offer a simple mechanism for computing accurate (7 digits or more) solutions at a lower cost than the second-order schemes.

The stability limits of SAB2 and SAB3 are very close, which implies that SAB3 is always preferable to SAB2, provided the additional storage required by SAB3 is available.

However, Figures 5.3 and 5.4 indicate that the SAB family is inferior to the other spectral schemes for modest accuracy requirements.

The combined effect of these comparisons is the following. The high-order finite difference schemes are the most efficient methods for solving the Schrödinger equation with low to modest accuracy requirements. There is an intermediate accuracy range for which SSF is most efficient. Finally, the higher-order spectral Adams-Bashforth methods are more efficient than SSF for high accuracy requirements. The SSF method is least affected by an increase in the mode number and the domain of superiority of SSF will expand as the mode number is increased.

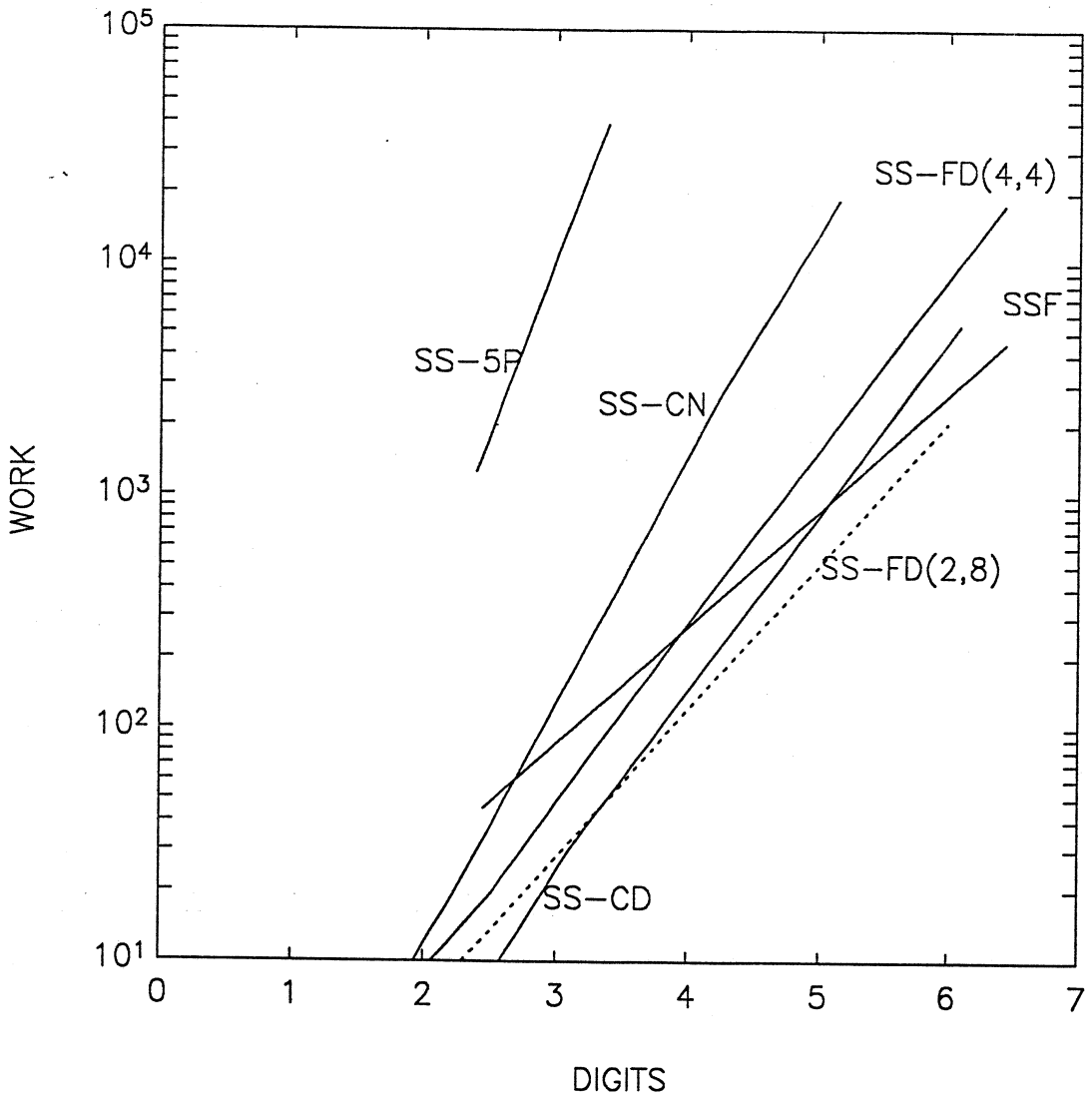


Figure 5.1: *Work vs. Accuracy. One unit of WORK is 1000 real floating point operations and DIGITS measures the number of correct digits in the computed solution at t=1. The solution contained only the lowest mode of the linear harmonic oscillator ($n = 0$ in (5.2)). The SS-5P data is based on taking the largest step allowed by the stability limit. The mesh sizes for the other finite difference methods were chosen by approximately solving the minimization problem (MP). For the split-step Fourier method, m was chosen to be 63 for all the data points.*

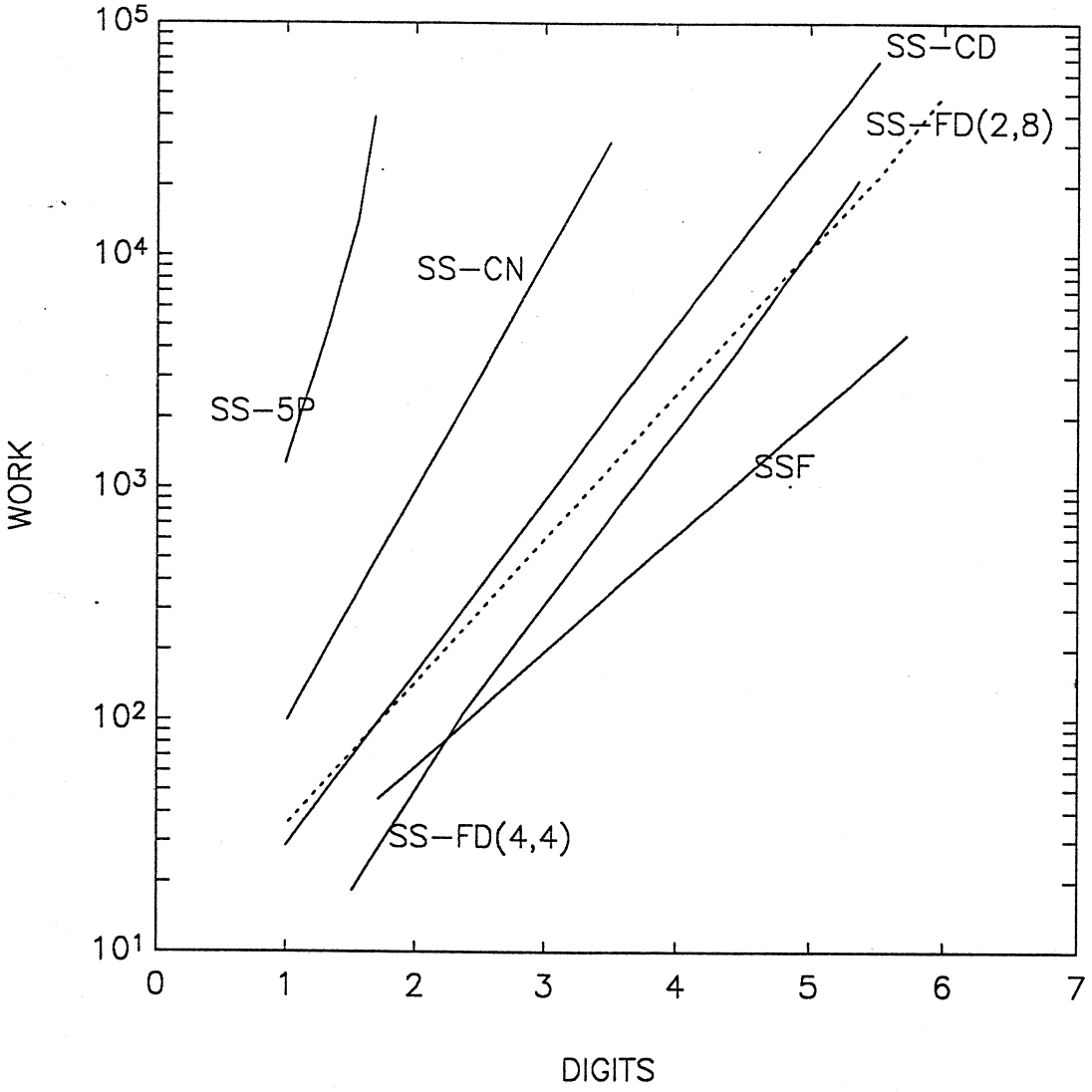


Figure 5.2: *Work vs. Accuracy. Same as Figure 5.1, except that the solution contained only the third and fourth modes. ($n = 3$ and 4 in (5.2)).*

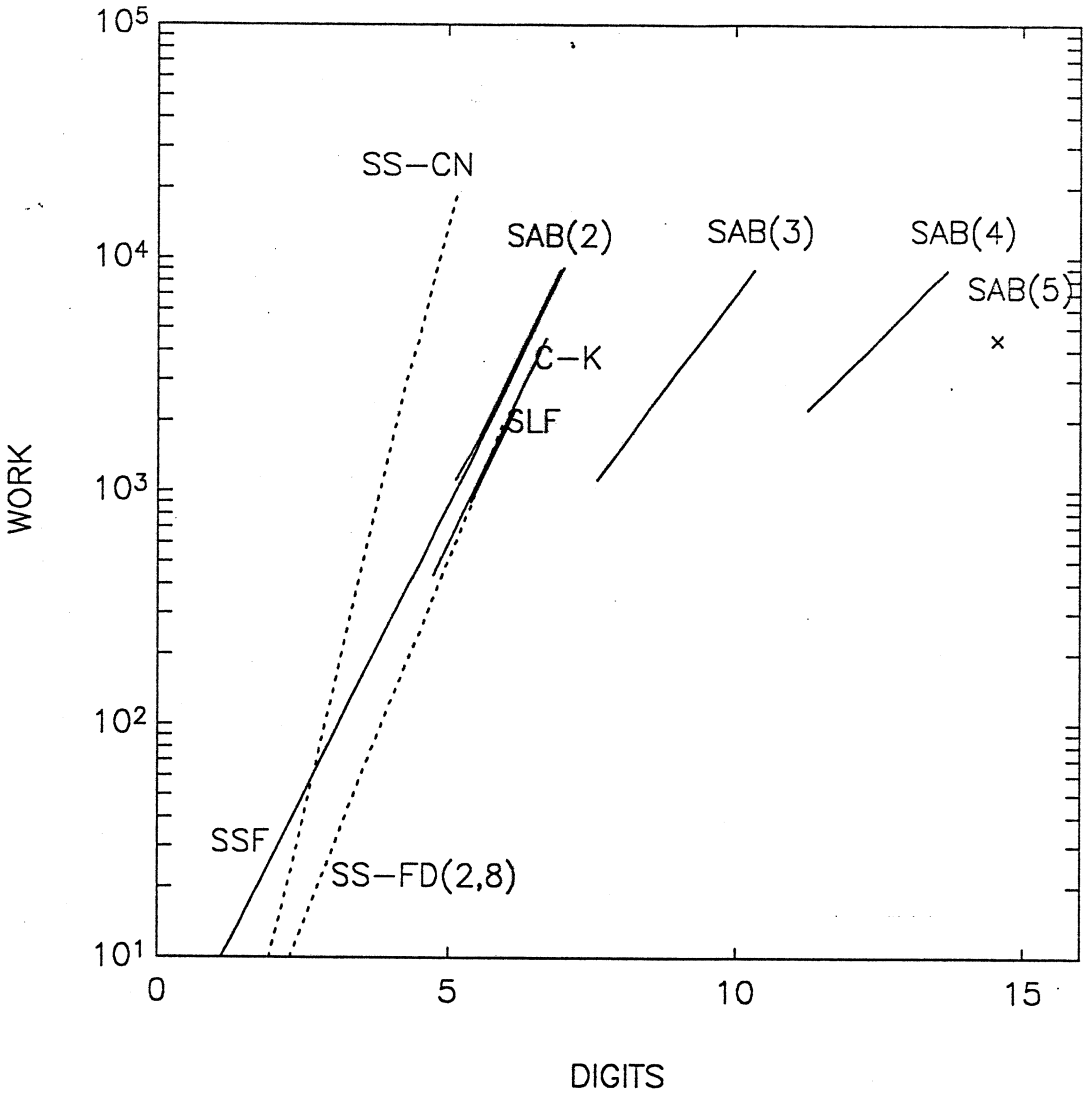


Figure 5.3: *Work vs. Accuracy. Spectral methods. Lowest mode only. For the spectral methods, m was chosen to be 63 for all the data points.*

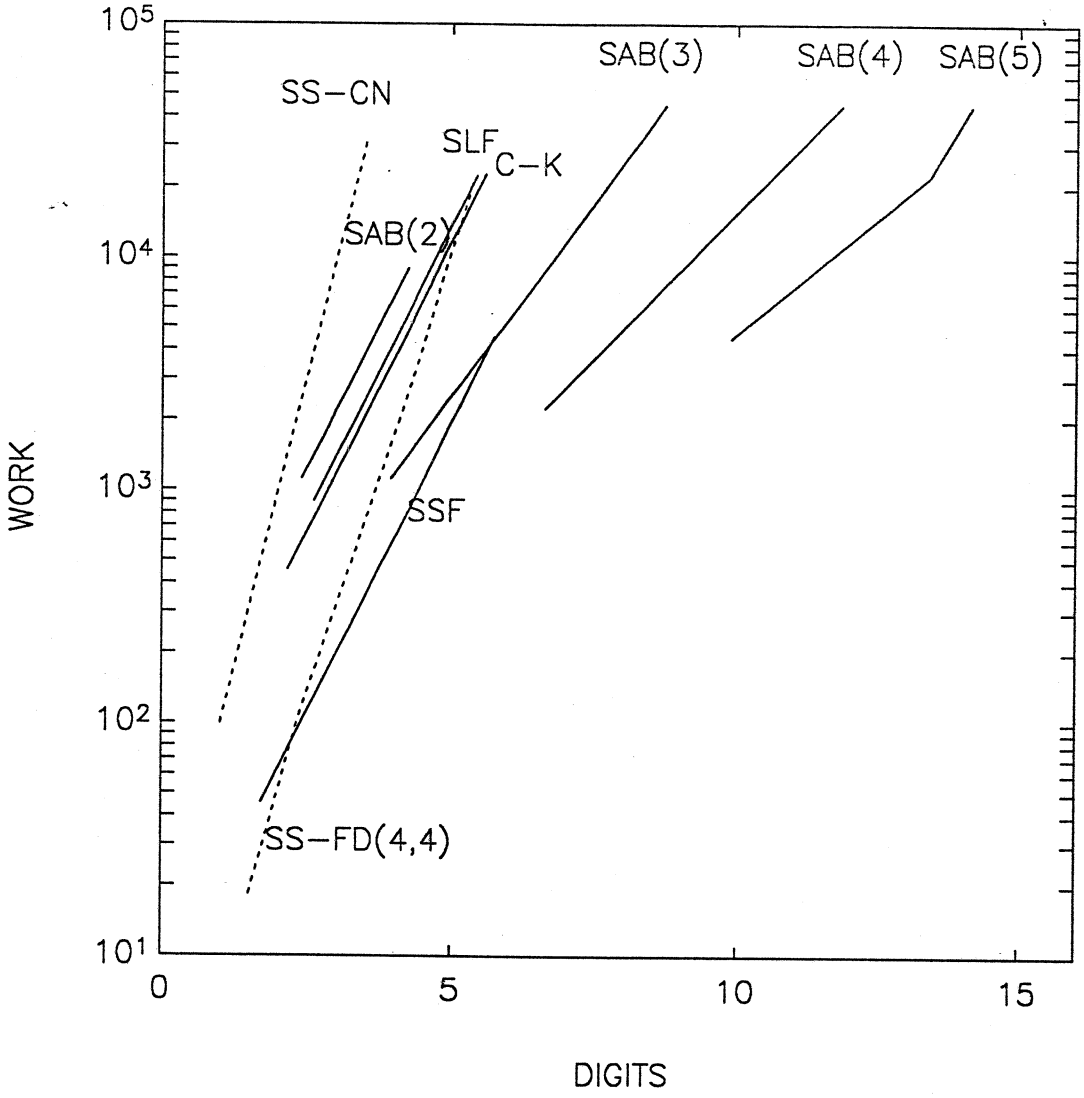


Figure 5.4: *Work vs. Accuracy. Same as Figure 5.3, except that the solution contained only the third and fourth modes. ($n = 3$ and 4 in (5.2)).*

Table 5.2:

Data for Figure 5.1, lowest mode of the linear harmonic oscillator.

Method	m	N	WORK	DIGITS
SS-5P	198	200	1,267	2.38
	613	2003	39,290	3.38
	998	5000	159,680	3.78
SS-CN	90	6	20.5	2.24
	298	18	203.8	3.19
	1325	55	2,769	4.26
	2855	172	18,660	5.14
SS-CD	31	5	5.9	2.32
	54	15	30.8	3.11
	97	48	176.9	4.11
	172	150	980	5.10
SS-FD(2,8)	19	5	7.6	2.09
	24	13	25	2.92
	33	44	116	3.97
	44	136	478	4.96
	60	430	2,064	5.99
SS-FD(4,4)	31	8	19.8	2.51
	54	23	99.4	3.40
	97	74	574	4.41
	172	233	3,206	5.41
SSF	63	10	45.4	2.44
	63	100	453	4.44
	63	500	2,268	5.84

Table 5.3:

Data for Figure 5.2, third and fourth modes of the linear harmonic oscillator.

Method	m	N	WORK	DIGITS
SS-5P	198	200	1,267	0.99
	315	500	5,040	1.34
	455	1000	14,240	1.56
SS-CN	138	19	99	1.01
	435	59	975	2.00
	1,376	186	9,725	3.00
SS-CD	84	51	162	2.02
	149	161	911	3.02
	264	511	5,126	4.01
	470	1,615	28,843	5.01
SS-FD(2,8)	45	83	611	2.51
	61	262	1,278	3.53
	81	829	5,371	4.53
	108	2,621	22,645	5.53
SS-FD(4,4)	83	61	106	2.37
	148	53	627	3.40
	263	169	3,555	4.41
	496	534	21,189	5.53
SSF	63	10	45	1.72
	63	100	453	3.72
	63	1000	4536	5.72

Table 5.4:
Data for Figure 5.3, lowest mode of the linear harmonic oscillator.

Method	N	WORK	DIGITS
C-K	100	504	4.74
	200	1,008	5.34
	500	2,520	6.14
	1,000	5,040	6.74
SLF	200	957	5.38
	500	2,394	6.18
	1,000	4,788	6.78
	2,000	9,576	7.38
SAB(2)	200	1,033	0.00
	250	1,291	5.12
	500	2,583	5.78
	1,000	5,166	6.38
	10,000	51,660	6.98
SAB(3)	200	1,083	1.66
	250	1,354	7.62
	500	2,709	8.52
	1,000	5,418	9.43
SAB(4)	500	2,835	11.26
	1,000	5,670	12.62
	2,000	11,340	13.52
SAB(5)	1,000	5,922	14.54
	5,000	29,610	14.46

Table 5.5:
Data for Figure 5.4, third and fourth modes of the linear harmonic oscillator.

Method	N	WORK	DIGITS
C-K	80	403	0.00
	100	504	2.18
	200	1,008	2.78
	1,000	5,040	4.18
SLF	200	957	2.62
	500	2,394	3.42
	1,000	4,788	4.02
	2,000	9,576	4.62
SAB(2)	200	1,033	0.00
	250	1,291	2.42
	500	2,583	3.02
	1,000	5,166	3.62
	10,000	51,660	4.23
SAB(3)	200	1,083	1.12
	250	1,354	3.93
	500	2,709	4.83
	1,000	5,418	5.73
SAB(4)	200	1,134	0.00
	500	2,835	6.62
	1,000	5,670	7.82
	2,000	11,340	9.02
SAB(5)	500	2,961	0.00
	1,000	5,922	9.90
	5,000	29,610	13.38

Chapter 6

Methods for the Schrödinger Equation in Two Space Dimensions

In this chapter, we present two-dimensional analogues of some of the methods introduced earlier for the numerical solution of the Schrödinger equation. The two-dimensional form of the Schrödinger equation (SE-2d) is

$$i\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + V(x,y,t)u, \qquad x \in R = (a,b)\times(c,d), \quad t > 0, \tag{6.1}$$

$$u(x,y,t) = 0, \quad \text{for} \quad (x,y) \in \partial R, \quad t > 0,$$

$$u(x,y,0) = \phi(x,y), \quad \text{for} \quad (x,y) \in R.$$

We will omit a discussion of methods for which the extension to two space dimensions is straightforward, such as the split-step explicit scheme of Section 4.2, the leap-frog schemes and the spectral Adams-Bashforth methods.

We will assume for simplicity that the number of points and the mesh width are the same in the x and y directions. Thus, the grid points (x_j, y_l) are given by $x_j = a + jh$ and $y_l = c + lh$, where $h = (b-a)/(n+1) = (d-c)/(n+1)$ and $j, l = 1, \ldots, m$. Unless otherwise stated, the grid points will be numbered in the "natural" order (by rows).

Let $U^n_{j,l}$ be the approximate solution of (SE-2d) at (x_j, y_l, t_n). With the subscripts omitted, U^n denotes the complex m^2-vector with components $U^n_{j,l}$, with the natural ordering. We will use the following finite difference operators:

$$\delta_x^2 U^n_{j,l} \equiv U^n_{j-1,l} - 2U^n_{j,l} + U^n_{j+1,l}, \qquad \delta_y^2 U^n_{j,l} \equiv U^n_{j,l-1} - 2U^n_{j,l} + U^n_{j,l+1}.$$

Differentiation with respect to x and y is denoted by $D_x^2 \equiv \frac{\partial^2}{\partial x^2}$ and $D_y^2 \equiv \frac{\partial^2}{\partial y^2}$.

Analogous to the one-dimensional case, $V^n_{j,l}$ denotes the value of the potential function V at (x_j, y_l, t_n). The $m^2 \times m^2$ diagonal matrix with diagonal entries $V^n_{j,l}$ will be denoted by V^n, with the natural ordering.

6.1 The Crank-Nicolson Method

The Crank-Nicolson scheme for the two-dimensional Schrödinger equation (SE-2d) has the form

$$\left[I + \frac{ir}{2}\left(\delta_x^2 + \delta_y^2\right) + \frac{ik}{2}V^{n+1}\right] U^{n+1} = \left[I - \frac{ir}{2}\left(\delta_x^2 + \delta_y^2\right) - \frac{ik}{2}V^{n}\right] U^{n}. \tag{6.2}$$

Although the structure of (6.2) is a simple generalization of the one-dimensional Crank-Nicolson scheme(2.2), the matrix on the left-hand side now has the non-zero structure of the discrete Laplacian, with non-constant main diagonal. This precludes the use of a fast solver for the implicit equations and makes the solution of this linear system more costly. Possible choices for solving the implicit equations are preconditioned iterative methods, or a sparse direct solver. In both cases, the work per grid point grows as the space mesh width is reduced. The truncation error of the Crank-Nicolson scheme in two dimensions is still $O(k^2, h^2)$. If V is independent of t, sparse Gaussian elimination is very attractive (provided sufficient memory is available), since we compute the LU factorization only once. By the same token, any pre-processing required for the preconditioner need only be performed once. For the time-dependent Schrödinger equation, where a sparse linear system has to be solved at each time step, iterative methods are better suited to exploiting a good initial guess, which can be generated by a predictor step. Closely related issues concerning the use of iterative methods for stiff systems of ordinary differential equations are discussed in [65] and [60].

6.2 The Split-step Fourier Method

The split-step approach discussed in Chapter 2 (Section 2) can be applied to the Schrödinger equation with more than one space dimension. In the case of two-space dimensions, the operators A and B are

$$A \equiv -i\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) = -i(D_x^2 + D_y^2) \tag{6.3}$$

and

$$B \equiv -iV(x, y). \tag{6.4}$$

As before, a split-step scheme based on SP_2 has the form

$$U^{n+1} = C_B\left(\frac{k}{2}\right) C_A(k) C_B\left(\frac{k}{2}\right) U^{n}, \tag{6.5}$$

where $C_A(k)$ and $C_B(k)$ are schemes for $u_t = Au$ and $u_t = Bu$, respectively (i.e., approximations to e^{kA} and e^{kB} respectively). The $O(k^2)$ accuracy of SP_2 is not altered as we go to higher dimensions.

If we write grid-valued functions, X, Y as two-dimensional arrays, then $Y = C_B(k)X$ can be written componentwise as

$$Y_{j,l} = e^{-ikV(x_j, y_l)} X_{j,l}, \quad \text{for} \quad j, l = 1, \ldots, m. \tag{6.6}$$

All split-step methods we will consider for the two-dimensional case will treat the potential V this way.

For the split-step Fourier method in two dimensions, the spectral approximation $C_A(k)$ is a direct analogue of the one dimensional case. We state it for completeness. Given a grid-valued, double-indexed vector X, we compute $Y = C_A(k)X$ as follows. First, we compute the coefficients $\tilde{X}_{r,s}$ of the truncated sine series for X:

$$X_{j,l} = \sum_{r=1}^{m}\sum_{s=1}^{m} \tilde{X}_{r,s} \sin\left(\frac{jr\pi}{m+1}\right) \sin\left(\frac{ls\pi}{m+1}\right), \quad \text{for} \quad j,l = 1,\ldots,m.$$

Next, the Fourier coefficients $\tilde{Y}_{r,s}$ of Y are given by

$$\tilde{Y}_{r,s} = e^{ik(r^2+s^2)\pi^2/L^2}\tilde{X}_{r,s},$$

where $L = (b-a) = (d-c)$. Finally we evaluate the double sine series for Y:

$$Y_{j,l} = \sum_{r=1}^{m}\sum_{s=1}^{m} \tilde{Y}_{r,s} \sin\left(\frac{jr\pi}{m+1}\right) \sin\left(\frac{ls\pi}{m+1}\right), \quad \text{for} \quad j,l = 1,\ldots,m.$$

The first and third steps each require a two-dimensional discrete sine transform, which can be implemented in terms of $2m$ one-dimensional fast sine transforms (m in each coordinate direction). This defines the split-step Fourier method in two space dimensions (SSF-2d).

The stability results for SSF-2d are the same as those for the SSF method. In particular, SSF-2d is unconditionally unitary (and hence stable).

This scheme has been implemented in the context of ocean acoustics [68].

6.3 Split-step Padé Schemes

The split-step finite difference methods we propose are based on the splitting SP_2, i.e., are of the form (6.5), and treat the potential V in the same way as the split-step Fourier (SSF-2d) method. We use Padé approximations to the second derivative operators with respect to x and y (D_x^2 and D_y^2, respectively) to construct schemes $C_A(k)$ for

$$u_t = -i(D_x^2 + D_y^2)u. \tag{6.7}$$

In the following, we will always approximate D_x^2 and D_y^2 with the same Padé approximation, i.e.,

$$D_x^2 \approx \frac{1}{h^2}\frac{\tilde{P}(\delta_x^2)}{\tilde{Q}(\delta_x^2)} \left(\equiv \frac{1}{h^2}\frac{\tilde{P}_x}{\tilde{Q}_x}\right), \qquad D_y^2 \approx \frac{1}{h^2}\frac{\tilde{P}(\delta_y^2)}{\tilde{Q}(\delta_y^2)} \left(\equiv \frac{1}{h^2}\frac{\tilde{P}_y}{\tilde{Q}_y}\right),$$

whereby the dependence of $\tilde{P}$ and $\tilde{Q}$ on (ν,μ) in the Padé approximation is not explicitly denoted. One could use approximations of different orders for D_x^2 and D_y^2, if this would result in greater efficiency.

Consider the following family of schemes $C_A(k)$ (parameterized by (ν,μ)) for use in the split-step approach (6.5),

$$\left[\tilde{Q}_x\tilde{Q}_y + \frac{ir}{2}(\tilde{P}_x\tilde{Q}_y + \tilde{P}_y\tilde{Q}_x)\right] U^{n+1} = \left[\tilde{Q}_x\tilde{Q}_y - \frac{ir}{2}(\tilde{P}_x\tilde{Q}_y + \tilde{P}_y\tilde{Q}_x)\right] U^n. \tag{6.8}$$

As in the one-dimensional case, all the schemes in this family are unconditionally stable for real $\tilde{P}$ and $\tilde{Q}$. The corresponding matrix is unitary and the split-step finite difference method obtained from such a scheme is thus unconditionally unitary.

For $(\nu,\mu) = (0,1)$, $(1,1)$ and $(2,2)$, we get the Crank-Nicolson (CN-2d), Crandall-Douglas (CD-2d) and FD(2,8)-2d schemes for (6.8), respectively. Each of these schemes is second order in time and has the same accuracy in space as the underlying Padé approximant. The corresponding split-step schemes, SS-CN-2d, SS-CD-2d and SS-FD(2,8)-2d, have the same order of accuracy (since the splitting error is also second order in time).

The matrices in the $C_A(k)$ part of SS-CN-2d, SS-CD-2d and SS-FD(2,8)-2d, (i.e., the matrices in (6.8)) have a special structure that can be exploited to reduce the cost of solving the implicit equations. This is an important advantage of the split-step approach which allows a separate treatment of the operators A and B defined in the previous section. For $(\nu,\mu) = (0,1)$, the matrix has the non-zero structure of the standard five-point discretization of the Laplacian, namely block tridiagonal, with tridiagonal matrices on the main (block) diagonal, and diagonal matrices on the off-diagonal blocks. The matrix for the $C_A(k)$ in the split-step Crandall-Douglas (SS-CD-2d) method ($(\nu,\mu) = (1,1)$) has the non-zero structure of the discrete Laplacian corresponding to a fourth-order nine-point stencil, namely, block tridiagonal, with tridiagonal matrices in all the blocks. The matrix for the split-step FD(2,8) scheme ($(\nu,\mu) = (2,2)$) is block pentadiagonal, with pentadiagonal matrices in each block, which corresponds to an eighth-order, 25-point discretization of the Laplacian.

The implicit equations for all three schemes can be solved using a straightforward extension of a "fast Poisson solver" based on FFTs in one direction and tridiagonal or pentadiagonal solves in the other (referred to as the matrix decomposition method in [59]). This is possible because the domain is a rectangle and the equation has constant coefficients. The key idea is that, for all three methods, the sub-matrices in each of the blocks are all diagonalizable by an FFT (more precisely, a fast sine transform). This is related to the fact that, on a rectangular grid, forming discrete sine transforms along the rows commutes with the operation of applying δ_y^2 along the columns. After performing FFTs in the x direction and re-ordering the unknowns by columns, we arrive at a set of tridiagonal systems, one for each grid column, for CN-2d and CD-2d and a set of pentadiagonal systems for FD(2,8). The principal difference with a standard fast Poisson solver is that a constant must be added to the main diagonal and that complex arithmetic is used. The operation count for a fast solver on an $m \times m$ grid is $O(m^2 \log_2 m)$ for the FFT part and $O(m^2)$ for the tri- (penta-) diagonal systems.

6.4 Split-step ADI Methods

To reduce the operation count, from $O(m^2 log_2 m)$ for SS-CN-2d and SS-CD-2d, we now consider alternating direction implicit (ADI) variants of these methods which have operation counts that

are $O(m^2)$. ADI methods for the diffusion equation have been studied extensively since the mid-1950s ([67], [64]). A closely related approach are the locally one-dimensional (LOD) schemes (see [16]). A review of ADI and LOD methods can be found in [1].

6.4.1 ADI schemes for $iu_t = u_{xx} + u_{yy}$

Using the notation of (6.8), we can write a family of ADI schemes for (6.7) as

$$\begin{aligned} \left[\tilde{Q}_x + \frac{ir}{2}\tilde{P}_x\right] U^* &= \left[\tilde{Q}_y - \frac{ir}{2}\tilde{P}_y\right] U^n, \\ \left[\tilde{Q}_y + \frac{ir}{2}\tilde{P}_y\right] U^{n+1} &= \left[\tilde{Q}_x - \frac{ir}{2}\tilde{P}_x\right] U^*, \end{aligned} \tag{6.9}$$

where the pair $(\tilde{P}, \tilde{Q})$ corresponds to a choice of (ν, μ). The matrices in (6.8), have the following approximate factorization:

$$\tilde{Q}_x\tilde{Q}_y \pm \frac{ir}{2}(\tilde{P}_x\tilde{Q}_y + \tilde{P}_y\tilde{Q}_x) = (\tilde{Q}_x \pm \frac{ir}{2}\tilde{P}_x)(\tilde{Q}_y \pm \frac{ir}{2}\tilde{P}_y) - \frac{r^2}{4}\tilde{P}_x\tilde{P}_y.$$

Thus, we can see that (6.9) differs from (6.8) in only the following term.

$$\frac{r^2}{4}\tilde{P}_x\tilde{P}_y(U^{n+1} - U^n).$$

This shows that each such ADI scheme is an $O(k^2)$ perturbation of the scheme in the class (6.8) with the same $\tilde{P}$ and $\tilde{Q}$. The point of introducing this perturbation is that it allows us to factor each of the matrices in (6.8) into two simpler ones. For Padé $(1,0)$, this corresponds to the well-known fact that the standard Peaceman-Rachford ADI method for the diffusion equation can be viewed as a second-order perturbation of the Crank-Nicolson method [1]. We will refer to this scheme as the PR scheme, for Peaceman-Rachford, although the name ADI-CN would be more consistent with our naming convention. The ADI scheme based on the Padé $(1,1)$ approximant is a second-order perturbation of the Crandall-Douglas scheme, CD-2d, and will be referred to as ADI-CD. Finally, ADI-FD(2,8) is the ADI scheme based on the Padé $(2,2)$ approximation to the second derivative operator.

For the schemes PR (= ADI-CN) and ADI-CD, the first half-step involves forming m tridiagonal matrix vector products along the columns and then solving m tridiagonal linear systems along the rows. The second half step has the same operations, with the roles of x and y reversed. The ADI-FD(2,8) scheme requires solving pentadiagonal systems along the rows and the columns of the grid on successive half steps. For all three ADI schemes, the work per time step is $O(m^2)$.

If we write (6.9) as

$$U^{n+1} = C_A(k)U^n,$$

then $C_A(k)$ is unitary. This means that the schemes (6.9) are unconditionally stable and furthermore, the split-step ADI methods for (SE-2d) based on these approximations to $\exp(kA)$ are

also unconditionally stable. We shall refer to these as SS-PR, SS-ADI-CD and SS-ADI-FD(2,8). These methods are second order in time and have the same accuracy in space as the underlying Padé approximant.

An improved implementation of ADI schemes that exploits a relationship between the matrices in (6.9) to reduce the operation count for each step is described in [66]. The extension of this idea to the Schrödinger equation can be described as follows. Suppose that, for a given U, U^1 is defined by

$$[\tilde{Q}_x + \frac{ir}{2}\tilde{P}_x]U^1 = [\tilde{Q}_x - \frac{ir}{2}\tilde{P}_x]U.$$

Then U^1 can be obtained more efficiently as follows:

$$[\tilde{Q}_x + \frac{ir}{2}\tilde{P}_x]U^* = \tilde{Q}_x U,$$

and

$$U^1 = 2U^* - U.$$

Thus, a matrix-vector product involving a complex tridiagonal matrix is replaced by one involving a real tridiagonal matrix.

In [62], an ADI method was stated for a generalization of the the standard diffusion equation in three space dimensions that includes the Schrödinger equation with $V = 0$ as a special case. In [61], an ADI scheme is analysed for the two-dimensional Schrödinger equation with $V = 0$ and a remark is made on how one might incorporate a non-zero potential, namely by going to a three level scheme, using Crank-Nicolson between the first and last time level, and treating the potential at the intermediate level. In [34], a higher order accurate ADI scheme was introduced for the diffusion equation.

6.4.2 Alternating sweep ordering in ADI methods

We now describe variants of the SS-ADI methods that are potentially better suited for implementation on parallel computers. The following idea is also applicable to the LOD variants.

We will refer to the half step in the ADI scheme that involves tridiagonal solves in the x-direction as the "X" half step and the other half step as the "Y" half step. Two successive ADI steps in the standard form can be denoted symbolically as (X, Y), (X, Y).

The idea of alternating sweep ordering is simply to interchange the order of the X and Y half steps on successive steps. In the notation introduced above, two successive steps of ADI with alternating sweep ordering (ASO) can be denoted symbolically as (X, Y), (Y, X). The point of this modification is that, in some approaches to implementing ADI methods on parallel computers, the ASO variant can lead to a reduction in the communication overhead.

Since δ_x^2 and δ_y^2 commute, the alternating sweep ordering does not alter the accuracy or stability of the underlying split-step ADI schemes.

Chapter 7

Solving Tridiagonal Linear Systems on a Hypercube Multiprocessor

In this chapter, we study the solution of tridiagonal systems of equations on hypercube multiprocessors. A number of methods for the Schrödinger equation require this operation (including the Alternating Direction Implicit methods) and this chapter prepares the ground for Chapter 8, where we will study hypercube algorithms for ADI methods.

We give a brief description of the hypercube architecture and the Intel iPSC/1 and iPSC/2, describe a performance model and review Gray codes, mesh embeddings and transposes on hypercubes. We then discuss hypercube methods for single and multiple tridiagonal systems.

7.1 Hypercube Multiprocessors

7.1.1 The hypercube architecture and the Intel hypercubes

A hypercube multiprocessor is a parallel computer with $P = 2^d$ processors, for some non-negative integer d, called the dimension of the hypercube. The processors P_j are numbered 0 through $P-1$ and each processor is connected to precisely d other processors. Let $j \in \{0, \ldots, P-1\}$ and let $j = (b_{d-1} \ldots b_1 b_0)_2$ be the binary representation of j. Then the *i-th neighbor* of P_j is processor $P_{n(i,j)}$ where $n(i,j)$ is the number whose binary representation differs from that of j in the i-th lowest bit only. For example, if $P = 64$ ($d = 6$), and we choose $j = 23$, i.e., $j = (10111)_2$, then the zero-th neighbor of P_{23} is $(10110)_2 = 22$, the first neighbor is $(10101)_2 = 21$, and the fourth neighbor is $(00111)_2 = 7$. Clearly, if P_k is the i-th neighbor of P_j then P_j is the i-th neighbor of P_k.

Some basic properties of hypercubes are discussed in [101], [98], and in some of the papers in [76] and [77].

We have conducted experiments on both the iPSC/1 and the iPSC/2. Both are MIMD (multiple instruction, multiple data) machines with distributed memories (four Megabytes per node). Communication between processors is achieved by message-passing. In practice, one often has the same program running on all the nodes but the MIMD property allows an overlapping

of different types of activity on different nodes through conditional statements that can be data-dependent or depend on the processor number. Both machines were programmed in FORTRAN.

Each iPSC/1 node consists of a processor board with a 80286/80287 CPU and floating point unit and four Megabytes of memory. Each node is capable of 30–40 Kflops (thousand floating point operations per second) in FORTRAN. Each communication activity has a start-up cost and a transfer time associated with it. For all practical purposes, communication and computation cannot be performed concurrently and the iPSC/1 cannot use multiple communication ports simultaneously.

The iPSC/2 nodes have a 80386 processor and the iPSC/2 on which our experiments were performed had a Weitek 1167 floating point chip on each node. Each node can achieve close to 0.5 Megaflops (million floating point operations per seconds) in FORTRAN.

7.1.2 The performance model

In this section, we describe the model we will use to predict the performance of parallel programs running on a hypercube.

Modeling the behavior of parallel programs is in general not an easy task. The total parallel time is determined by the dependencies in the algorithm which can extend across many processors in an arbitrarily complex, possibly data-dependent manner. For example, a program that passes a message around a ring embedded in the cube will take longer than one in which each processor simply exchanges data with its zero-th neighbor, even though each processor does one send and one receive to/from a nearest neighbor in both cases.

However, the problem becomes simpler when we restrict ourselves to reasonably structured algorithms. If, in addition, it is possible to map the computations onto the cube in a balanced manner, then a simple model will give a good estimate of the time required. In such cases, inter-node communication has a synchronizing effect and the parallel program can be analysed in terms of separate phases defined by the communication activities.

We now introduce some notation that will be used in our model and in the subsequent discussion.

The time to send n bytes of data to a nearest neighbor is

$$T_{comm}(n) = \sigma + n\tau, \tag{7.1}$$

assuming that messages are not packetized by the communication software. The iPSC/1 does in fact break messages down into $B = 1024$ byte packets, each of which has a start-up cost $\tilde{\sigma}$ associated with it. Thus a more accurate model for the iPSC/1 is

$$\tilde{T}_{comm}(n) = \sigma + \mathrm{mod}(n, B)\tau + \lfloor \tfrac{n}{B} \rfloor (\tilde{\sigma} + B\tau) = \sigma + \lfloor \tfrac{n}{B} \rfloor \tilde{\sigma} + n\tau. \tag{7.2}$$

On the iPSC/1, $\sigma \approx 1.5$ milliseconds, $\tilde{\sigma} \approx 0.75$ milliseconds, and $\tau \approx 1$ microsecond $= 0.001$ milliseconds.

Table 7.1:

Notation.

ω	time for one floating point operation
σ	start-up time for inter-node communication
τ	time to transfer each byte in inter-node communication
t_{copy}	time (per byte) to transpose a rectangular array
P	number of processors in hypercube
P_x	number of processors in processor mesh in x direction
$P_y = P/P_x$	number of processors in processor mesh in y direction
$d = \log_2 P$	dimension of hypercube
$d_x = \log_2 P_x$	number of dimensions in x direction in the processor mesh
$d_y = \log_2 P_y$	number of dimensions in y direction in the processor mesh
N_x, N_y	number of grid points in x, y directions, respectively.
$N = N_x\ N_y$	total number of grid points

This model makes no attempt to account for the variability in the communication times on the iPSC/1 for repeated runs with the same input parameters. This is caused by the non-repeatable pattern of failures in the sends or receives which cause the sending processor to timeout (on the order of 10 milliseconds!) and try again. Such a "failure" effects performance but not the correctness of the program, since the communication software guarantees that the message will eventually get through, provided sufficient system buffer space is available. The variability makes it necessary to use average times over a series of runs as the measure of performance. The *minimum* time over many runs is fairly repeatable and is closer to the model times than the average, probably because it corresponds to runs with few or no failures. The timings for the arithmetic phases do not exhibit the variability encountered in the communication phases.

On the iPSC/2, messages are not packetized and the variability in the timings is considerably lower. Thus, (7.1) is a reasonable model for the cost of communication with $\sigma \approx 0.5$ milliseconds. The cost of multi-hop communication is lower for the iPSC/2 than for the iPSC/1, but we have not exploited this feature.

Overlapping head-to-head communication: In virtually all the algorithms we present, communication occurs in the form of exchanges between nearest neighbors. On the iPSC/1, head-to-head communication cannot be overlapped. This is reflected in the estimates we give for various algorithms. On the iPSC/2, head-to-head communication can be overlapped, and the communication costs for exchanges are halved. Thus, for the model times given below, the factors in front of the T_{comm} terms should be halved for the iPSC/2.

Global data dependencies: Since the diameter of the hypercube is d, any hypercube implementation of an algorithm with global data dependencies must have a term that grows like $\log_2 P$. Typically, the $\log_2 P$ term involves the start-up parameter σ while the actual transfer time (the τ term) falls with P (for a fixed size problem, the message lengths decrease as we go to a higher dimensional cube). The arithmetic cost typically falls like $1/P$. For algorithms that fall into this class, the model gives an expression for the total cost that has the form

$$T(P) = \frac{A}{P} + B \log_2 P, \tag{7.3}$$

where A and B are positive and independent of P. Such a $T(P)$ has a minimum as a function of P. $T'(P^*) = 0 \Rightarrow P^* = (A/B)\ln 2$, so that using more than P^* processors will actually *increase* the running time. Furthermore, since $T''(P^*) = B^3/(A^2(\ln 2)^3)$, the minimum will be flat, if $B \ll A^{2/3}$.

7.1.3 Gray codes

A key element in designing parallel algorithms is keeping the amount of data movement between processors to a minimum which means that the mapping of grid points to processors must preserve locality. Methods for partial differential equations that employ finite differences usually have some local data dependency which means that neighboring grid points should be mapped

on to neighboring processors (if not the same processor). The most popular way of achieving this is the binary-reflected Gray code (BRGC) mapping which not only preserves the nearest neighbor property but has additional structure that makes it possible to exploit the hypercube interconnect in other phases of the algorithm.

Let $\Pi(P)$ denote the set of permutations of the set $\{0,\ldots,P-1\}$ and let $\gamma^{(P)} \in \Pi(P)$ denote the binary-reflected Gray code. If we write this permutation for P as

$$\gamma^{(P)} = \left(\gamma_0^{(P)},\ldots,\gamma_{P-1}^{(P)}\right),$$

where $\gamma^{(P)}$ takes $j \in \{0,\ldots,P-1\}$ to $\gamma_j^{(P)}$, then the permutation for $2P$ can be written as

$$\gamma^{(2P)} = \left(\gamma_0^{(P)},\ldots,\gamma_{P-1}^{(P)}, P+\gamma_{P-1}^{(P)},\ldots,P+\gamma_0^{(P)}\right).$$

For $P = 1,2,4,8$, we have

$$\gamma^{(1)} = (0), \quad \gamma^{(2)} = (0,1), \quad \gamma^{(4)} = (0,1,3,2) \quad \gamma^{(8)} = (0,1,3,2,6,7,5,4).$$

There are a number of permutations that are in some sense generalizations of the BRGC. In particular, there are $(\log_2 P)!$ permutations in $\Pi(P)$ that preserve the hypercube property *and* leave zero unchanged. (There are $P(\log_2 P)!$such permutations if we do not require them to leave 0 invariant [98].) If $\pi^{(P)}$ is such a permutation, then

$$\tilde{\gamma}^{(P)} = \gamma^{(P)} \circ \pi^{(P)}$$

might be called a *generalized* BRGC. For example, if $\rho^{(P)}$ is the permutation defined by bit-reversal, then $\gamma^{(P)} \circ \rho^{(P)}$, the *bit-reversed binary-reflected Gray code* can be used whenever the use of the BRGC would be appropriate.

7.1.4 Embedding a two-dimensional processor mesh in a hypercube

The hypercube topology has a rich interconnect that allows a number of different topologies to be embedded in a hypercube. We review some details of a mesh embedding, that is discussed in [112]. Suppose that we have an $N_x \times N_y$ computational grid. We subdivide the grid into blocks, with P_x and P_y blocks in the x and y directions, respectively, where P_x and P_y are powers of two and $P_xP_y = P$. If N_x and N_y are both powers of two, then all the blocks can be chosen to be the same size, with $(N_y/P_y) \times (N_x/P_x)$ grid points. (If N_x and N_y are not powers of two, then some blocks will have an extra row or column or both.) We label each block by (i,j), with $i \in \{0,\ldots,P_x-1\}$ and $j \in \{0,\ldots,P_y-1\}$, where the indices i and j correspond to the x and y directions, respectively, analogous to the natural labeling of the grid points by pairs of indices. For such i and j, we define

$$j||i \equiv jP_x + i,$$

which is the concatenation of the binary representations of i and j. The simplest embedding of a two-dimensional array into a hypercube is to assign block (i, j) to processor k, where $k = j||i$ (or equivalently, $k = i||j$). However, with this mapping grid points that are adjacent on the grid are not always mapped to processors which are nearest neighbors in the hypercube.

A second embedding is to pick k to be

$$k = \gamma^{(P_y)}(j)||\gamma^{(P_x)}(i),$$

where $\gamma^{(P)}$ is the binary-reflected Gray code. This embedding is illustrated in Figure 7.1 for $P = 64$, $P_x = P_y = 8$. All $\log_2 P$ neighbors of a processor lie in the same row or column in the processor mesh. The ordering that was defined for the set of neighbors of each processor induces an ordering on the sets of neighbors that lie in the same processor row or column, namely, the *i-th row-neighbor* is the i-th neighbor, for $i \in \{0, \ldots, \log_2 P_x - 1\}$ and the *j-th column-neighbor* is the $(j + dx)$-th neighbor, for $j \in \{0, \ldots, \log_2 P_y - 1\}$.

7.1.5 Transposes on the hypercube

We will now describe an algorithm for matrix transposition on a hypercube. We will discuss the behavior of this algorithm as predicted by our model and propose an improvement.

A number of authors ([102], [72], [100], [107], [93], [81], [113], [89]) have considered the problem of matrix transposition.

We will give an informal description of the algorithm in a simple case (see [81], [113]). Assume that we have a $P \times P$ matrix that is stored by rows on a P processor hypercube, using the binary-reflected Gray code, i.e. the p-th processor has the r-th row, where $p = \gamma^{(P)}(r)$. The object is to redistribute the data so that each processor ends up with one column of the matrix. This can be achieved in $\log_2 P$ steps. At the i-th step ($i = 0, ..., \log_2 P - 1$), each processor exchanges half of its data with its i-th neighbor. After the exchange, each processor reorders the unsent part of its data and the received data so as to make the data corresponding to a column of the matrix contiguous and ordered. We will refer to this local data permutation as a shuffle (cf. [102]). The exact form of the shuffle depends on the index i and the processor number. A typical case is illustrated in Figure 7.2.

One consequence of the shuffle is that all the data to be sent out at the next step lies entirely in the first or second half of the local array. The number of different columns that are represented in a processor is halved at each step and the number of elements per column is doubled at each step, keeping the amount of local data constant. The recursive structure of the hypercube makes this process particularly efficient. At the beginning, the data is distributed across the processors by rows, which implies that each column is spread across the entire cube. At the end of the first step, the data corresponding to the first $P/2$ columns lies in one sub-cube of dimension $(\log_2 P) - 1$ and the other $P/2$ columns lie in the other sub-cube of dimension $(\log_2 P) - 1$. At the end of the second step, each of the four sub-cubes of dimension $(\log_2 P) - 2$ contains $P/4$ complete columns. This process of "separation" is continued until the $(\log_2 P)$-th step, at the

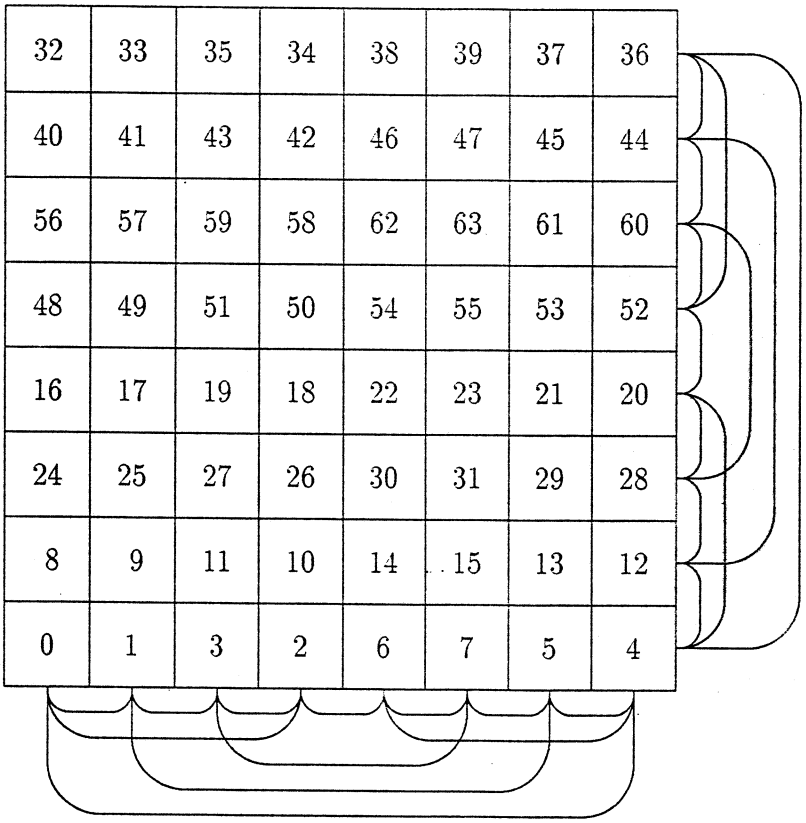


Figure 7.1: *An 8×8 processor mesh embedded in a 64 processor cube, using the binary-reflected Gray code. The links show the hypercube connectivity in the rows and columns. For example, processor 23 is connected to processors 22, 21, 19 (row neighbors) and 31, 55, 7 (column neighbors).*

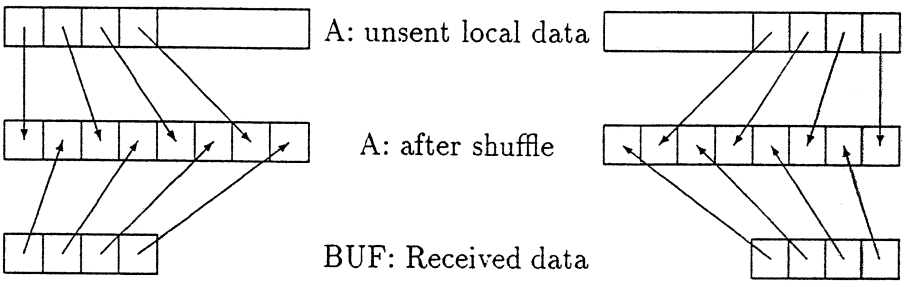


Figure 7.2: *Structure of the shuffle operation in a typical step in the hypercube transpose. Each processor tests the i-th bit in its number to determine which form of the shuffle to perform.*

end of which each processor has one entire column. The procedure described above can be stated in algorithmic form as follows.

A Transpose Algorithm for hypercubes.
$bit(i)$ is the i-th lowest bit in the binary representation of the "block number" blk which is related to the processor number p, by $p = \gamma^{(P)}(blk)$. The local data consists of P numbers stored in an array A, indexed from 0 to $P-1$.

for $i = d-1$ **step** -1 **to** 0
1. Exchange with i-th neighbor.
   1a. $dest = nbr(i)$
   1b. **if** (bit$(i) = 0$) **then** $j_0 = 0$ **else** $j_0 = P/2$
   1c. SEND $A(j_0)$ through $A(j_0 + P/2 - 1)$ to P_{dest}
   1d. RECEIVE $P/2$ numbers from P_{dest} in BUF
2. Shuffle unsent part of local data in A with received data in BUF

end for

If we stop the process after fewer than d steps, then we have an "incomplete transpose", with each column distributed across a subcube instead of on a single processor.

This method generalizes to an $N \times M$ matrix by imposing a $P \times P$ block structure on the matrix and transposing the individual blocks in an initial pass. In place of the matrix elements, one has to deal with sub-matrices.

The model times for the hypercube transpose of a total of B bytes of data is

$$T(B, P; \sigma, \tau, t_{copy}) = [2T_{comm}(B/(2 * P)) + (B/P)t_{copy}] \log_2 P. \tag{7.4}$$

This expression has one part that grows like $\log_2 P$ and another that falls like $(\log_2 P)/P$. For large problems or low start-up times, the cost of the transpose falls as we go to larger cubes but is bounded below by $2\sigma \log_2 P$.

The similarities between hypercube algorithms for the transpose and the fast Fourier transform (FFT) suggest a new transpose algorithm, the *Radix-4 hypercube transpose*. The idea is simply to combine two consecutive steps of the hypercube transpose (the "Radix-2 hypercube transpose") into a single step. At each step, each processor exchanges one quarter of its local data with each of the three processors that lie in a two-dimensional sub-cube, and then does a shuffle-like operation. For an even-dimensional cube, the model time for the Radix-4 hypercube transpose is

$$T(B, P; \sigma, \tau, t_{copy}) = \frac{\log_2 P}{2} \left[6T^{(2)}_{comm}(B/(4 * P)) + (B/P)t_{copy}\right], \tag{7.5}$$

where $T^{(h)}_{comm}(n) = \sigma + h \cdot n\tau$ models the cost of "h-hop" communication. The total cost of the shuffles is halved while the number of start-ups increases by 50%.

Table 7.2:
A comparison of the costs of three different variants of the hypercube transpose.

	$\sigma \log_2 P$	$B\frac{\log_2 P}{P}\tau$	$B\frac{\log_2 P}{P}t_{copy}$
Radix-2 (7.4)	2	1	1
Radix-4 (7.5)	3	3/2	1/2
Radix-8	14/3	7/4	1/3

One can visualize a Radix-8 or Radix-16 transpose along the lines of the Radix-4 version. Table 7.2 compares the costs of the Radix-2, 4 and 8 transposes.

This should be compared to a different improvement of the basic hypercube transpose algorithm proposed in [81], which also reduces the shuffle time at the cost of more communication. Their variant *doubles* the number of start-ups at each successive step while not performing the shuffle. When the increased communication cost exceeds the cost of the shuffle, one reverts to the basic algorithm for the remaining steps. The shuffle operation in the basic hypercube transpose algorithm essentially creates a single message buffer. The variant in [81] skips the buffering step when it is cheaper to communicate the separate "chunks" of data individually.

The Radix-4 transpose will lead to a greater saving for large P. Of course, one can combine the two approaches, using the variant of [81] for the initial steps and the Radix-4 transpose as the "basic" algorithm to revert to.

7.2 Hypercube Methods for a Single Tridiagonal System

In this section, we review methods for solving a single tridiagonal system of linear equations on a hypercube. We consider substructuring, together with a number of approaches to solving a single tridiagonal system, with one row per processor (the reduced system). We then describe cyclic reduction for hypercubes.

7.2.1 Substructuring

The idea of using substructuring to solve tridiagonal systems in parallel is a natural one and was used in [99], and subsequently in [108], where it is called the "partition" method. It is a form of block Gaussian elimination and is closely related to (one-dimensional) domain decomposition, and the Schur complement approach [74].

Consider the problem of solving a tridiagonal system of order N on a hypercube with P processors, where P divides N. We impose a block structure on the matrix, each diagonal block being tridiagonal of order N/P. At the internal block boundaries, there will be elements outside the blocks on either side of the diagonal which represent the coupling between the blocks. We assign the r-th block row to processor $p = \gamma^{(P)}(r)$, using the binary-reflected Gray code, $\gamma^{(P)}$.

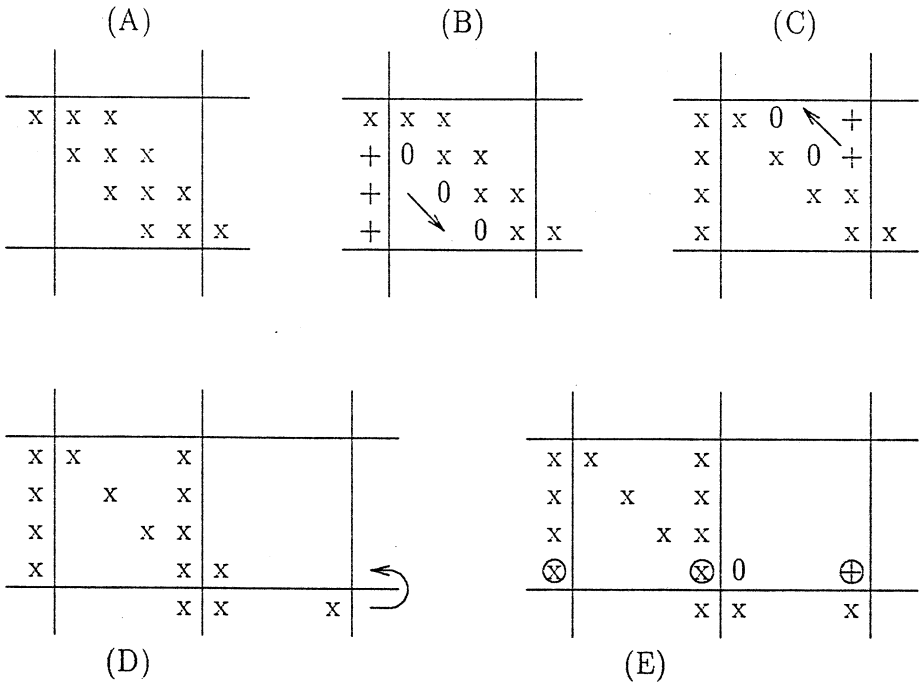


Figure 7.3: *The substructuring algorithm for tridiagonal matrices. 'x' indicates a non-zero element, '+' indicates a new non-zero and '0' stands for entries that are zeroed out in the elimination process. (A): The initial state of the j-th block row. (B): The forward elimination sweep. (C): The backward elimination sweep. (D): A multiple of the first row of the $(j+1)$-th block is added to the last row of the j-th block. (E): The state of the j-th block at the end of the first phase of substructuring. The circled entries are part of the reduced system.*

Figure 7.3 illustrates the steps in the substructuring algorithm [108]. First, each processor operates locally on its block, using Gaussian elimination to zero out the sub-diagonal elements in a forward elimination sweep and then zeroing the super-diagonal elements in a backward elimination sweep. These forward and backward sweeps each require six floating point operations per row and cause fill-in in the column to the left of the diagonal block and in the last column of the block. Now each processor sends its first row (four numbers) to the processor assigned to the previous block. The Gray code ensures that this is a send/receive between adjacent nodes. Each receiving processor adds a multiple of the received row to its last row, zeroing out the element to the right of the diagonal element.

The reduced system consists of the last row of each block (i.e., it has order P) and is distributed across the cube, with one row per processor. An important property of this procedure is that the reduced system is itself tridiagonal and may be solved by any applicable method.

After the reduced system has been solved, we perform a back-solve within each local block at a cost of 5 floating point operations per row and two nearest neighbor communications involving a single number.

The total cost of the substructuring phase is

$$T_{SS}(N,P;\omega,\sigma,\tau) = 17\frac{N}{P}\omega + 2[T_{comm}(16) + T_{comm}(4)] \tag{7.6}$$

When solving diagonally dominant matrices with constants along the diagonals, the reduced system is more diagonally dominant than the original system ([108]). In the most favorable case, it is essentially diagonal, and can be solved with no additional communication!

Substructuring requires about twice as many arithmetic operations per row as standard Gaussian elimination. This causes a significant drop in the speed-up when the substructuring phase dominates the total cost (large N and small P).

We now consider the problem of solving a tridiagonal system of order P spread across a hypercube with P processors. This is precisely the problem that arises when we use the substructuring approach, namely the reduced system.

7.2.2 Cyclic reduction

The cyclic reduction method for tridiagonal matrices is due to Golub and Hockney (see [83]). A number of authors have discussed its implementation on vector and parallel computers (see the references in [95]). We state the sequential (point) cyclic reduction algorithm to set up the notation for describing the parallel versions (see also [78]).

Sequential Cyclic Reduction for tridiagonal systems, $Ax = d$
We assume that the matrix is stored by diagonals and denote the k-th row of A by $Row(k) = (a(k), b(k), c(k))$, for $k = 0, 1, \ldots, N-1$. For simplicity, we suppose that $N = 2^d$ ($a(0) = c(N-1) = 0$).

The Reduction Phase:
for $j = 1$ **step** 1 **to** d
for $k = 0$ **step** 2^j **to** $N-1$
REDUCE $(Row(k), j)$
end for
end for
Solve for $x(0)$:
$x(0) = d(0)/b(0)$
The Back-substitution Phase:
for $j = d$ **step** -1 **to** 1
for $k = 2^{j-1}$ **step** 2^j **to** $N-1$
BACK-SUB $(Row(k), j)$
end for
end for

The following are the REDUCE and BACK-SUB operations for a generic (k, j) pair. It as assumed that arithmetic operations involving array indices outside the range $0, 1, \ldots, N-1$ are not performed. The required conditionals are omitted for notational simplicity.

REDUCE $(Row(k), j)$:

$$m_1 = a(k)/b(k-2^j)$$
$$m_2 = c(k)/b(k+2^j)$$
$$b(k) = b(k) - m_1 * c(k-2^j) - m_2 * a(k+2^j)$$
$$d(k) = d(k) - m_1 * d(k-2^j) - m_2 * d(k+2^j)$$
$$a(k) = -m_1 * a(k-2^j)$$
$$c(k) = -m_2 * c(k+2^j)$$

BACK-SUB $(Row(k), j)$:

$$x(k) = (d(k) - a(k) * x(k-2^j) - c(k) * x(k+2^j))/b(k)$$

The key idea of the hypercube implementation of cyclic reduction is to note that for each j in the sequential algorithm, the iterations of the inner loop are independent, and can be performed in parallel. This reduces the arithmetic cost roughly from P to $\log_2 P$ Reduce/Back-sub operations. We now discuss the communication structure of the parallel algorithm that achieves this.

The structure of the hypercube algorithm is more complicated because we have to view the algorithm from each processor's point of view, rather than from a global point of view. It would clearly be desirable to have a parallelizing compiler that would allow the algorithm to be expressed in a manner similar to the statement of the sequential algorithm.

We assume that the rows of the matrix are numbered 0 through $P-1$ and are assigned to processors using the binary-reflected Gray code. Since we are solving P equations on P processors, we require $\log_2 P$ steps in the reduction and substitution phases. Twelve and five floating point operations per row are required for each Reduce and Back-sub step, respectively. The Gray code allocation scheme ensures that processors that need to communicate are never more than distance two apart.

However, we implement cyclic reduction *with exchanges* [85]. Rows are exchanged in the reduction phase, so that all processors that are to be active in the next step lie in a lower dimensional sub-cube and need to communicate only with nearest neighbors. The exchanges are reversed in the substitution phase. At the end of the reduction phase, Processor 0 solves for the zero-th unknown of the reduced system.

Cyclic Reduction on a Hypercube: The Reduction Phase

We suppose that we are given a tridiagonal matrix of order P, distributed across a P processor hypercube. The k-th row is assigned to processor p, where $p = \gamma^{(P)}(k)$. Before performing a Reduce operation, a processor must receive rows numbered $k-2^j$ and $k+2^j$. "Left" and "right" refer to the neighboring processors that have rows $k-2^j$ and $k+2^j$, respectively. The Receives corresponding to row numbers outside the range $0, 1, \ldots, P-1$ (and the associated Sends) are not performed. The required conditionals are omitted for notational simplicity.

$$k = (\gamma^{(P)})^{-1}(p)$$

```
for j = 0  step 1  to d − 1
    if p(mod 2^j) ≡ 0 then
        if p/2^j is odd then
            SEND Row(k) "left"
            SEND Row(k) "right"
        else if p/2^j is even then
            RECEIVE Row(k − 2^j) "from left"
            RECEIVE Row(k + 2^j) "from right"
            REDUCE (Row(k), j)
        end if
        if p/2^j and k/2^j have opposite parities then
            Exchange row with nbr(j):
                SEND row to nbr(i)
                RECEIVE row from nbr(i)
            Reset k to be row number of the received row.
        end if
    end if
end for
```

The outermost conditional "turns off" all but $P/2^j$ processors. This reflects the decreasing processor utilization in the reduction phase. Half of the active processors are involved in exchanges.

The back-substitution phase has a very similar structure, with the order of the steps reversed (and a Back-Sub rather than a Reduce operation).

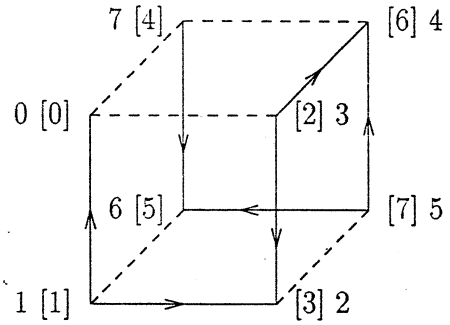
In Figure 7.4, we illustrate the pattern of processor activity and communication in the reduction phase, when solving an 8x8 system on a 3-cube. The numbers of the active processors are given in brackets, along with the row they are handling. Links involved in communication are indicated.

As stated above, the parallel algorithm requires four start-ups for the communication in each reduction and substitution step (a send and a receive for the exchange and two sends or two receives for the cyclic reduction process). However, one of the two sends or two receives is redundant for processors that are involved in an exchange, since a row is sent twice to the same neighbor. If the redundant send/receive pairs are not performed, cyclic reduction with exchanges can be implemented with three start-ups for each reduction and substitution step. This saving is non-negligible on a hypercube like the iPSC/1 where the start-up cost is high relative to the arithmetic speed. The parallel cost of cyclic reduction on a hypercube is given by

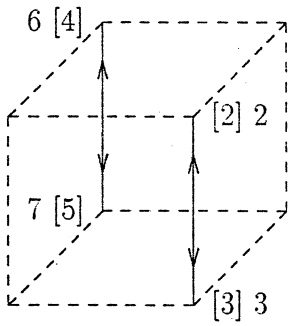
$$\begin{aligned} T_{CR}(P;\omega,\sigma,\tau) &= [17\omega + 3T_{comm}(16) + 3T_{comm}(4)]\log_2 P - 5\omega - 4T_{comm}(12) \\ &\approx (t_{arith} + t_{comm})\log_2 P = t_{cr}\log_2 P, \end{aligned} \tag{7.7}$$

where

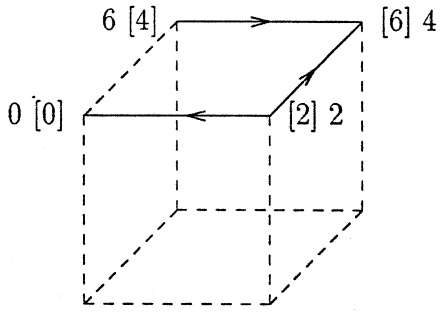
$$t_{arith} = 17\omega, \quad t_{comm} = 6\sigma + 60\tau, \quad \text{and} \quad t_{cr} = t_{arith} + t_{comm}. \tag{7.8}$$



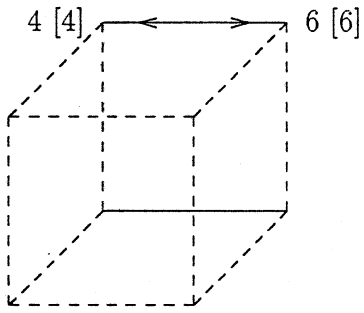
Step 0



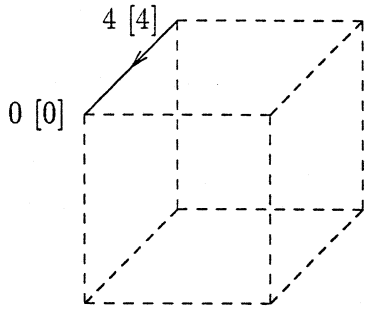
Exchange in Step 0



Step 1



Exchange in Step 1



Step 2

Figure 7.4: *Pattern of processor activity and communication in the reduction phase of cyclic reduction with exchanges on a hypercube, for an* 8×8 *tridiagonal system. The processor numbers are given in brackets. The accompanying number is the index of the row being handled by the processor at that step. Communication occurs along links that are drawn as solid lines. The back-substitution phase has a very similar communication pattern, with the order of the steps reversed.*

If N is a power of 2, one could perform $\log_2(N/P)$ steps of cyclic reduction to the *entire* system and also arrive at a reduced system of order P on P processors. The arithmetic cost of this approach would be about the same as that for substructuring. However, this "global" cyclic reduction would require $\log_2(N/P)$ nearest neighbor communications as opposed to one for substructuring, and hence substructuring is preferable. Substructuring has the additional advantage of being more flexible in terms of the choice of N.

For sufficiently diagonally dominant matrices, the system obtained after a few steps of cyclic reduction is essentially diagonal ([78]) which can be exploited to reduce both arithmetic and communication. The potential gain is greater for hypercubes than for serial computers because of the savings in communication.

A disadvantage of cyclic reduction on a hypercube is that many processors are idle in the course of the computations. The processor-time product for the hypercube algorithm is asymptotically $O(P \log_2 P)$ as opposed to $O(P)$ for the sequential algorithm, which indicates that the fraction of processors that are active on average is $O(1/\log(P))$.

We denote by SS/CR the method obtained by combining substructuring and cyclic reduction. This method is described in [87] where it is called GECR. The cost of the SS/CR method is

$$\begin{aligned} T_{SS/CR}(N,P;\omega,\sigma,\tau) &= T_{SS}(N,P;\omega,\sigma,\tau) + T_{CR}(P;\omega,\sigma,\tau) \\ &\approx 17\tfrac{N}{P}\omega + [17\omega + 6\sigma + 60\tau]\log_2 P \\ &= t_{arith}\tfrac{N}{P} + [t_{arith} + t_{comm}]\log P \end{aligned} \tag{7.9}$$

The expression in (7.9) has a minimum as a function of the number of processors, and the optimal number of processors, P^*, for SS/CR satisfies

$$\ln 2\frac{N}{P^*} = \frac{[6\sigma + 60\tau + 17\omega]}{17\omega} = 1 + \frac{t_{comm}}{t_{arith}}, \tag{7.10}$$

This information can be stated in a number of different ways.

- For a given set of hypercube parameters (ω,σ,τ), there is an optimal number of rows that should be assigned per processor (N/P^*). For the iPSC/1 parameters, we get a value of about 20 rows per processor. When communication time dominates arithmetic time, there is a penalty associated with using more processors and the optimal block size is correspondingly larger. In particular, if the arithmetic speed is increased by a certain factor (e.g., by installing new hardware) and the communication costs are kept fixed or improved by a smaller factor, the number of processors that can be usefully employed for a given N will be smaller.

- When t_{comm} is small compared to t_{arith}, $P^* \approx N \ln 2$.

- The optimal number of processors is asymptotically $O(N)$.

- As σ is reduced (or N increased), the function (7.9) becomes flatter in a neighborhood of the optimal P (cf. (7.3)). In this situation, the time required on $P^*/2$ processors will be only slightly more than that on P^* processors.

- For a system of order $N = 2^n$, the total time is minimized on a cube of dimension $n - r$, where $r = \log_2(t_{cr}/t_{arith}) + \log_2 \log_2 e$.

The minimum time for SS/CR is bounded by $t_{cr}\,(\log_2 P^* + 1)$ Thus, t_{cr} is a measure of the cost of solving a tridiagonal system on a hypercube. With the implementation we have outlined, and for the iPSC/1 parameters, $t_{cr} \approx 9$ milliseconds.

7.2.3 The CR(ℓ) hybrid scheme

Since the processor utilization for cyclic reduction falls as the reduction process progresses, it is natural to consider switching to a different method after a few steps of cyclic reduction (see [83], [104]). In this section, we describe a scheme, CR(ℓ), in which we solve a tridiagonal system of order P on a P processor cube by performing ℓ steps of cyclic reduction, $\ell \in \{0, \ldots, \log_2 P\}$, assembling the (tridiagonal) reduced system to a single processor, solving it by standard Gaussian elimination, distributing the solution back, and then performing the ℓ steps of the substitution phase of cyclic reduction. This scheme can be used for the reduced system obtained after substructuring.

For arbitrary ℓ between 0 and $\log_2 P - 1$, the time for CR(ℓ) is given by

$$T_{CR(\ell)}(P, \ell; \omega, \sigma, \tau) \approx \ell t_{cr} + 2\sigma(\log_2 P - \ell) + 20(\frac{P}{2^\ell} - 1)\tau + 9(\frac{P}{2^\ell} - 1)\omega. \tag{7.11}$$

$T_{CR(\ell)}$ is minimized with respect to ℓ when

$$2^\ell = P\frac{[20\tau + 9\omega]}{(t_{cr} - 2\sigma)} \ln 2,$$

giving

$$\ell_{\text{opt}} = max\{0,\ \log_2 P - \nu(\omega, \sigma, \tau)\},$$

where

$$\nu(\omega, \sigma, \tau) = \log_2 \log_2 e + \log_2 \left(\frac{4\sigma + 60\tau + 17\omega}{20\tau + 9\omega}\right).$$

Thus the optimal number of cyclic reduction steps in SS/CR(ℓ) is the cube dimension minus a constant (up to rounding effects). Lowering the start-up time relative to the arithmetic time will reduce ν, and thereby increase the optimal number of cyclic reduction steps.

In Figure 7.5, we show the running time of SS/CR(ℓ) on the iPSC/1 using the simpler (8 start-ups per step) version of cyclic reduction with exchanges. The times are for a tridiagonal system of order 4096 on a 32 processor cube, with the number of CR steps ranging form 0 through $\log_2 P$. For each value of ℓ, we have given the times of 30 independent runs to demonstrate the variability in the iPSC/1 times. The solid line in Figure 7.5 represents the average times and the dashed line links the *minimum* times. The average times include the effects of message collisions/failures and the minimum times probably represent runs on which few collisions occurred. For $\ell = 0$, the difference between the largest and smallest times is about 125 milliseconds. The minimum times indicate a value of ν around 3 or 4, whereas the average times suggest that ν is at least 5.

Remark. The CR(ℓ) method can be extended to take advantage of the BABE algorithm [88].

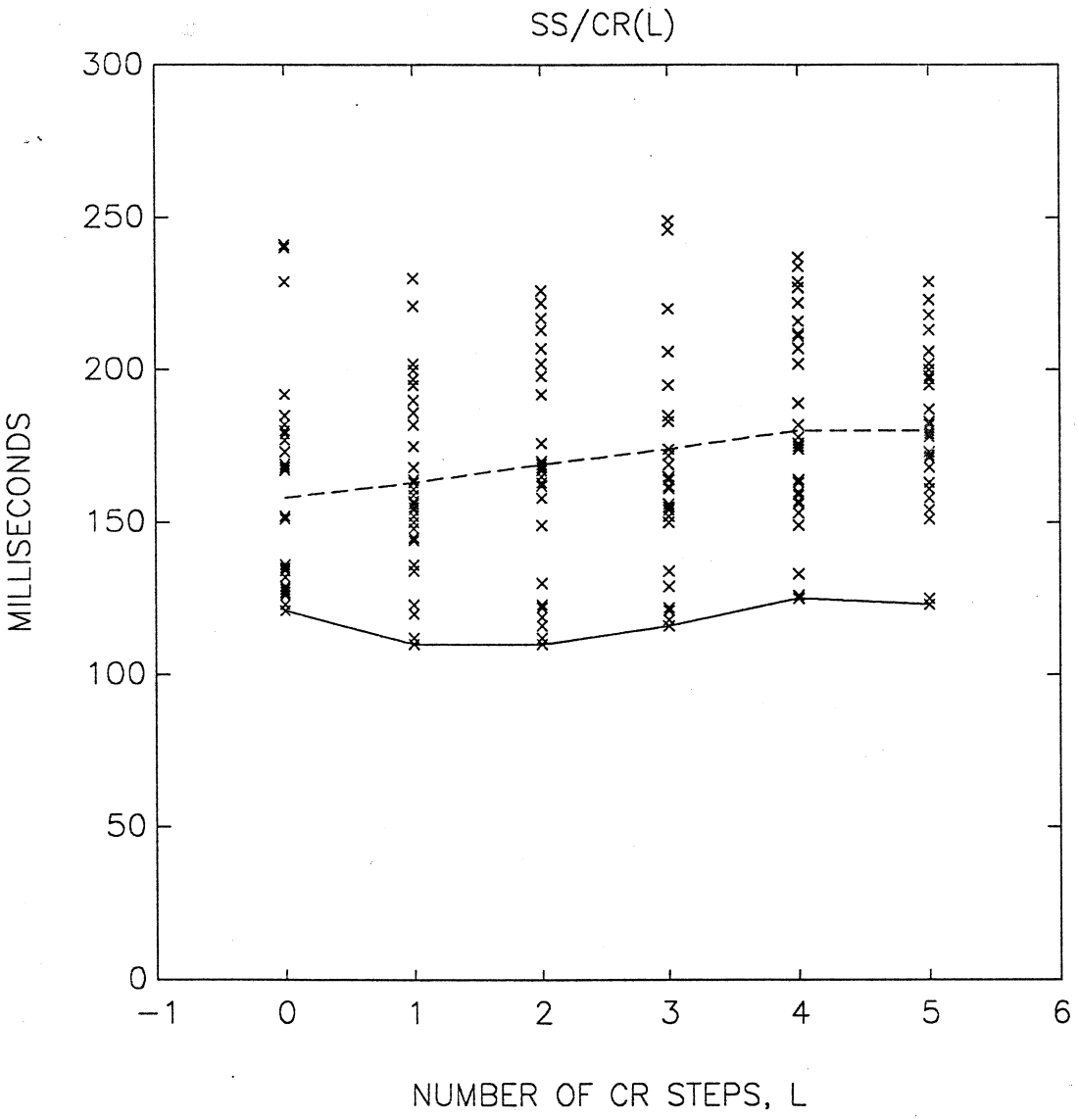


Figure 7.5: *iPSC/1 times for the SS/CR(ℓ) method, with* $N = 4096$ *and* $P = 32$. *The times for* 30 *independent runs are plotted for each value of* ℓ. *The solid line corresponds to the average times and the dashed line to the minimum times.*

7.2.4 Cyclic elimination

Cyclic elimination (CE) is a modified form of cyclic reduction in which all the rows are involved in all $\log_2 P$ steps [78] . The characteristic property of cyclic elimination is that the super and sub-diagonals of the original tridiagonal matrix "move away" from the main diagonal in the course of the algorithm and a tridiagonal matrix of order P is transformed into a diagonal matrix in $\log_2 P$ steps. Cyclic elimination has a higher sequential arithmetic complexity ($O(P \log_2 P)$) than cyclic reduction ($O(P)$) [78], i.e. it is not *consistent* in the sense of Lambiotte and Voigt ([91]) and is therefore not competitive with cyclic reduction on sequential (or vector) computers. However, on hypercubes, cyclic elimination has the same complexity as cyclic reduction. The main advantage of cyclic elimination is the fact that it requires just $12 \log_2 P$ floating point operations per row as opposed to $17 \log_2 P$ for CR. Thus, one would prefer CE on cubes on which arithmetic is slow compared to communication. CR is always superior to CE on the iPSC/1, the reason being the high cost for communication [88]. Cyclic elimination has been used by a number of authors for parallel computers, such as the ICL DAP ([111]), the Goodyear MPP ([110]), the TRAC computer ([90]).

The PARACR method of Hockney and Jesshope [84] is essentially the cyclic elimination method.

Some other approaches to the parallel solution of tridiagonal systems are discussed in [103], [80], [105], [75], [112], and [96].

7.3 Hypercube Algorithms for Multiple Tridiagonal Systems

In this section, we will consider the problem of solving M independent tridiagonal systems each of order N on a hypercube with P processors. Multiple tridiagonal systems arise, for example, in fast Poisson solvers based on matrix decomposition that use FFTs in one direction, and solve tridiagonal systems in the other direction. Also, in ADI methods, which we will discuss in greater detail in the next section, one solves a number of independent tridiagonal systems along rows and columns of a rectangular grid.

Substructuring for multiple tridiagonal systems. We will restrict ourselves in this section to cubes with a moderate degree of parallelism ($P \leq N$). In this situation, substructuring is an obvious choice. We will assume that all systems are identically distributed across the cube and that substructuring for each system is done in the manner described earlier for a single system. We arrive at M reduced systems, each of order P, and each distributed across all P processors. We will describe several approaches to solving this set of reduced systems. As in the single system case, there is a penalty associated with substructuring in terms of the operation count and any method using substructuring will operate at 50% efficiency on a two processor cube. Since the efficiency typically falls with increasing cube size, one would not expect to achieve more than 50% efficiency when substructuring is used. We will also describe a method that does not perform substructuring.

7.3.1 Unbalanced cyclic reduction

First we describe SS/UnBalCR, a simplistic extension of cyclic reduction to solve the reduced systems in which all M systems are treated identically. Thus, the computational load on each processor is increased by a factor of M, as is the volume of data moved in each communication activity. The number of start-ups and the pattern of processor activity is the same as that for SS/CR. The cost of solving M systems each of order N, on a cube with P processors is

$$\begin{aligned} T_{SS/UnBalCR}(M,N,P;\omega,\sigma,\tau) &= 17\frac{MN}{P}\omega + 6\sigma\log_2 P + [60\tau + 17\omega]M\log_2 P. \\ &+ [60\tau + 17\omega]M\log_2 P. \end{aligned} \tag{7.12}$$

The $\log_2 P$ factor in final term is a source of inefficiency which is why we will now discuss balanced cyclic reduction.

7.3.2 Balanced cyclic reduction

We now describe SS/BalCR, in which the reduced systems obtained from substructuring are solved by a balanced form of cyclic reduction.

For convenience, let $M = P$. For $M > P$, one simply replaces each equation by an equation group. Our implementation is designed to handle $M = 2^n$, and a generalization to arbitrary M could be done at the expense of introducing some load imbalance. In the following, we assume that like the processor numbers, the system and row indices lie in the range $0, \ldots, P-1$. Assume that the p-th processor has the r-th row of each system, where $p = \gamma^{(P)}(r)$. The reduction phase consists of $d = \log_2 P$ steps.

In cyclic reduction for a single system, at the i-th step ($i = 0, 1, \ldots, d-1$), the odd numbered rows are used to perform the Reduce operation on the even numbered rows, where the definition of odd/even depends on i. At the end of the d-th step, we are left with one equation with one unknown (the 0-th row). We will refer to this row as the "target" row and say that the cyclic reduction process "converges" to it.

If we alter the algorithm (for a single system) so that the even numbered rows modify the odd numbered rows at each step, the cyclic reduction process would "converge" to the last row (the $(P-1)$-th row).

If the decision as to whether the odd rows or the even rows are to be modified is varied from step to step, then we can "steer" the cyclic reduction process and make it "converge" to an arbitrary target row.

In the context of solving multiple tridiagonal systems, this strategy can be used to implement cyclic reduction in a load-balanced manner. Each of the P tridiagonal systems (or groups of systems) is made to converge to a different target row. Since all the systems are identically distributed, the cyclic reduction process will converge to different processors for different systems (or groups of systems). To make this idea work on a hypercube involves a certain amount of "messy" index computations, involving processor numbers, the indices of certain groups of the tridiagonal systems and row indices within each tridiagonal system.

One natural way to "steer" the cyclic reduction process is to take the binary representation of the system number $s = b_{d-1} \ldots b_1 b_0$, and at the i-th step modify the even rows, if b_i is 0, and the odd rows, if b_i is 1. It is easy to see this rule results in the cyclic reduction process converging to the s-th row.

Alternatively, if we traverse the bits in the binary representation of s in the opposite order (i.e., high to low), and modify the even rows at the i-th step if $b_{d-i-1} = 0$ (and the odd rows otherwise), then the process would converge to the r-th row, where r is obtained from s by bit-reversal.

We introduce a notation for denoting certain groups of the systems. For $0 \leq i \leq d-1$ and $0 \leq j \leq 2^i - 1$, let $G_j^{(i)}$ be the subset of $\{0, 1, \ldots, P-1\}$ consisting of an even number of consecutive integers. The first and second halves of $G_j^{(i)}$ are denoted by $G_{j,0}^{(i)}$ and $G_{j,1}^{(i)}$ respectively. We set $G_0^{(0)} = \{0, 1, \ldots, P-1\}$. Thus $G_{0,0}^{(0)} = \{0, 1, \ldots, \frac{P}{2} - 1\}$ and $G_{0,1}^{(0)} = \{\frac{P}{2}, \ldots, P-1\}$. The subsequent groups are defined by

$$G_j^{(i+1)} = G_{j,0}^{(i)}, \qquad G_{j+2^i}^{(i+1)} = G_{j,1}^{(i)}.$$

$G_j^{(i)}$ consists of $P/2^i$ consecutive integers, beginning with bit-reversed(j). For $P = 8$, the $G_j^{(i)}$ are the following.

$$\begin{array}{llll} G_0^{(0)} = \{0,1,2,3,4,5,6,7\}, & & & \\ G_0^{(1)} = \{0,1,2,3\}, & & G_1^{(1)} = \{4,5,6,7\}, & \\ G_0^{(2)} = \{0,1\}, & G_1^{(2)} = \{4,5\} & G_2^{(2)} = \{2,3\}, & G_3^{(2)} = \{6,7\}. \end{array}$$

We introduce an auxiliary variable 'label' with values between 0 and $P-1$ that is used in deciding what a processor does at each step. The label for processor p is initially set to $(\gamma^{(P)})^{-1}(p)$, where $(\gamma^{(P)})^{-1}$ denotes the inverse of the binary-reflected Gray code. At the i-th step, $(0 \leq i \leq d-1)$, the cube is divided into 2^i sub-cubes of dimension $d-i$ each. Each sub-cube consists of processors whose labels are equal modulo 2^i. Each sub-cube operates on $P/2^i$ systems, namely those in $G_j^{(i)}$, where $j \equiv$ label mod 2^i. Within each sub-cube, the processors for which label$/2^i$ is even modify the first half of the systems assigned to that sub-cube $(G_{j,0}^{(i)})$, while the processors for which label$/2^i$ is odd modify the second half $(G_{j,1}^{(i)})$. For each sub-cube,

$$\frac{\mathrm{p}}{2^i} = \gamma^{(P)}\left(\frac{\mathrm{label}}{2^i}\right). \tag{7.13}$$

where division by 2^i is integer division. To maintain the relationship between p and label expressed in (7.13), each time a processor is involved in an exchange, its label is reset to that of its exchange partner, as in the single system case. The groups that are to be operated on in a sub-cube have their rows distributed according to the Gray code, whereby a linear array is embedded in each sub-cube. This defines a left and right neighbor for each processor, except for the zero-th and last which have only one neighbor (right and left, respectively). Note that

the left and right neighbors of a processor change with i. The following code is executed by all processors during the reduction phase.

BalCR on a Hypercube: The Reduction Phase

p is the processor number. Edge processors in the linear arrays that are embedded in each sub-cube have only one (left or right) neighbor, and hence do only one SEND and one RECEIVE. The required conditionals are omitted for notational simplicity.

$label = (\gamma^{(P)})^{-1}(p)$
for $i = 0$ step 1 to $d-1$
  $j = label(\text{mod} 2^i)$
  if $(label/2^i$ even) **then** $l = 0$; $m = 1$ **else** $l = 1$; $m = 0$ **endif**
  id of left neighbor $= \gamma^{(P)}(label/2^i - 1) * 2^i + j$
  id of right neighbor $= \gamma^{(P)}(label/2^i + 1) * 2^i + j$
  SEND $G^{(i)}_{j,m}$ left and right
  RECEIVE $G^{(i)}_{j,l}$ from left and right
  REDUCE $G^{(i)}_{j,l}$ using received rows
  if (label/2^{i+1} odd) **then**
    Exchange with $nbr(i)$
      SEND $G^{(i)}_{j,l}$
      RECEIVE the corresponding group
    Reset label to that of $nbr(i)$:
      $label = 2^i * (\gamma^{(P)})^{-1}(nbr(i))/2^i + j$
  endif
end for

At the end of the reduction phase, the label of each processor is equal to p. Processor p solves for the p-th unknown of the s-th system, where p = bit-reversed(s).

In Figure 7.6, we show which rows are modified at each reduction step when solving 8 tridiagonal systems each of order 8, on a 3-cube. We show only systems 3 and 6. For system 3, in successive steps, the odd, even and even rows, respectively are modified, and the reduction process converges to row 6, processor 6. For system 6, the order is odd, even, even, and the reduction process converges to row 3, processor 3. System 1, which is not depicted in Figure 7.6 would converge to its 4-th row on processor 4, since 4 = bit-reversed(1). Furthermore, from the binary encoding of 4 (100), we can tell that in the 3 reduction steps, the even, even and odd rows, respectively will be modified.

In Figure 7.7, we illustrate the steps in the reduction phase of BalCR when solving 8 tridiagonal systems each of order 8, on a 3-cube. In particular, we show the values of the labels for each processor, the sub-cubes that are active at each stage, the communication paths and the system groups $G^{(i)}_{j,0}$, $G^{(i)}_{j,1}$ that are modified at each step.

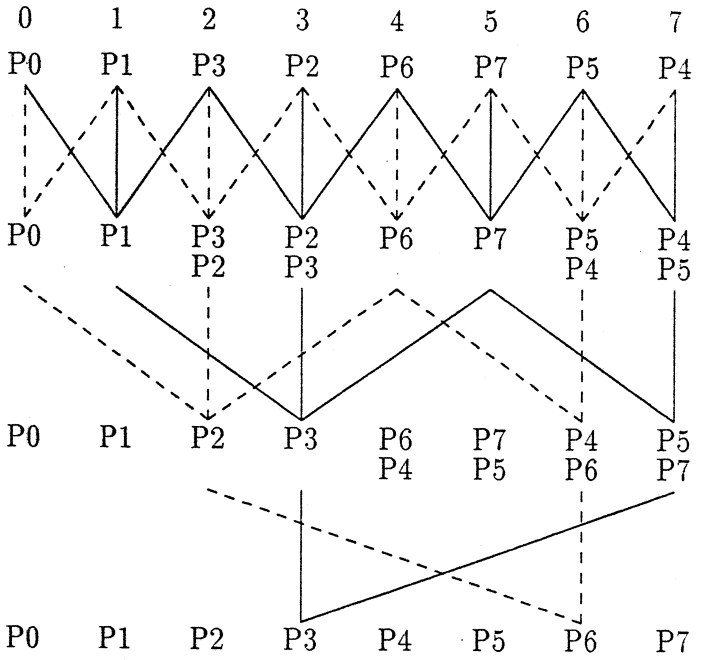
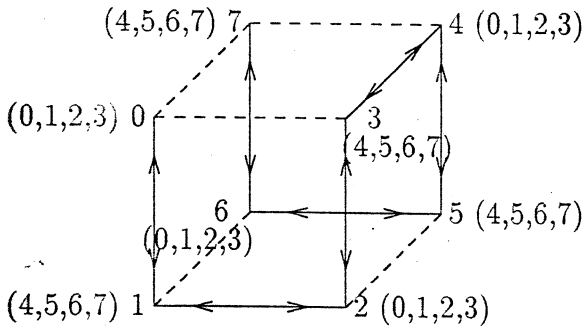
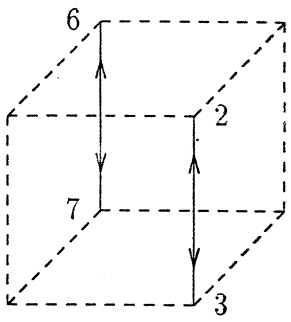


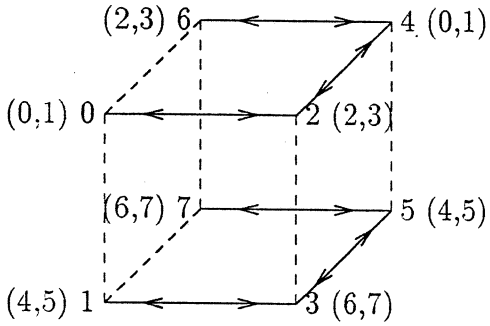
Figure 7.6: *The relationship between processors, systems and rows in balanced cyclic reduction on an* 8 *processor hypercube with* 8 *systems of order* 8 *each. Only systems 3 (dashed line) and 6 (solid line) are shown. Each column in the diagram corresponds to a row of the matrix. Pn denotes Processor n. Vertical lines indicate active rows. Exchanges at the end of steps* 0 *and* 1 *are indicated by a change in the processor number for a given row. This figure shows that for System 6, BalCR converges to Row 3 on Processor 3. For System 3, BalCR converges to Row 6 on Processor 6.*



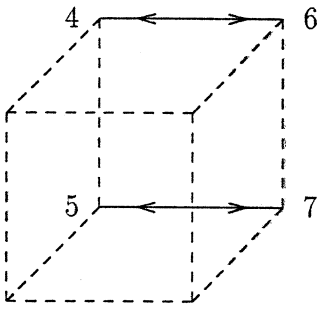
Step 0



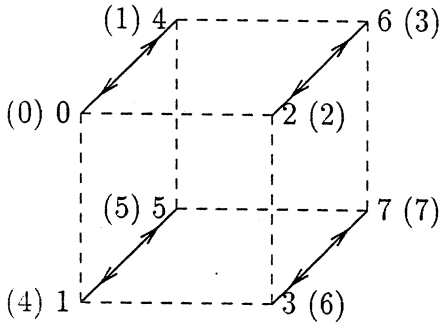
Exchanges in Step 0



Step 1



Exchanges in Step 1



Step 2

Figure 7.7: *Pattern of processor activity in the reduction phase of SS/BalCR when solving 8 systems on a 3-cube. The rows of each system are initially distributed according to the binary-reflected Gray code. The number at each vertex is the "label" of that processor, the numbers in parentheses are the indices of the systems whose rows are to be modified at the current step. Communication occurs along links that are drawn as solid lines. In Step 2, the labels coincide with the processor numbers.*

At each step except the last, there are processors that perform two sends, two receives and are involved in an exchange, which requires one send and one receive, giving a total of six start-ups per step. In the final step, there are no exchanges and each processor does one send and one receive only. At the i-th step, each communication activity, including the exchanges, involves $M/2^{i+1}$ systems. Thus the total data shipped over d sends is the data associated with

$$\sum_{i=0}^{d-1} \frac{M}{2^{i+1}} = M(1 - \frac{1}{P})$$

systems, namely $16M(1 - \frac{1}{P})$ bytes, since we send four floating point numbers per row. This is the reason why the $\log_2 P$ factor in front of the τ and ω (7.12) drops out for SS/BalCR. The exchanges in the substitution phase are the same as those in the reduction phase in reverse order, so that at the end of the two phases, all the data is back to its initial distribution.

We state the steps that have to be performed in the back-substitution phase.

BalCR on a Hypercube: The Back-substitution Phase

Edge processors in the linear arrays that are embedded in each sub-cube have only one (left or right) neighbor, and hence do only one SEND and one RECEIVE. The required conditionals are omitted for notational simplicity.

$label = p$
for $i = d - 1$ **step** -1 **to** 0
  $j = \text{MOD}(label, 2^i)$
  if $label/2^{i+1}$ odd) **then**
    Exchange with $nbr(i)$
      SEND $G_{j,l}^{(i)}$
      RECEIVE the corresponding group
    Reset value of $label$ to that of $nbr(i)$:
      $label = 2^i * (\gamma^{(P)})^{-1}(id)/2^i + j$
  endif
  if ($label/2^i$ even) **then** $l = 1$; $m = 0$ **else** $l = 0$; $m = 1$ **endif**
  id of left neighbor $= \gamma^{(P)}(label/2^i - 1) * 2^i + j$
  id of right neighbor $= \gamma^{(P)}(label/2^i + 1) * 2^i + j$
  SEND $G_{j,m}^{(i)}$ left and right
  RECEIVE $G_{j,l}^{(i)}$ from left and right
  Perform BACK-SUB in $G_{j,l}^{(i)}$
    (Solve for $label$-th unknown using received rows.)
end for

At the end of the substitution phase, all the data is back to its initial distribution, and the back-substitution part of the substructuring is performed for each system in exactly the same manner as for a single system.

The algorithm as described requires 12 $\log_2 P$ start-ups . It is possible to reduce this to $8 \log_2 P$, since there is some redundancy in the data movement. This involves splitting the arithmetic for CR into two stages – one involving data from the row above and the other involving data from the row below. For the rows involved in exchanges, these two stages are performed on different processors.

The parallel complexity of SS/BalCR for solving M tridiagonal systems, each of order N, on a hypercube with P processors is

$$\begin{aligned} T_{SS/BalCR}(M,N,P;\omega,\sigma,\tau) &= 17\frac{MN}{P}\omega + 8\sigma \log_2 P + 80M(1-\frac{1}{P})\tau \\ &+ 17M(1-\frac{1}{P})\omega. \end{aligned} \tag{7.14}$$

The optimal number of processors P^* satisfies

$$\frac{MN}{P^*} \approx \left(\frac{8\sigma}{17\omega}\right)\frac{1}{\ln 2}.$$

For the iPSC/1, this indicates that the optimal number of rows per processor is under 30.

Balanced cyclic reduction is discussed in [112] and [88], and its performance on the iPSC/1 is discussed in [113].

7.3.3 Transpose with Gaussian elimination

We now describe an alternative approach to solving multiple tridiagonal systems of equations.

The idea is to move M/P entire systems to each processor and solve them by standard Gaussian elimination. The data movement can be implemented as the transposition of a rectangular matrix. In each of d steps, the amount of data in each processor remains fixed and one half of the data is exchanged with one of the nearest neighbors. This is in contrast to balanced cyclic reduction, where the message lengths are halved at each step. This has an important effect on the relative performance of the two methods.

The performance of the methods that use the transpose depends on the t_{copy} parameter which measures the time for the shuffle operation in the transpose. In a straightforward FORTRAN implementation of the transpose on the iPSC/1, the shuffle time can account for a large fraction of the cost, depending on the problem size and the number of processors. This can be reduced by using the Radix-4 transpose, or the buffering strategy of [81], or by going to an assembly language version of the shuffle (or some combination of the three).

At the end of the transpose, we have M/P entire tridiagonal systems on each processor that are solved by standard Gaussian elimination. This is followed by what may be termed a "back-transpose", to move the solution back to the initial data distribution.

SS/TGET: When these steps are performed on the reduced system produced by substructuring, the resulting method is called SS/TGET (for Substructuring/Transpose + Gaussian elimination). Note that the matrix entries of the *reduced* system are involved in the first transpose but only the solution is moved in the second transpose.

The cost of SS/TGET is

$$T_{SS/TGET}(M,N,P;\omega,\sigma,\tau) \approx 17\frac{MN}{P}\omega + 2T_{comm}(16M) + 2T_{comm}(4M) \qquad (7.15)$$
$$+(\log_2 P - 1)\left[2T_{comm}(16\frac{M}{2}) + 16Mt_{copy} + 2T_{comm}(4\frac{M}{2}) + 4Mt_{copy}\right] + 9M\omega.$$

TGET: When the transpose is used without substructuring, the method is called TGET. This method is particularly efficient for small cube dimensions, because it does not incur the substructuring penalty. For TGET, we assume the same distribution of the right-hand sides and the solution vectors across the hypercube as we did for SS/BalCr and SS/TGET. However, since TGET never requires a matrix entry in more than one processor, we can assume that all matrix entries are stored where they are needed, and do not need to be moved. (Such an assumption would not be justified for SS/BalCR or SS/TGET, because these algorithms need the entries of each reduced system on one or more processors different from the processor on which substructuring was performed. Furthermore, the reduced system depends on the dimension of the cube.) The cost of solving M tridiagonal systems of order N each on a P processor hypercube by TGET using the BABE variant of Gaussian elimination is

$$T_{TGET}(M,N,P;\omega,\sigma,\tau) = \text{Transpose time} + \text{G.E. time} \qquad (7.16)$$
$$= 2(\log_2 P - 1)\left[2T_{comm}(2\frac{MN}{P}) + 4\frac{MN}{P}t_{copy}\right] + 9\frac{MN}{P}\omega + 2T_{comm}(4M/P)),$$

Note that there are only $\log_2 P - 1$ steps in each of the two transposes because of the BABE variant. This saving is partially offset by the communication required for the Gaussian elimination phase which is given by the last term.

7.3.4 A comparison of hypercube methods for multiple tridiagonal systems

In this section, we compare SS/BalCR, SS/TGET and TGET on the basis of their performance in experiments conducted on the iPSC/1, and also on the the basis of model times, in parameter regimes different from the iPSC/1. We recall some properties of these methods that account for the differences in performance.

An advantage of SS/BalCR is that the amount of data involved in the exchanges is *halved* at each step (in the reduction phase) whereas it remains constant in the transpose.

TGET has the following advantages.

- It has a smaller number of start-ups per step.
- It enables one to use standard Gaussian elimination, which requires roughly half as much arithmetic as substructuring or cyclic reduction.
- It does not require moving the matrix entries.

These properties put TGET at an advantage for moderate sized cubes and problems. However, they reduce the cost by a constant factor. The total complexities of the data movement are $O(M)$ for SS/BalCR and $O(M \log_2 P)$ for SS/TGET.

Figure 7.8 shows the time needed to solve 128 real independent tridiagonal systems, each of order 128 on the iPSC/1, on cubes of dimension up to 6, using SS/BalCR, SS/TGET and TGET. The times are average times and include the effect of delays caused by message failures. The substructuring penalty is apparent for SS/BalCR and SS/TGET. SS/TGET approaches TGET as P approaches N, while SS/BalCR performs more start-ups per step than SS/TGET.

Out of the methods we have investigated, TGET is the fastest and the simplest to program. However, we will now demonstrate that there exist regions in the parameter space where SS/BalCR is superior to TGET. This further comparison will be based on model times.

Figure 7.9 compares the three methods using model times under the following assumptions: $N = M$, $P \leq N$, $\sigma = 1500$, and $\tau = 1$. The model also accounts for the "internal start-ups" associated with each packet of a message. The model times are based on using the BABE algorithm in SS/TGET and TGET. Each (P, N) pair is represented by a box, which is shaded to indicate which method is fastest for that pair. The four boxes represent different choices of the parameters ω and t_{copy}. SS/BalCR is the fastest for sufficiently large P and N. Reducing the copying time shifts the crossover boundary in favor of TGET (or SS/TGET). Reducing the arithmetic cost favors methods based on substructuring. Note that the top right box corresponds approximately to the parameters for the iPSC/1.

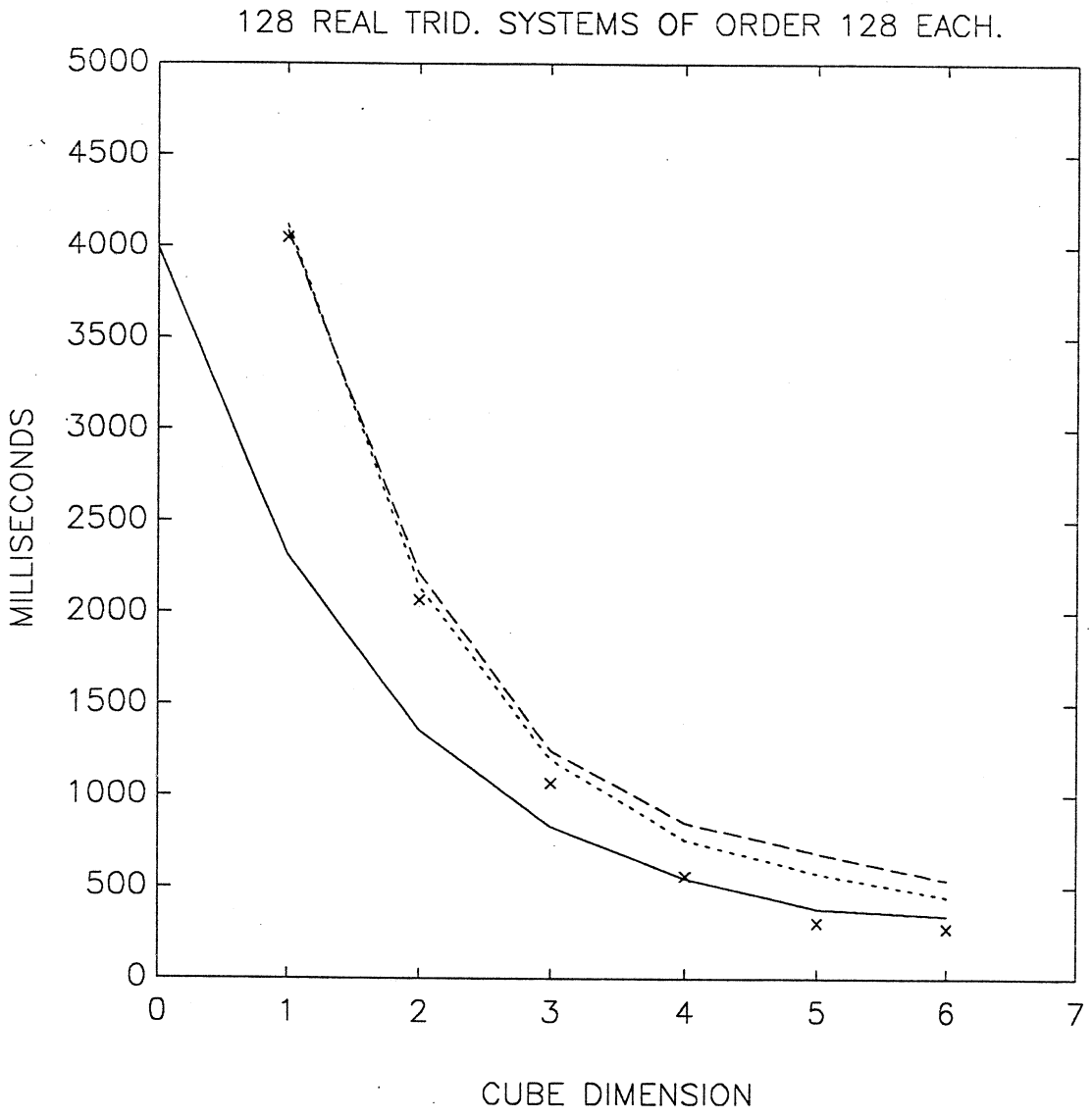


Figure 7.8: *Times on the Intel iPSC/1 to solve* 128 *tridiagonal systems of order* 128. *Solid line: TGET, dotted line: SS/TGET, dashed line: SS/BalCR. The time spent by SS/BalCR and SS/TGET in substructuring is marked by 'X's.*

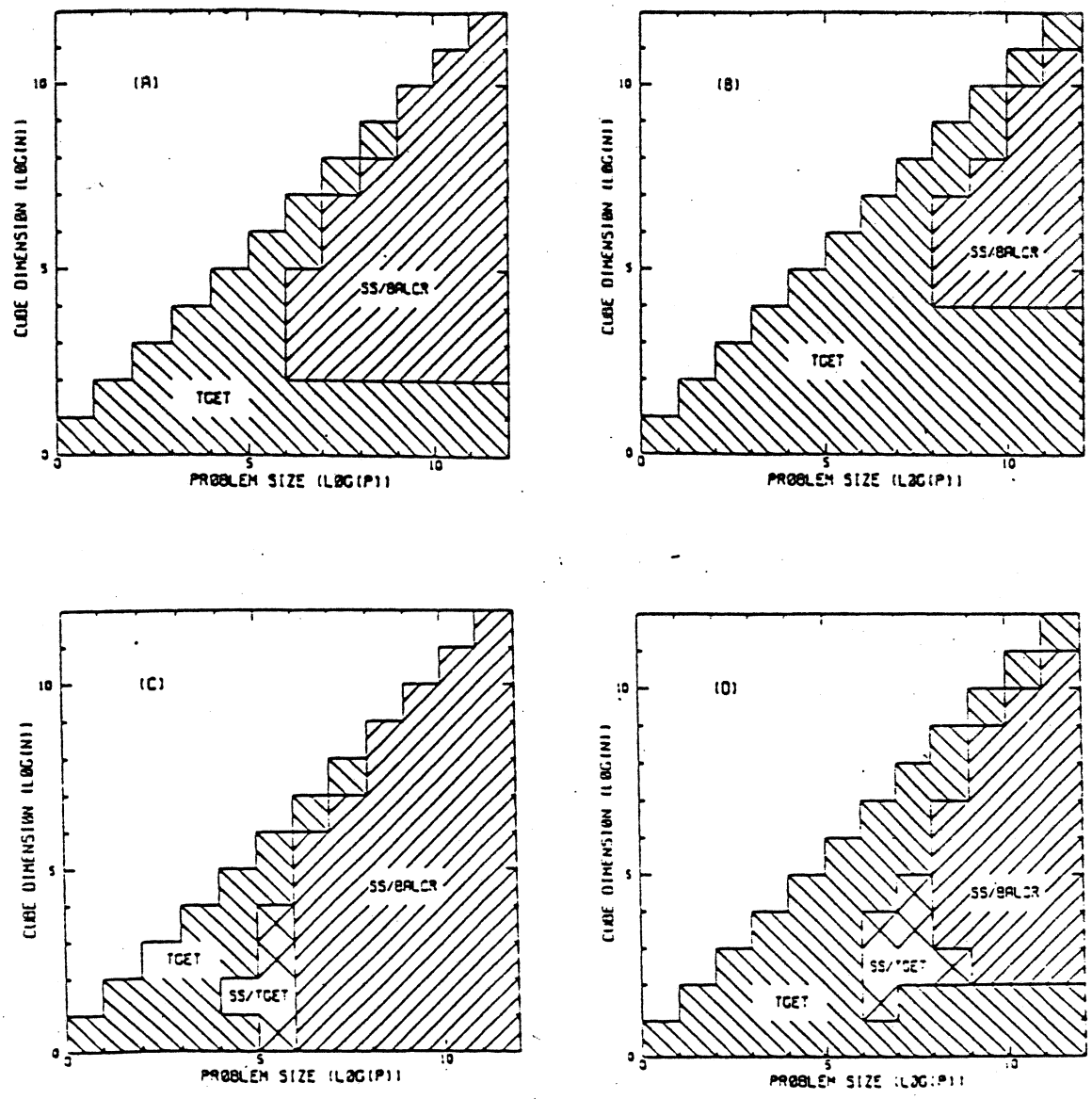


Figure 7.9: *Comparison of TGET, SS/TGET and SS/BalCR based on model times for solving N tridiagonal systems of order N, on a P processor hypercube. The cube parameters were $\sigma = 1500$, $\tilde{\sigma} = 1000$, $\tau = 1$. The (ω, t_{copy}) pairs used in each box are given in parentheses.*

Chapter 8

ADI Methods on a Hypercube Multiprocessor

In this chapter, we consider algorithms for Alternating Direction Implicit methods for the hypercube. Split-step ADI schemes for the Schrödinger equation in two space dimensions were discussed in Chapter 6. On the basis of the results of the previous chapter, we will focus on methods that use the transpose. We state the basic algorithm, CUBE-ADI, and discuss some of its properties. We give an expression for the cost of one step of the CUBE-ADI method based on our model. We present results of experiments on the iPSC/1 and the iPSC/2 and their relation to the predicted model times. We describe some communication-reducing variants of the basic algorithm. We study the sensitivity of the CUBE-ADI method to variations in the aspect ratio. In conclusion, we discuss extensions of our parallel algorithms to hypercubes with higher parallelism (e.g. a few thousand processors).

We recall the structure of the ADI scheme (see Chapter 6).

$$\left[I + \frac{ir}{2}\delta_x^2\right] U^* = \left[I - \frac{ir}{2}\delta_y^2\right] U^n, \tag{8.1}$$

$$\left[I + \frac{ir}{2}\delta_y^2\right] U^{n+1} = \left[I - \frac{ir}{2}\delta_x^2\right] U^*, \tag{8.2}$$

The operations required to implement (8.1), (8.2) can be stated as follows.

Algorithm SERIAL-ADI

Step 1a. Form matrix-vector products in the y-direction.
(Right-hand side of (8.1), N_x columns, each of length N_y.)
Step 1b. Solve tridiagonal systems in x-direction (8.1).
Step 1c. Transpose data so that it is ordered by (grid) columns

Step 2a. Form matrix-vector products in the x-direction.
(Right-hand side of (8.2), N_y columns, each of length N_x).
Step 2b. Solve tridiagonal systems in y-direction (8.2).
Step 2c. Transpose data so that it is ordered by (grid) rows

There are certain improvements one can make to SERIAL-ADI (see, for example, [66]), which we have omitted for simplicity. These improvements are applicable to the parallel implementation as well.

8.1 The CUBE-ADI Method

We now give a detailed description of our basic method for the ADI method on a rectangular domain. The starting point is to embed a two-dimensional processor mesh in the hypercube, using the binary-reflected Gray code, as proposed in [112] (see Figure 7.1). However, our CUBE-ADI method does not use any of the methods discussed in [112]. Performance results for the iPSC/1 for a similar hypercube ADI method that used one-dimensional domain decomposition and the BABE variant in one direction are given in [113].

Algorithm CUBE-ADI

Steps 1.1 to 1.5 comprise the first half step of the ADI method. Steps 2.1 to 2.5 comprise the second half step of the ADI method. This algorithm uses the transpose algorithm described in Chapter 7, and the BABE variant of G.E. that is described in Appendix A (in particular the 2×2 system that arises in the BABE variant).

Step 1.1 (MVP) Matrix-vector product along columns.
EXCHANGE data with North and South neighbors.
Perform matrix-vector product for local grid block.

Step 1.2 Global transpose along processor rows.
for $i = 0$ **step** 1 **to** $\log_2 P_x - 2$
(1) EXCHANGE data with i-th row-neighbor.
(2) Shuffle unsent local data and received data.
end for

Step 1.3 (TRID) Solve tridiagonal systems along rows by the BABE variant of Gaussian elimination. Each system is split across two processors that are neighbors in the hypercube.
for $j = 1$ **step** 1 **to** $2\frac{N_y}{P}$
Factor and forward solve in the local half of the j-th system.
end for
EXCHANGE data corresponding to the 2×2 system in the center for each system with i-th row-neighbor ($i = \log_2 P_x - 1$.)
Each 2×2 system is solved on both processors that share the full system.
for $j = 1$ **step** 1 **to** $2\frac{N_y}{P}$
Back-solve for the local half of the j-th system.
end for

Step 1.4 Global transpose along processor rows. (Step 1.2 "in reverse".)

for $i = \log_2 P_x - 2$ **step** -1 **to** 0

(1) "Unshuffle". Prepare message buffer.

(2) EXCHANGE data with i-th row-neighbor.

end for

Step 1.5 Local transpose.

Reorder the local data so that it is stored by columns.

Step 2.1 (MVP) Matrix-vector product along rows.

EXCHANGE data with East and West neighbors.

Perform matrix-vector product for local grid block.

Step 2.2 Global transpose along processor columns.

for $i = 0$ **step** 1 **to** $\log_2 P_y - 2$

(1) EXCHANGE data with i-th column-neighbor.

(2) Shuffle unsent local data and received data.

endfor

Step 2.3 (TRID) Solve tridiagonal systems along columns by the BABE variant of Gaussian elimination. Each system is split across two processors that are neighbors in the hypercube.

for $j = 1$ **step** 1 **to** $2\frac{N_x}{P}$

Factor and forward solve in the local half of the j-th system.

end for

EXCHANGE data corresponding to the 2×2 system in the center for each system with i-th column-neighbor ($i = \log_2 P_y - 1$.)

Each 2×2 system is solved on both processors that share the full system.

for $j = 1$ **step** 1 **to** $2\frac{N_x}{P}$

Back-solve for the local half of the j-th system.

end for

Step 2.4 Global transpose along processor columns. (Step 2.2 "in reverse".)

for $i = \log_2 P_y - 2$ **step** -1 **to** 0

(1) "Unshuffle". Prepare message buffer.

(2) EXCHANGE data with i-th column-neighbor.

end for

Step 2.5 Local transpose.

Reorder the local data so that it is stored by rows.

"Edge" processors have only one exchange in Step 1.1 or 2.1 (or both). If $P_x = 1$, Steps 1.2 and 1.4 are not performed and the standard Gaussian elimination is used to solve the tridiagonal systems instead of Step 1.3. Similarly, if $P_y = 1$, then Steps 2.2 and 2.4 are skipped and Step 2.3 is replaced by standard Gaussian elimination.

The cost of the shuffle in the global transposes (the effective value of t_{copy}) was reduced by unrolling the inner loops in the shuffles. Local data is stored as a one dimensional array. The local transposes (Steps 1.5, 2.5) ensure that the memory accesses have stride one in the tridiagonal solves. The local transposes account for less than three percent of the total time.

We now list a number of features of the CUBE-ADI method.

- The transposes along rows and columns in the processor mesh are efficient because each processor row and column is itself a sub-cube.
- The BABE variant of Gaussian elimination results in saving one step in each of the four transposes.
- All inter-node communication is in the form of exchanges between adjacent processors.
- The matrix elements are not moved between nodes.

The BABE algorithm was used in the ADI implementation in [113] where it led to the most efficient method reported in that paper. However, in [113] the BABE algorithm was used in only *one* direction only. It was pointed out in [88] that the BABE algorithm can be used in *both* the x and y directions, thereby reducing the number of transpose steps from $2\log_2(P) - 2$ to $2\log_2(P) - 4$.

The CUBE-ADI algorithm can be used, with a few, simple modifications, as the basis for parallelizing several methods for the Schrödinger equation other than the SS-PR and SS-ADI-CD methods for which the CUBE-ADI algorithm is directly applicable (the treatment of the potential operator parallelizes trivially in the split-step approach).

1. With very simple modifications, CUBE-ADI can handle penta-diagonal matrices instead of tridiagonal. These modifications would only affect the arithmetic loops and not alter the communication structure. Thus, the SS/ADI/FD(2,8) method for the Schrödinger equation in two space dimensions can be implemented on a hypercube with a speed-up at least as great as that of CUBE-ADI.

2. The communication structure of the CUBE-ADI algorithm can be used as the basis of a hypercube implementation of a two-dimensional FFT. One would replace Step 1.1 by a number of local FFTs (in the x direction), merge Steps 1.2 and 1.4, omitting Step 1.3. At this point, we would have to perform a number of local FFTs (in the y direction), completing the forward transform. The form of the inverse transform would be very similar, except that the steps would be performed in the reverse order (using inverse FFTs). This could serve as the basis of parallelizing the split-step Fourier method, the spectral leap-frog and the spectral Adams-Bashforth methods for the Schrödinger equation in two space dimensions.

3. A different modification of CUBE-ADI leads naturally to a hypercube implementation of a "conventional" fast Poisson solver, based on FFTs and tridiagonal solves ([59]). The main changes would be to omit the matrix-vector products in Steps 1.1 and 2.1 and replace the tridiagonal solves in the Step 1.3 by the appropriate (local) FFTs. As in the previous remark, Steps 1.2 and 2.2 would be merged with Steps 1.4 and 2.4, respectively. We have discussed in Chapter 6 how fast Poisson solver techniques can be used in conjunction with the split-step approach to efficiently solve the implicit equations in schemes such as SS/CN-2d, SS/CD-2d and SS/FD(2,8)-2d.

8.2 The Parallel Cost of CUBE-ADI

We state the cost of each step in the CUBE-ADI algorithm and sum it to get the total cost. In the results we will report, we have assumed that the matrices occurring have ones on the off-diagonals and complex entries on the main diagonal. Although complex arithmetic is used in the program, the constants C_{MVP} and C_{GE} are the number of *real* floating point operations per unknown required in the matrix-vector products and the tridiagonal solves, respectively. With the assumptions we are making about the structure of the matrices, these constants are 10 and 28, respectively. In the following cost estimates, we have omitted some lower order terms.

Step 1.1:	$C_{MVP}\frac{N}{P}\omega + 4T_{comm}\left(\frac{8N_x}{P_x}\right)$
Step 1.2:	$(\log_2 P_x - 1)\left[2T_{comm}\left(\frac{4N}{P}\right) + 8\frac{N}{P}t_{copy}\right]$
Step 1.3:	$C_{GE}\frac{N}{P}\omega + 2T_{comm}\left(\frac{24N_y}{P}\right)$
Step 1.4:	same as Step 1.2
Step 1.5:	$8\frac{N}{P}t_{copy}$
Step 2.1:	$C_{MVP}\frac{N}{P}\omega + 4T_{comm}\left(\frac{8N_y}{P_y}\right)$
Step 2.2:	$(\log_2 P_y - 1)\left[2T_{comm}\left(\frac{4N}{P}\right) + 8\frac{N}{P}t_{copy}\right]$
Step 2.3:	$C_{GE}\frac{N}{P}\omega + 2T_{comm}\left(\frac{24N_x}{P}\right)$
Step 2.4:	same as Step 2.2
Step 2.5:	same as Step 1.5

We first state the cost of SERIAL-ADI. This is simply the cost of Steps 1.1, 1.3, 1.5, 2.1, 2.3 and 2.5, without the T_{comm} terms:

$$T_{SERIAL-ADI}(N;\omega,t_{copy}) = 2\left[(C_{MVP}+C_{GE})\omega + 8t_{copy}\right]N \tag{8.3}$$

We can now state the parallel cost of one full step of CUBE-ADI as follows.

$$\begin{aligned} T_{CUBE-ADI}(N,P,N_x,P_x;\omega,\sigma,\tau,t_{copy}) &= \frac{1}{P}\cdot T_{SERIAL-ADI}(N;\omega,t_{copy}) \\ &+4T_{comm}\left(8N_x/P_x\right) + 4T_{comm}\left(8N_y/P_y\right) \\ &+2T_{comm}\left(24N_x/P\right) + 2T_{comm}\left(24N_y/P\right) \\ &+4\left(\log_2 P - 2\right)T_{comm}\left(4N/P\right) + 16\left(\log_2 P - 2\right)\frac{N}{P}t_{copy} \end{aligned} \tag{8.4}$$

The last three rows of (8.4) represent the overhead of parallelizing the algorithm. The cost of communication in the matrix-vector product and tridiagonal solve phases is given by the second

and third rows, respectively. The fourth row gives the cost of the global transposes (to simplify the expression, we have assumed that $P_x, P_y > 2$).

Note that the dependence on N_x, N_y, P_x, P_y appears only in the communication costs in the matrix-vector products and tridiagonal solves, and not in the dominant terms, namely the arithmetic and global communication terms. We will examine the effect of varying these parameters, for fixed N and P.

8.3 Performance of CUBE-ADI

We now present performance results for the CUBE-ADI algorithm for a set of experiments that were conducted on both the iPSC/1 and the iPSC/2. The results for both hypercubes are summarized in Table 8.3.

8.3.1 Results for the iPSC/1

Table 8.1 gives a breakdown of times on the iPSC/1 for one step of the CUBE-ADI method. The times are in milliseconds. The times for a number of different grids sizes and cube dimensions are presented. The processor meshes were chosen to be square (or nearly square). The times for the matrix-vector products, the tridiagonal solves and the local transposes have a very small variance. The global transposes have the largest variability of all the phases. The first value in the global transpose column in Table 8.1 is an estimate for the communication time based on the model (7.2). The second value in that column is the total time spent in the shuffles over all global transposes. This value is based on a separate timing of the appropriate code segments. The T_{pred} column has the predicted times for a full ADI step and is the sum of the times in the same row for the individual phases. The T_{avg} column has the measured times, averaged over 10 ADI steps. The last (Δ) column shows the discrepancy between the predicted and the model times.

The rows corresponding to $P = 1$ are the single processor times. For the larger grids, we have included an extrapolation to a 128 processor cube. The extrapolated values for the global transpose phase were chosen to be about the same as the measured times for the 64 processor case. This represents a slight over-estimation, since the model predicts a *decrease* in the transpose time on a larger cube, provided that the effect of the start-up cost is not appreciable, which is the case for the larger grids.

The table shows that the times for the arithmetic phases are approximately halved as the number of processors is doubled. The global transposes account for the bulk of the parallel overhead and the percentage of the (model) time spent in them grows slowly with P. The times spent in the shuffles are comparable to the model time for the communication in the transposes. It should be noted that the *percentage* of the total time spent in the global transpose is smallest when N is large and P small.

On the iPSC/1, a communication activity can fail due to certain bottlenecks and there is a long wait before a re-try (5 to 10 milliseconds). The most plausible explanation is that when

more than one of the 82586 (LAN) chips on a node is trying to access the system buffer part of the memory, they are competing for the memory bus, which is not able to service more than one 82586 chip at a time. For the user, the effect is a large variability in the times for a given program with the same input parameters. The delay due to a collision on any processor will propagate to all other processors due to the tightly coupled nature of our algorithm (unless there are independent, simultaneous collisions, which will be "overlapped"). In this context, we point out that the *minimum* times over many iterations, which are not displayed, are fairly repeatable and are closer to the predicted times than the average times. This phenomenon was discussed in the context of the CR(ℓ) times in Figure 7.5, in the previous chapter.

In Figure 8.1, The efficiency of the CUBE-ADI algorithm on the iPSC/1 is plotted as a function of the hypercube dimension for various problem sizes.

Table 8.1:

Breakdown of the time (in milliseconds) needed for one full step of CUBE-ADI on the iPSC/1. MVP: matrix-vector products, TRID: tridiagonal solves. The global transpose times are broken down in the form [T1] + T2, where T1 is the communication time based on the model and T2 is the shuffle time in the transposes obtained from a separate timing of the appropriate code segments. T_{pred} is the predicted time, based on the measured times for the MVP, TRID, Local transpose and T2 together with the model *times for T1. T_{avg} is the measured time, averaged over 10 ADI steps. Δ is the discrepancy between T_{avg} and T_{pred}. The times in parentheses are extrapolated values.*

N_x (= N_y)	P ($P_x \times P_y$)	MVP	TRID	Global Tr'pose	Local Tr'pose	T_{pred}	T_{avg}	Δ
64	1					(7,825)		
	2 (1 x 2)	960	2,855	0	110	3,925	3,926	1
	4 (2 x 2)	490	1,430	0	55	1,975	1,978	3
	8 (2 x 4)	250	720	[10] + 27	30	1,037	1,061	24
	16 (4 x 4)	130	360	[18] + 28	20	556	612	56
	32 (4 x 8)	70	185	[24] + 24	10	313	387	74
	64 (8 x 8)	40	100	[28] + 20	5	193	312	119
128	1					(31,300)		
	2 (1 x 2)	3,850	11,360	0	430	15,640	15,650	10
	4 (2 x 2)	1,930	5,680	0	210	7,820	7,830	10
	8 (2 x 4)	970	2,830	[80] + 90	110	4,080	4,070	-10
	16 (4 x 4)	500	1,420	[80] + 91	60	2,151	2,160	9
	32 (4 x 8)	250	720	[60] + 71	30	1,131	1,180	49
	64 (8 x 8)	130	360	[40] + 51	15	596	706	110
	128	(65)	(180)	(100)	(10)	(355)		
256	1					(125,200)		
	16 (4 x 4)	1,930	5,660	[320]+344	210	8,364	8,360	-4
	32 (4 x 8)	970	2,850	[240]+260	110	4,430	4,370	-60
	64 (8 x 8)	490	1,450	[160]+176	60	2,336	2,410	74
	128	(250)	(750)	(300)	(30)	(1,330)		
512	1					(500,800)		
	64 (8 x 8)	1,940	5,760	[640]+673	240	9,253	9,343	90
	128	(1000)	(2,880)	(1400)	(120)	(5,400)		

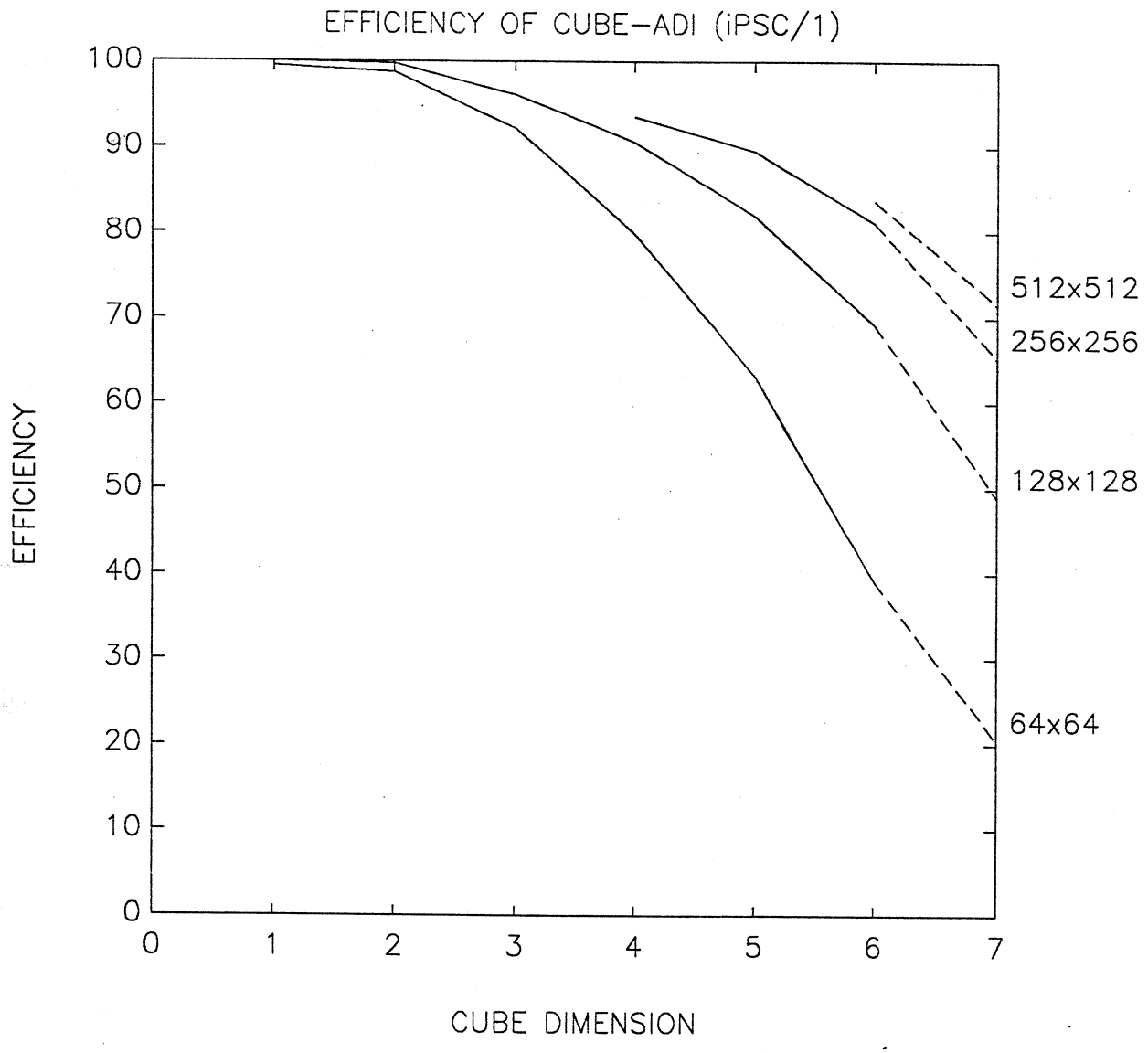


Figure 8.1: *Efficiency vs. cube dimension for the CUBE-ADI algorithm on the iPSC/1. The efficiencies are base on the timings displayed in Table 8.1. The grid sizes are shown on the right. The dashed lines denote extrapolated values.*

8.3.2 Results for the iPSC/2

In this section, we discuss the performance of the CUBE-ADI algorithm on the iPSC/2. The experiments performed were essentially the same as those for the iPSC/1 that were discussed above. The iPSC/1 code was ported to the iPSC/2 with a few straightforward changes. In addition to the improved performance on the iPSC/2, the variability in the timings is substantially lower than on the iPSC/1.

Table 8.2 gives a breakdown of the times (in milliseconds) on the iPSC/2 for one step of the CUBE-ADI algorithm. The form of Table 8.2 is similar to that of Table 8.1, except that the times for the global transpose in Table 8.2 are the differences between the average total times and the sum of the times for the MVP, TRID and local transpose phases. For the iPSC/2, the difference between the predicted and average times was not significant enough to warrant a discussion of the discrepancy between the two.

In particular, the iPSC/2 achieves over 22 Megaflops on a 256×256 grid for CUBE-ADI with 64 processors. The estimated performance for a 1024×1024 grid on a 128 processor iPSC/2 (which we did not have access to) is over 40 Megaflops.

The efficiency of CUBE-ADI on the iPSC/2 is plotted in Figure 8.2 as a function of the hypercube dimension.

Table 8.3 contains a summary of the performance of CUBE-ADI on the iPSC/1 and the iPSC/2.

Table 8.2:

Breakdown of the time (in milliseconds) needed for one full step of CUBE-ADI on the iPSC/2. The column headings have the same meaning as in Table 8.1, except that the global transpose times were obtained from the other times in the same row. The times in parentheses are extrapolated values.

N_x ($= N_y$)	P ($P_x \times P_y$)	MVP	TRID	Global Tr'pose	Local Tr'pose	T_{avg}
64	1	123	462	0	44	634
	2 (1 x 2)	62	240	0	22	327
	4 (2 x 2)	35	124	0	11	171
	8 (2 x 4)	22	64	11	5	100
	16 (4 x 4)	14	33	15	3	62
	32 (4 x 8)	6	16	17	1	40
	64 (8 x 8)	4	10	14	2	29
128	1					(2,536)
	2 (1 x 2)	245	941	0	87	1,274
	4 (2 x 2)	125	480	0	43	648
	8 (2 x 4)	70	245	31	22	368
	16 (4 x 4)	33	124	31	12	200
	32 (4 x 8)	18	65	27	6	116
	64 (8 x 8)	10	31	24	3	68
	128	(6)	(16)	(21)	(2)	(45)
256	1					(10,144)
	4 (2 x 2)	503	1,909	0	181	2,594
	8 (2 x 4)	291	963	108	88	1,450
	16 (4 x 4)	136	478	112	44	770
	32 (4 x 8)	64	243	88	22	417
	64 (8 x 8)	33	121	63	11	230
	128	(18)	(62)	(50)	(5)	(135)
512	1					(40,576)
	16 (4 x 4)	549	1,906	463	180	3,098
	32 (4 x 8)	263	959	329	88	1,634
	64 (8 x 8)	129	476	225	44	874
	128	(68)	(242)	(160)	(22)	(492)
1024	1					(162,30)
	64 (8 x 8)	555	1,927	873	181	3,530
	128	(285)	(970)	(512)	(91)	(1858)

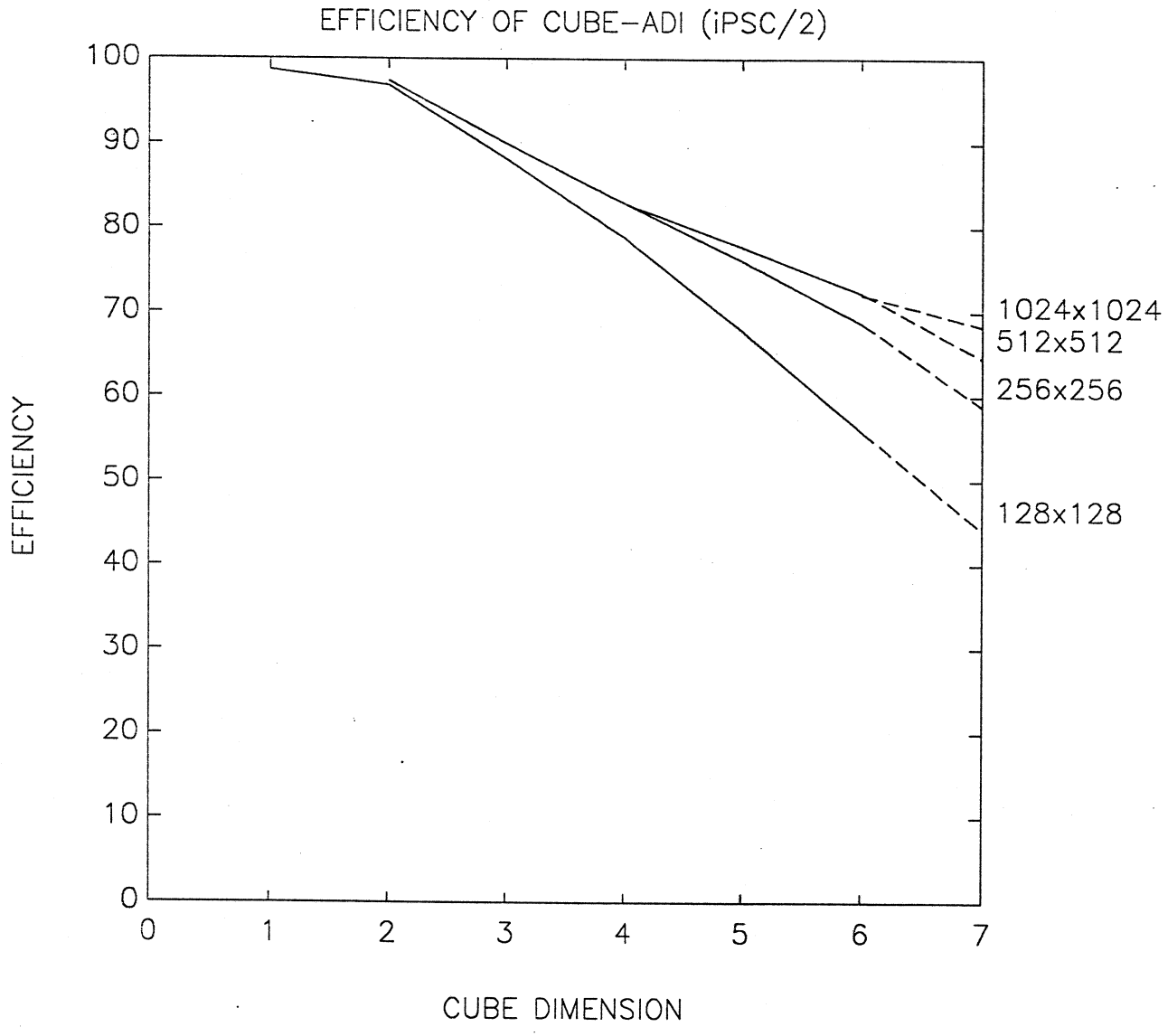


Figure 8.2: *Efficiency vs. cube dimension for the CUBE-ADI algorithm on the iPSC/2. The efficiencies are base on the timings displayed in Table 8.2. The grid sizes are shown on the right. The dashed lines denote extrapolated values.*

Table 8.3:

A summary of the performance of CUBE-ADI on the iPSC/1 and the iPSC/2. The times on a single processor and on 64 *processors are given in milliseconds. The speed-up values given in* [...] *are the single processor times divided by the* 64 *processor times. RATIO is the iPSC/1 time divided by the iPSC/2 time. Estimated Megaflop rates for the* 64 *processor times are also shown.*

	iPSC/1	iPSC/2	RATIO
128 × 128 grid:			
1 processor	31,300	2,536	12.34
64 processors	706	68	10.38
Speed-up	[44.3]	[37.3]	
Megaflops	1.72	17.9	
256 × 256 grid:			
1 processor	125,200	10,144	12.34
64 processors	2,410	230	10.47
Speed-up	[52.0]	[44.1]	
Megaflops	2.02	21.17	
512 × 512 grid:			
1 processor	500,800	40,576	12.34
64 processors	9,343	874	10.69
Speed-up	[53.6]	[46.4]	
Megaflops	2.08	22.27	

Table 8.4:

An example of the communication pattern in the CUBE-ADI algorithm. Each communication activity is an exchange of data with the indicated processor. This table shows the exchange partners at each step for processors 0, 5, 16 and 21 on a 128×128 *grid with 64 processors and an* 8×8 *processor mesh. The size of the messages in bytes is given. The first column gives the step number in the CUBE-ADI algorithm.*

Step	Message length	P_0	P_5	P_{16}	P_{21}
1.1	128	P_8	P_{13}	P_{24}, P_{48}	P_{29}, P_{53}
1.2	1024	P_1	P_4	P_{17}	P_{20}
		P_2	P_7	P_{18}	P_{23}
1.3	384	P_4	P_1	P_{20}	P_{17}
1.4	1024	P_2	P_7	P_{18}	P_{23}
		P_1	P_4	P_{17}	P_{20}
1.5					
2.1	128	P_1	P_7, P_4	P_{17}	P_{23}, P_{20}
2.2	1024	P_8	P_{13}	P_{24}	P_{29}
		P_{16}	P_{21}	P_0	P_5
2.3	384	P_{32}	P_{37}	P_{48}	P_{53}
2.4	1024	P_{16}	P_{21}	P_0	P_5
		P_8	P_{13}	P_{24}	P_{29}
2.5					

8.4 Communication-reducing Variants of CUBE-ADI

The CUBE-ADI algorithm has a moderately complex pattern of communication activity. This is displayed in Table 8.4.

Each processor exchanges data with each of its $\log_2 P$ neighbors in the course of a full ADI step. The manner in which this is done results in information flowing from each processor to every other. Table 8.4 indicates some redundancy in the communication, for example, the last exchange partner in Step 1.4 is an exchange partner in Step 2.1. One can consider variants of the CUBE-ADI algorithm that reduce the communication requirements by eliminating the "redundant" exchanges.

The use of the BABE variant of Gaussian elimination is a communication-reducing strategy that is already implemented in CUBE-ADI.

The communication in the matrix-vector product phases can be reduced either by going from an ADI scheme to a locally one-dimensional (LOD) scheme or by modifying the CUBE-ADI algorithm so that the matrix-vector product from a given half step is moved to the tridiagonal solves of the previous half step.

The alternating sweep ordering variant of ADI is primarily a communication-reducing device. In CUBE-ADI/ASO, the number of global transpose steps is halved (when we average over two full ADI steps). Thus, instead of the $2\log_2(P) - 4$ transpose steps required by CUBE-ADI, each full step of CUBE-ADI/ASO would perform $\log_2(P) - 2$ transpose steps. For large P, this saving will have a greater impact than each of the two strategies discussed above. A combination of all three devices will clearly be the most effective approach.

8.5 Effect of Aspect Ratio of Processor Mesh

In Figure 8.3, we have plotted the time taken by the CUBE-ADI algorithm on a 5-cube as a function of P_x to investigate the effect of variations in the aspect ratio of the processor mesh. The times for three different computational grids with the same total number of points are plotted (128×128, 256×64, 512×32). The parts of the expression for the model time (8.4) that depend on the aspect ratios of the grid and processor mesh (as opposed to the total number of grid points and processors) is a lower order term. The dependence is essentially of the form

$$T_{comm}(N_x/P_x) + T_{comm}(N_x/P_x).$$

The standard area/perimeter argument (applied to rectangles) shows that the minimum, for a fixed N/p, occurs when $N_x/P_x = N_y/P_y$. It should be kept in mind that the cases $P_x = 1$ and $P_x = P$ are penalized by the fact that the BABE algorithm can only be employed in one direction only and therefore the total number of transpose steps is $2(\log_2 P - 1)$ rather than $2(\log_2 P - 2)$. Also, in the more skewed cases (for example, a 1×32 processor mesh for a 512×32 grid), the exchanges with the NORTH and SOUTH neighbors involve long messages. However, if one ignores these edge effects and picks $P_x \approx \sqrt{P}$ ($P_x = 4$ or 8 for a 5-cube), the time is within 2% of the time with the "optimal" P_x. Figure 8.3 shows that the minimum is very flat, reflecting the fact that we are minimizing a lower order term.

8.6 Extensions to Large Hypercubes

The algorithms that we have considered for the Intel hypercubes assume that we have a moderate number of processors. In fact, the algorithms we have described assume that $P \leq \min\{N_x, N_y\}$. We will now describe two different approaches to extending our methods to the case when we have more the $\sqrt{N}$ processors. We will use the following values for illustrative purposes in our description: $N_x = N_y = 128$, $P_x = P_y = 32$, i.e. $N = 16384$ and $P = 1024$. We assume the usual two-dimensional domain decomposition with a $P_x \times P_y$ processor mesh. We consider in detail the solution of tridiagonal systems along rows and outline two different methods that use the "building blocks" that have been developed earlier.

T/SS/CR(ℓ): Our first algorithm combines a few steps of the transpose with the SS/CR(ℓ) method. Each processor row has $N_y/P_y = 128/32 = 4$ systems, each spread across the $P_x = 32$

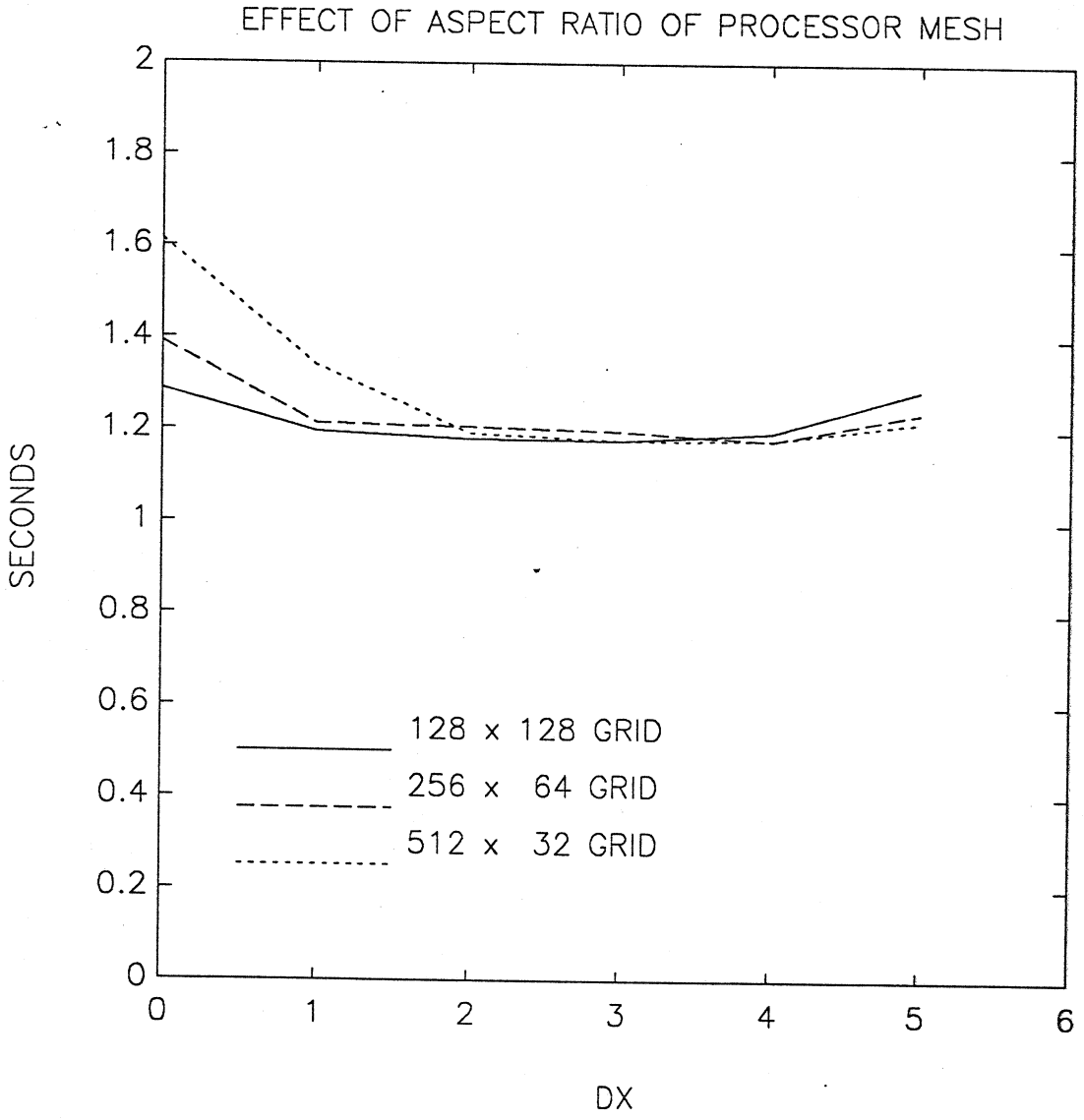


Figure 8.3: *Sensitivity of the CUBE-ADI algorithm to variations in the aspect ratio of the processor mesh. DX is the number of dimensions in the x direction, i.e.* $P_x = 2^{DX}$.

processors. In the first stage, we perform $\log_2(N_y/P_y)$ (= 2) steps of the transpose. (We cannot perform the full transpose because we have "too many" processors.) We now have the 4 systems separated, each spread across a separate sub-cube consisting of P/N_y (= 8) processors. Since the order of each system is $N_x = 128$, we have $N/P = 16$ unknowns per processor. We are now in a position to apply any algorithm for solving a single tridiagonal system. If we use the SS/CR(ℓ) method, the whole method can be called the T/SS/CR(ℓ) method.

SS/BalCR/CR(ℓ): Our second algorithm uses a few steps of balanced cyclic reduction rather than the transpose and employs substructuring and CR(ℓ) somewhat differently from the first algorithm. In this method, we *first* perform substructuring on each of the $N_y/P_y = 4$ systems in each processor row. We now have 4 reduced systems, each of order $P_x = 32$, spread across a sub-cube consisting of P_x processors (the processor row). The second stage is to perform $\log_2(N_y/P_y)$ = 2 steps of balanced cyclic reduction (BalCR) on each sub-cube. (Again, we cannot perform all the steps of BalCR because we have more processors than systems.) At the end of the second stage, each of the reduced systems has been further reduced by the cyclic reduction process to a smaller system of order $P/N_y = 8$ and each of these is spread across a distinct sub-cube having $P/N_y = 8$ processors. In the third and final stage, each of these systems is solved by some method for solving single tridiagonal systems that have one row per processor, such as CR(ℓ). We will refer to this scheme as SS/BalCR/CR(ℓ).

Both these methods do the same amount of work in their substructuring and CR(ℓ) stages and both perform $\log_2(N_y/P_y)$ steps of the transpose and BalCR, respectively. When P is large, the number of transpose/BalCR steps will be small and in this situation, the transpose has an advantage. Hence, we expect T/SS/CR(ℓ) to be the better of the two methods for ADI on hypercubes with a large (but not massive) number of processors. This statement must be qualified by the reminder that the hardware (and software) parameters of any real machine may make it preferable to use something other than CR(ℓ) for solving a single system of order k on a k processor sub-cube, for example, the cyclic elimination method. Note that as P grows, both the number of transpose steps and the amount of substructuring required in T/SS/CR(ℓ) decrease and the importance of the solver for single systems (with one unknown per processor) increases.

ADI methods for massively parallel machines have been studied in [111] and [110].

Appendix A

Sequential Tridiagonal Solvers

In this section, we describe some variants of Gaussian elimination for the solution of tridiagonal systems of linear equations.

The BABE ("burn at both ends") variant of Gaussian elimination is a well-known algorithm for solving tridiagonal systems (see [69], [71], [106]). The BABE algorithm is often more efficient than standard Gaussian elimination on sequential computers[1].

BABE variant of Gaussian elimination for Ax = d

We will state this algorithm for a tridiagonal system whose sub- and superdiagonal entries all have the value one and whose main diagonal is stored in an array b. The reciprocals of the pivots are computed and stored in b. For simplicity, we assume that N, the order of the system, is even.

1. Factor and forward solve.

   $b(1) = 1/b(1)$
   $b(N) = 1/b(N)$
   for $j = 2$ **step** 1 **to** $\frac{N}{2}$
   $k = N - j + 1$
   $b(j) = 1/(b(j) - b(j-1))$
   $d(j) = d(j) - b(j-1) * d(j-1)$
   $b(k) = 1/(b(k) - b(k+1))$
   $d(k) = d(k) - b(k+1) * d(k+1)$
   end for

2. Solve a 2x2 system in the center for $x(\frac{N}{2})$, $x(\frac{N}{2}+1)$.

   $\Delta = (1 - b(\frac{N}{2}) * b(\frac{N}{2}+1))$
   $x(\frac{N}{2}) = (d(\frac{N}{2}) * b(\frac{N}{2}+1) - d(\frac{N}{2}+1) * b(\frac{N}{2}) * b(\frac{N}{2}+1))/\Delta$
   $x(\frac{N}{2}+1) = (d(\frac{N}{2}+1) * b(\frac{N}{2}+1) - d(\frac{N}{2}) * b(\frac{N}{2}) * b(\frac{N}{2}+1))/\Delta$

3. Back solve.

   for $j = \frac{N}{2} - 1$ **step** -1 **to** 1
   $k = N - j + 1$
   $x(j) = (d(j) - x(j-1)) * b(j-1)$
   $x(k) = (d(k) - x(k+1)) * b(k+1)$
   end for

[1] In particular, the tridiagonal solver in LINPACK [71] uses the BABE variant of Gaussian elimination

Remark. If A is a constant on the main diagonal and all off-diagonal entries are one, then the pivots computed by the BABE algorithm have a symmetry: $b(j) = b(N+1-j)$ and one need not compute the pivots for the lower half of the matrix.

Remark. Standard Gaussian elimination, when applied to a matrix of this form, with $|\lambda| > 2$, has the property that the pivots converge ([92], [94]). If $r(j)$ is the j-th pivot computed by Gaussian elimination and r^* is the limit of the $r(j)$'s, then

$$\frac{|r(j) - r^*|}{|r^*|} < \epsilon, \quad \text{for} \quad j > K = K(\epsilon, \lambda),$$

where K is defined as follows. Let $z_\pm = \left(\lambda \pm \sqrt{\lambda^2 - 4}\right)/2$ and set $\rho = \max\{|z_+|, |z_-|\}$. Then

$$K(\epsilon, \lambda) = \lceil \log \epsilon^{-1} / \log \rho^2 \rceil. \tag{A.1}$$

If $|\lambda|$ is very close to 2, then K can be large. However, for $\epsilon = 10^{-16}$ and $|\lambda|$ equal to 3, 4, 5, 6, K has the values 19, 13, 11, 10 ([94]). Their analysis is carried out for real λ but their proof goes through for complex λ, which is what is needed in applying this technique to Schrödinger's equation.

Based on these two remarks, we now combine the BABE algorithm with pivot convergence to get a variant of Gaussian elimination that should be efficient for tridiagonal systems that have ones on the off-diagonals and a constant λ on the main diagonal (with $|\lambda| > 2$, so that no pivoting is required).

BABE GE with pivot convergence for Ax = d

A is a tridiagonal matrix, with a constant λ ($|\lambda| > 2$) on the main diagonal and ones on the sub- and superdiagonals. For a given tolerence ϵ, K is defined in (A.1). The reciprocals of the pivots are stored in the array $b(j)$, $j = 1, \ldots, K$, and $\hat{b}$ is the reciprocal of the limit of pivots. We state the algorithm for the case where N is even and $K < N$.

1. Factor and forward solve.

$b(1) = 1/\lambda$
for $j = 2$ **step** 1 **to** K
  $k = N - j + 1$
  $d(j) = d(j) - b(j-1) * d(j-1)$
  $d(k) = d(k) - b(j-1) * d(k+1)$
  $b(j) = 1/(\lambda - b(j-1))$
end for
$\hat{b} = b(K)$
for $j = K+1$ **step** 1 **to** $\frac{N}{2}$
  $k = N - j + 1$
  $d(j) = d(j) - \hat{b} * d(j-1)$
  $d(k) = d(k) - \hat{b} * d(k+1)$

end for

2. Solve a 2x2 system in the center for $x(\frac{N}{2})$, $x(\frac{N}{2}+1)$.

   $x(\frac{N}{2}) = (d(\frac{N}{2}) * \hat{b} - d(\frac{N}{2}+1) * \hat{b}^2)/(1-\hat{b}^2)$

   $x(\frac{N}{2}+1) = (d(\frac{N}{2}+1) * \hat{b} - d(\frac{N}{2}) * \hat{b}^2)/(1-\hat{b}^2)$

3. Back solve.

   for $j = \frac{N}{2} - 1$ **step** -1 **to** $K+1$

   $k = N - j + 1$

   $x(j) = (d(j) - x(j+1)) * \hat{b}$

   $x(k) = (d(k) - x(k-1)) * \hat{b}$

   end for

   for $j = K$ **step** -1 **to** 1

   $k = N - j + 1$

   $x(j) = (d(j) - x(j+1)) * b(j)$

   $x(k) = (d(k) - x(k-1)) * b(j)$

   end for

Bibliography

Chapters 1–2

[1] A. R. Gourlay. Splitting methods for time dependent partial differential equations. In *The State of the Art in Numerical Analysis*, D. Jacobs, ed., Academic Press, London, 1977.

[2] G. G. O'Brien, M. A. Hyman and S. Kaplan. A study of the numerical solution of partial differential equations. *J. Math. and Phys.*, 29:223-251, 1950

[3] E. M. Reingold, J. Nievergelt and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice Hall, New Jersey, 1977.

[4] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-value Problems.* Interscience Publishers, New York, 1967.

[5] Paul N. Swarztrauber. Vectorizing the FFTs. In *Parallel Computations*, G. Rodrigue (Ed.), Academic Press, New York, 1982.

[6] F. D. Tappert. The parabolic approximation method. In *Wave Propagation and Underwater Acoustics*, J. B. Keller and J. S. Papadakis (Eds.), Lecture Notes in Physics, Vol. 70, Springer-Verlag, New York, 1977.

Chapter 3

[7] W. F. Ames. *Numerical Methods for Partial Differential Equations.* Academic Press, New York, 1977.

[8] J. W. Cooley, P. A. W. Lewis and P. D. Welch. The fast Fourier transform: programming considerations in the calculations of the sine, cosine and Laplace transforms. *J. Sound Vib.*, 12:315–337, 1970.

[9] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods: Theory and Applications.* SIAM, Philadelphia, 1977.

[10] R. H. Hardin and F. D. Tappert. Application of the split-step Fourier method to the numerical solution of nonlinear and variable coefficient wave equations. (Abstract) *SIAM Rev.*, 15:423, 1973.

[11] D. Lee, G. Botseas and J. S. Papadakis. Finite difference solution to the parabolic wave equation. *J. Acoust. Soc. Am.*, 70:795–800, 1981.

[12] D. Lee and S. T. McDaniel. *Ocean acoustic propagation by finite difference methods.* Pergamon Press, New York, 1988.

[13] R. J. Leveque and J. Oliger. Numerical methods based on additive splittings for hyperbolic partial differential equations. *Math. Comp.*, 40:469–497, 1983.

[14] S. T. McDaniel. Applications of energy methods to finite-difference solutions of the parabolic wave equation. In *Computational Ocean Acoustics*, M. H. Schultz and D. Lee (Eds.), Pergamon Press, New York, 1985.

[15] W. G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 6:506–517, 1968.

[16] N. N. Yanenko. *The Method of Fractional Steps.* Springer-Verlag, New York, 1971.

Chapter 4

[17] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* National Bureau of Standards, New York, 1972.

[18] R. F. Boisvert. Families of high order discretizations of some elliptic problems. *SIAM J. Sci. and Stat. Comput.*, 2:268–284, 1981.

[19] E. A. Burke. Extended Numerov method for the numerical solution of the Hartree-Fock equations. *J. Math. Phys.*, 21:1366–1369, 1980.

[20] M. Ciment and S. H. Leventhal. Higher order compact implicit schemes for the wave equation. *Math Comp.*, 29:985–994, 1975.

[21] L. Collatz. *Numerical Treatment of Differential Equations.* Springer-Verlag, New York, 1960.

[22] S. H. Crandall. An optimum implicit recurrence formula for the heat conduction equation. *Q. Appl. Math.*, 13:318–320, 1955.

[23] J. Douglas, Jr. The solution of the diffusion equation by a higher order correct difference equation. *J. Math. Phys.*, 35:145–151, 1956.

[24] B. L. Ehle. A-stable methods and Padé approximations to the exponential. *SIAM J. Math. Anal.*, 4:671–680, 1973.

[25] G. Fairweather and A. R. Mitchell. A high accuracy alternating direction method for the wave equation. *J. Inst. Math. Appl.*, 1:309–316, 1965.

[26] R. Guardiola and J. Ros. On the numerical integration of the Schrödinger equation in the finite-difference schemes. *J. Comp. Phys.*, 45:374–389, 1982.

[27] P. Henrici. Poisson's equation in a hypercube: discrete Fourier methods, eigenfunction expansions, Padé approximation to eigenvalues. In *Studies in Numerical Analysis*, G. H. Golub (Ed.), Math. Association of America, 1984.

[28] F. B. Hildebrand. *Introduction to Numerical Analysis.* McGraw-Hill, New York, 1956.

[29] A. Iserles. On the A-acceptability of Padé approximations. *SIAM J. Numer. Anal.*, 10:1002–1007, 1979.

[30] A. Iserles. Order stars, approximations and finite differences III. Finite differences for $u_t = \omega u_{xx}$. *SIAM J. Math. Anal.*, 16:1020–1033, 1985.

[31] M. K. Jain. *Numerical Solution of Differential Equations.* Halsted Press, John Wiley & Sons, New York, 1979.

[32] R. Jeltsch. Stability on the imaginary axis and A-stability of linear multistep methods. *BIT*, 18:170–174, 1978.

[33] R. E. Lynch and J. R. Rice. The HODIE method and its performance for solving elliptic partial differential equations. In *Recent Advances in Numerical Analysis*, C. de Boor and G. H. Golub (Eds.), pages 143–175, Academic Press, New York, 1978.

[34] A. R. Mitchell and G. Fairweather. Improved forms of the alternating direction methods of Douglas, Peaceman and Rachford for solving parabolic and elliptic equations. *Numer. Math.*, 6:285–292, 1964.

[35] R. S. Varga. On higher order stable implicit methods for solving parabolic partial differential equations. *J. Math. Phys.*, 40:220–231, 1961.

[36] G. Wanner, E. Hairer and S. P. Nørsett. Order stars and stability theorems. *BIT*, 18:475–489, 1978.

[37] G. Wanner, E. Hairer and S. P. Nørsett. When I-stability implies A-stability. *BIT*, 18:503–503, 1978.

Chapter 5

[38] A. Askar and A. S. Cakmak. Explicit integration method for the time-dependent Schrödinger equation for collision problems. *J. Chem. Phys.*, 68:2794–2798, 1978.

[39] T. F. Chan. Stability analysis of finite difference schemes for the advection-diffusion equation. *SIAM J. Numer. Anal.*, 21:272–284, 1984.

[40] T. F. Chan and T. Kerkhoven. Fourier methods with extended stability intervals for the Korteweg-de Vries equation. *SIAM J. Numer. Anal.*, 22:441-454, 1985.

[41] T. F. Chan, D. Lee and L. Shen. Stable explicit schemes for equations of the Schrödinger type. *SIAM J. Numer. Anal.*, 23:274–281, 1986.

[42] T. F. Chan and L. Shen. Stability analysis of difference schemes for variable coefficient Schrödinger type equations. *SIAM J. Numer. Anal.*, 24:336–349, 1987.

[43] G. Dahlquist. A special stability problem for linear multistep methods. *BIT*, 3:27–43, 1963.

[44] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, New York, 1971.

[45] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* John Wiley & Sons, New York, 1962.

[46] H.-O. Kreiss and J. Oliger. *Methods for the Approximate Solution of Time Dependent Problems.* GARP Publication Series, 1973.

[47] D. Kosloff and R. Kosloff. A Fourier method solution for the time dependent Schrödinger equation as a tool in molecular dynamics. *J. Comp. Phys.*, 52:35–53, 1983.

[48] D. Lee and J. S. Papadakis. Numerical solutions to the parabolic wave equation: an ordinary differential equation approach. *J. Acoust. Soc. Am.*, 68:1482–1488, 1980.

[49] D. Lee and S. Prieser. Generalized Adams methods for solving underwater wave propagation problems. *Comp. and Maths. with Appls.*, 2:195–202, 1981.

[50] J. J. H. Miller. On the location of zeros of certain classes of polynomials with applications to numerical analysis. *J. Inst. Math. Applics.*, 8:397–406, 1971.

[51] G. Peggion and J. J. O'Brien. An explicit finite-difference scheme for solving the ocean acoustic parabolic wave equation. *Comp. and Maths. with Appls.*, 11:937–942, 1985.

[52] M. F. Reusch, L. Ratzan, N. Pomphrey and W. Park. Diagonal Padé approximations for initial value problems. *SIAM J. Sci. Stat. Comput.*, 9:829–838, 1988.

[53] J. A. C. Weideman and B. M. Herbst. Split-step methods for the solution of the nonlinear Schrödinger equation. *SIAM J. Numer. Anal.*, 23:485-507, 1986.

Chapter 6

[54] T. F. Chan. *Comparison of Numerical Methods for Initial Value Methods.* Ph.D. Thesis, Stanford University, 1978.

[55] J. Gary. On the optimal time step and computational efficiency of difference schemes for PDE. *J. Comp. Phys.*, 16:298–303, 1983.

[56] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods.* John Wiley & Sons, New York, 1966.

[57] L. I. Schiff. *Quantum Mechanics.* McGraw-Hill, New York, 1968.

[58] R. T. Walsh. Optimization and comparison of partial difference methods. *SIAM J. Numer. Anal.*, 10:785–797, 1973.

Chapter 7

[59] B. L. Buzbee, G. H. Golub and C. W. Nielson. On direct methods for solving Poisson's equation. *SIAM J. Numer. Anal.*, 7:627–656, 1970.

[60] T. F. Chan and K. R. Jackson. The use of iterative linear-equation solvers in codes for large systems of stiff IVPs for ODEs. *SIAM J. Sci. and Stat. Comput.*, 7:378–417, 1986.

[61] J. E. Dendy, Jr. An alternating direction method for Schrödinger's equation. *SIAM J. Numer. Anal.*, 14:1028–1032, 1977.

[62] J. Douglas, Jr. Alternating direction methods for three space directions. *Numer. Math.*, 4:41–63, 1962.

[63] J. Douglas, Jr. and J. E. Gunn. A general formulation of alternating direction methods. Part I. Parabolic and hyperbolic problems. *Numer. Math.*, 6:428-453, 1964.

[64] J. Douglas, Jr. and H. H. Rachford, Jr. On the numerical solution of heat conduction problems in two and three space variables. *Trans. Am. Math. Soc.*, 82:421–439, 1956.

[65] C. W. Gear and Y. Saad. Iterative solution of linear equations in ODE codes. *SIAM J. Sci. and Stat. Comput.*, 4:583–601, 1983.

[66] J. D. Lawson and J. Ll. Morris. A note on the efficient implementation of splitting methods in two space variables. *BIT*, 17:492–493, 1977.

[67] D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *J. Soc. Indust. Appl. Math.*, 3:28-41, 1955.

[68] J. S. Perkins and R. N. Baer. An approximation to the three-dimensional parabolic equation for acoustic propagation. *J. Acoust. Soc. Am.*, 72:515-522, 1982.

Chapter 8

[69] I. Babuška. Numerical stability in problems of linear algebra. *SIAM J. Numer. Anal.*, 9:53–77, 1972.

[70] R. F. Boisvert. *Algorithms for special tridiagonal systems.* Center for Comput. and Appl. Math., Nat. Bur. of Stand., 1988.

[71] J. J. Dongarra, C. B. Moler, J. R. Bunch and G. W. Stewart. *LINPACK User's Guide.* SIAM, Philadelphia, 1979.

[72] J. O. Eklund. A fast computer algorithm for matrix transposing. *IEEE Trans. Computers,* C-21(7):801–803, 1972.

[73] D. J. Evans. An algorithm for the solution of certain tridiagonal systems of linear equations. *Computer J.*, 15:356–359, 1972.

[74] G. H. Golub and C. F. Van Loan. *Matrix Computations* The Johns Hopkins University Press, Baltimore, Maryland, 1983.

[75] M. Hatzopoulos. Parallel linear system solvers for tridiagonal systems. In *Parallel Processing Systems*, D. J. Evans (Ed.), Cambrigde University Press, Cambridge, 1982.

[76] M. T. Heath. *Hypercube Multiprocessors 1986.* SIAM, Philadelphia, 1986.

[77] M. T. Heath. *Hypercube Multiprocessors 1987.* SIAM, Philadelphia, 1987.

[78] D. Heller. Some aspects of the cyclic reduction algorithm applied to block tridiagonal linear systems. *SIAM J. Numer. Anal.*, 13:484–496, 1976.

[79] D. Heller. A survey of parallel algorithms in numerical linear algebra. *SIAM Rev.*, 20:740–777, 1978.

[80] D. Heller, D. Stevenson and J. Traub. Accelerated iterative methods for the solution of tridiagonal linear systems on parallel computers. *J. ACM*, 23:636–654, 1976.

[81] C. T. Ho and S. L. Johnsson. Matrix transposition on Boolean n-cube configured ensemble architectures. Department of Computer Science, Yale University, YALU/CSD/RR-494, 1986.

[82] R. Hockney. A fast direct solution of Poisson's equation using Fourier analysis. *JACM*, 12:95-113, 1965.

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

[84] S. L. Johnsson. Odd-even cyclic reduction on ensemble architectures. Department of Computer Science, Yale University, 339, 1984.

[85] S. L. Johnsson. Data permutation and basic linear algebra computations on ensemble architectures. Department of Computer Science, Yale University, 367, 1984.

[86] S. L. Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. and Stat. Comput.*, 8:354–392, 1987.

[87] S. L. Johnsson and C.-T. Ho. Multiple tridiagonal systems, the alternating direction method and boolean configured multiprocessors. Department of Computer Science, Yale University, YALEU/DCS/RR-532, 1987.

[88] S. L. Johnsson and C.-T. Ho. Algorithms for matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9:419–454, 1988.

[89] R. Kapur and J. Browne. Techniques for solving block tridiagonal linear systems on reconfigurable array computers. *SIAM J. Sci. and Stat. Comput.*, 5:701–719, 1984.

[90] J. J. Lambiotte and R. G. Voigt. The solution of tridiagonal linear systems on the CDC STAR-100 computer. *ACM Trans. Math. Softw.*, 1:308–329, 1975.

[91] M. A. Malcolm and J. Palmer. A fast method for solving a class of tridiagonal systems of linear equations. *Comm. ACM.*, 17:14–17, 1974.

[92] O. McBryan and E. van der Velde. Hypercube programs for computational fluid dynamics. In *Hypercube Multiprocessors 1986*, M. T. Heath (Ed.), SIAM, Philadelphia, 1986

[93] S. T. O'Donnell, P. Geiger and M. H. Schultz. Solving the Poisson equation on the FPS-164. Department of Computer Science, Yale University. YALEU/DCS/RR-293, NOV, 1983.

[94] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[95] J. M. Ortega and R. G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Rev.*, 27:149–240, 1985.

[96] D. J. Rose. An algorithm for solving a special class of tridiagonal systems of linear equations. *Comm. ACM*, 12:234–236, 1969.

[97] Y. Saad and M. H. Schultz. Topological properties of hypercubes. Department of Computer Science, Yale University, YALEU/DCS/RR-389, 1985.

[98] A. Sameh and D. Kuck. On stable parallel linear system solvers. *JACM*, 25:81-91, 1978.

[99] U. Schumann. Comments on "A fast computer algorithm for matrix transposing" and application to the solution of Poisson's equation. *IEEE Trans. Computers*, C-22(5):542–543, 1973.

[100] C. L. Seitz. The Cosmic Cube. *Comm. ACM*, 28:22–33, 1985.

[101] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Computers*, C-20(7):153–161, 1971.

[102] H. S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *JACM*, 20:27–38, 1973.

[103] Paul N. Swarztrauber. The method of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Review*, 19:490–501, 1977.

[104] Paul N. Swarztrauber. A parallel algorithm for solving general tridiagonal equations. *Math. Comp.*, 33:185–189, 1979.

[105] H. A. van der Vorst. Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. *Parallel Computing*, 5:45–54, 1987.

[106] H. H. Wang. Transposing matrices on a vector computer. Report G320-3389, IBM Palo Alto Scientific Center, 1979.

[107] H. H Wang. A parallel method for tridiagonal equations. *ACM Trans. Math. Softw.*, 7:170–182, 1981.

Chapter 9

[108] D. J. Evans. Fast A.D.I. methods for the solution of linear parabolic parabolic partial differential equations involving 2 space dimensions. *BIT*, 17:486–491, 1977.

[109] R. A. Fatoohi and C. E. Grosch. Implementation of an ADI method on parallel computers. ICASE, Report No. 87-43, 1987.

[110] D. J. Hunt, S. J. Webb and A. Wilson. Application of a parallel processor to the solution of finite difference problems, In *Elliptic Problem Solvers*, M. H. Schultz (Ed.), Academic Press, New York, 1981.

[111] S. L. Johnsson, Y. Saad and M. H. Schultz. Alternating direction methods on multiprocessors. *SIAM J. Sci. and Stat. Comput.*, 8:686–700, 1987.

[112] F. Saied, C.-T. Ho, S. L. Johnsson and M. H. Schultz. Solving Schrödinger's equation on the Intel iPSC by the Alternating Direction Method. In *Hypercube Multiprocessors 1987*, M. T. Heath (Ed.), SIAM, Philadelphia, 1987.

[113] P. Vu and C. Yang. Comparing tridiagonal solvers on the CRAY X-MP/416 system. *Cray Channels*, Vol. 9, No. 4, 22-25, 1988.