Parallel Algorithms for the Solution of the Schrodinger Equation

by

Xu Huang

A Thesis Submitted in Fulfilment of the Requirements
for the Degree of Master of Science
in the Department of Computing Science
at the University of Glasgow

May 1989

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ACKNOWLEDGEMENTS

I am greatly indebted to my supervisor Professor A. C. Allison for suggesting this interesting subject, for his continued assistance, guidance and interest.

Many thanks must undoubtedly go to Professor H.Y. Wong for his guidance and assistance in many ways throughout the whole period. My thanks are also due to the High Education Bureau of Guangdong Province and Shantou University for giving me the opportunity to study this course and financial support.

Finally I would like to thank my family and all my friends for their help and encouragement.

Abstract

Many of the traditional numerical algorithms do not map easily onto the architecture of parallel computers that have emerged recently. For the economic use of these expensive machines and to reduce the total computing time, it is necessary to develop efficient parallel algorithms.

The purpose of the thesis is to develop several parallel algorithms for the numerical solution of the Schrodinger equation which arises in many branches of atomic and molecular physics. Common models of systems which are of interest may represent stable configurations of two particles, the bound state or eigenvalue problem. Alternately one may consider either single-channel or multi-channel scattering. All three mathematical models will be investigated in this work.

Emphasis is placed on parallel algorithms for MIMD machines. All the algorithms have been implemented and tested on a transputer network which is a MIMD machine without shared memory.

Existing numerical methods such as those ascribed to Numerov and De Vogelaere have been investigated and parallel versions of them have been developed. Two exponentially fitted versions of the De Vogelaere algorithm have been developed and they are found to be more efficient than the normal De Vogelaere algorithm.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

The past few years have seen a tremendous growth in interest in parallel architectures and parallel processing. Various new machine designs, prototypes and languages for parallel and distributed computing have been proposed and some parallel systems have been made commercially available. Examples of parallel computers are the CRAY-XMP, Amdahl's VP series, BBN Advanced Computers' Butterfly Parallel Processor, CDC's CYBERPLUS, Goodyear Aerospace Corporation's MPP and ASPRO, NCUBE's NCUBE parallel Processing Systems, Sension's TES (Jesshope, 1987). This achievement is largely due to the advances in VLSI technology.

The development of parallel algorithms, however, seems to have lagged behind the rapidly growth of parallel computers. For many problems, the available algorithms are sequential and, certainly, can not take the advantage of the parallel machines. In the area of numerical computation, much effort has been applied to the development of parallel algorithms for various applications. For example, many parallel algorithms in numerical linear algebra have been developed (Ahmed, 1981; Schendel, 1984). However, there is still a long way to go for many applications. Problems arising from the radial Schrodinger equations are the examples. For these problems, few parallel algorithm have been seen in the literature. In this thesis, we will investigate these problems and manage to develop corresponding parallel algorithms.

1.2 Parallel computer models and Performance measures

Since parallelism is possible in three main units of a computer: control unit, processor and store, parallel computers can be designed in various forms, depending on the application. Flynn introduced a classification based on how the machine relates instructions to the data being processed. He defined instruction stream as a sequence of instructions executed by a processor and a data stream as a sequence of data on which the processor operates (Flynn, 1972). According to whether the instruction or data stream are simple or multiple, he proposed four broad classifications of machine organizations:

SISD: Single Instruction stream, Single Data stream;
SIMD: Single Instruction stream, Multiple Data stream;
MISD: Multiple Instruction stream, Single Data stream;
MIMD: Multiple Instruction stream, Multiple Data stream.

SISD machine is the conventional von Neumann model and the MISD category is empty (there are not this kind of machines yet). Only the SIMD and MIMD computers are considered as parallel machines.

The SIMD category consists typically of array processors. All the processors interpret the same instructions and execute them on different data. These processors are under the control of a central control unit which provides instructions and operands for them. A machine of this kind can have a large number of processors. Early examples include the ICL/DAP (Distributed Array Processors) consisting of 4096 processors, the 16384 processor Goodyear MPP (Massively Parallel Processor). The DAP

has recently been reborn as a 1024 processor array design by AMT (Association Memory Technology). To achieve higher performance, algorithms for SIMD machines are best formulated in terms of vector and matrix operations.

MIMD machines consist of more than one processor, each executing a separate instruction stream. There are two kinds of these machines, depending on whether or not processors share their memory. On the shared memory model, all processors share a global memory accessed by a processor-memory interconnection network while the alternative is actually a network with a number of processors, each with its own local memory and each with the ability to communicate with other processors in the network. Networks of transputers fit this latter model and it is on those that we will concentrate since an actual hardware configuration is available.

To achieve high performance, the algorithms for parallel computers should have parallel structures. It is of the utmost importance that the algorithm can assess the speed gain expected from the operation of p processors in parallel. For this purpose the speed-up ratio S is introduced. It is defined as

$$S(p, n, A) = \frac{\text{time required by the serial algorithm}}{T(p, n, A)}$$

where T(p, n, A) is the time required by parallel algorithm A to compute a problem of size n on a parallel machine with p processors.

Another useful measure of parallel algorithm performance is efficiency E(p, n, A) defined by

$$E(p, n, A) = \frac{T(p, n, A)}{p}$$

1.3 Transputer array

The transputer array is a nonshared memory MIMD machine which is built around an innovative chip called the transputer, designed by INMOS Ltd. The transputer consists of a CPU, communication channels and some memory on a single chip. The communication channels, or links, make it possible to build flexible multitransputer networks by connecting transputers through links(Hey, 1988).

1.4 Applications arising from the Schrodinger equation

Differential equations play an important role in scientific research and engineering. In many branches of atomic, molecular and nuclear physics, we often encounter the Schrodinger equation. Usually, there are no analytical solutions for these equations and solution must be obtained numerically.

The one-dimensional radial form of the Schrodinger equation may be written as

$$y''(r) + f(r)y(r) = 0 \qquad (1.1)$$

where

$f = E - l(l+1)/r^2 - V(r)$ and the potential function $V(r)$ vanishes as r increases.

Boundard conditions are imposed over the semiinfinite range $[0, \infty)$, the solution vanishing at the origin. i.e. $y(0)=0$. The

behaviour for large values of r is decided by physical considerations. Here are two problems arising from eq.(1.1):

1) In the case of E<0, the solution will tend to an exponential function. It is only for some special values of E that the solutions display decreasing exponential behaviour and vanish for large values of r. These particular values of E determine the bound eigenstates of the system. The boundary conditions for the eigenvalue problem (or bound state problem) are

$$y(r)=0 \quad \text{at} \quad r=0 \,,$$

$$y(r)\rightarrow 0 \quad \text{at} \quad r\rightarrow\infty \tag{1.2}$$

2) In the case of E>0, the solution of the equation will increase rapidly and then oscillate exhibiting sinusoidal behaviour. The solution is parameterised by the phase shift. The boundary conditions for the problem are thus

$$y(r)=0 \quad \text{at} \quad r=0 \,,$$

$$y(r)\rightarrow A\sin(\omega r - l/2\pi+\delta) \quad \text{as} \quad r\rightarrow\infty \tag{1.3}$$

where $\omega=\sqrt{E}$ and δ is the phase shift required (Allison, 1970; Raptis, 1977; Raptis and Allison 1978).

3) Another application of the Schrodinger equation is in molecular scattering where the model is represented by sets of N coupled differential equations. The coupled differential equations have the form :

$$Y''(r)+F(r)Y(r)=0 \tag{1.4}$$

where F, Y are NxN matrices and the elements in F are given by

$$F_{ij}=\delta_{ij}[k_i^2-l_i(l_i-1)/r^2] \quad \text{as} \quad r\rightarrow\infty \tag{1.5}$$

The boundary conditions are

$$y_{ij}(r)=0 \quad \text{at} \quad r=0$$

$$y_{ij}(r)\rightarrow k_i r j_{l_i}(k_i r)\delta_{ij} + (k_i/k_j)^{1/2}R_{ij}k_i r n_{l_i}(k_i r) \quad \text{at} \quad r\rightarrow\infty, \tag{1.6}$$

where $j_l(x)$ and $n_l(x)$ are the spherical Bessel and Neumann functions, respectively. The R matrix contains all the necessary information about the physical system(Arthurs and Dalgarno,1960; Allison, 1970).

For the above three problems, there are many numerical methods available. Allison(1988) has given an up-to-date review of the numerical methods that have been developed over the years to address these problems. The usual approach is direct step by step numerical integration and two of the mostly widely used methods are Numerov's method (Allison, 1970) and De Vogelaere's method (De Vogelaere, 1955; Allison, 1970; Coleman, 1980). Much effort has been made in enhancing the efficiency of these algorithms. The accuracy of some numerical formulae have been significantly improved by using exponential fitting techniques(Raptis and Allison, 1978; Ixru and Rizea, 1980). For coupled equations, the standard methods generalise to matrix form and, in principle, matrix inversion is required. Efficient algorithms without matrix inversion have been developed(Lester, 1968; Allison, 1970; Baylis and Peel, 1982).

All the algorithms available are based on some recurrence formulae and the integrations are carried out step by step. They are clearly sequential algorithms which make poor use of parallel

and vector computers. Though some of them are written in terms of matrices where parallel algorithms of linear algebra could be applied, the parallelism of most of them has to be exploited by users in the light of the specific computer model they use. For the phase shift problem, an initial investigation has been studied by Ishibashi et al.(1989), who suggested the use of a matrix formalism on vector computers and high performance has been achieved. Ishibashi's approach is based on a Cauchy-type propagation matrix. In this thesis, parallel algorithms for single equation are developed by using matrix formalism but Numerov-like methods are used.

1.5 Contents

Since the Numerov and the De Vogelaere algorithms are frequently used methods for direct integration of these equations, we will give more details about the two algorithms in Chapter 2. These will include their derivations, truncation errors, global errors and stabilities. The Exponential-fitting Numerov algorithm proposed by Raptis and Allison(1978) is studied. The exponential-fitting technique is generalized to both the Numerov and the De Vogelaere algorithms, resulting in various new formulae which are more efficient than the original versions. These new formulae can be applied to parallel computation in a similar fashion to the original methods.

Chapter 3 contains a brief introduction to our Transputer network, which is considered as one kind of nonshared memory MIMD machine, and to Occam, the native language for the transputer. A actual model is illustrated and is used to test all the parallel algorithms in latter chapters. Some simple examples are

given to describe what parallel algorithms for transputer networks look like.

In Chapter 4, parallel algorithms for the eigenvalue problem arising from the Schrodinger equation are investigated. For this problem, one technique is 'matching in the middle'. We integrate both forwards and backwards and then the two solutions are matched at one or two points which are, normally, close to the middle of the integration range. The eigenvalues are calculated by solving the matching equations. In this chapter, we present four parallel algorithms which are based on the 'matching technique' for the problem (Fox, 1968). Three of them are based on the Newton process for the solution of the matching equations while the other is based on the secant method. One of them is obtained directly from the conventional sequential algorithm. High performance can be achieved if only four processors are available and the matching points are properly chosen. The other three are developed by using 4x4 or 2x2 matrix formalisms and they can be run on a parallel machine with any number of processors. Numerov method is the difference formula for these algorithms. All the variants of Numerov method can be similarly used and we take the Exponential-Fitting Numerov method as an example.

Phase shift problems are dealt with in Chapter 5. K. Ishibashi et al (1989) have described an algorithm designed for a vector computer. The algorithm is based on a Cauchy-type propagation matrix(Gordon, 1971). In this chapter, we generalize the matrix formalism technique to Numerov method and change of the

stepsize is implemented. Emphasis is placed on the treatment of the stepsizes which are arranged in advance.

The parallel algorithms for coupled equations are presented in Chapter 6. The technique used is to treat the solution of each equation as an independent process which communicates with other processes only when a coupling term is encountered. Since communication depends on the topology of the network, two kinds of structures, the pipeline structure and the loop structure, are under investigation. The Iterative Numerov method and De Vogelaere method show their superiority since they are relatively independent of the coupling.

Chapter 7 contains the conclusion.

Chapter 2

Numerical methods for Schrodinger equation

2.1 Introduction

The radial form of the Schrodinger equation can be written

$$y''=f(r)y \tag{2.1}$$

where $f(r)= -[E-l(l+1)/r^2-V(r)]$ and $V(r)$ vanishes as r increases.

At the present time, there are several general direct integration methods available for solving initial value problems of the form of eq.(2.1). One approach uses linear multi-step methods such as the Numerov algorithm(Allison, 1970), which is based on polynomial approximation. Due to the oscillatory (or decreasing exponential) behaviours of the solution, some authors have suggested the use of special function approximation and many efficient algorithms, which exactly integrate a special set of functions, have been developed in the past few years(Isaru & Rizea, 1980; Raptis & Cash, 1986). Such methods are generally classified under the heading "exponential fitted".

One problem of most linear multi-step methods is that they require a matrix inversion at each step in solving coupled equations. Though it can be tackled by efficient iterative mechanisms(Allison, 1970), the additional computation would still take a significant proportion of total time consumption. Some hybrid methods, such as De Vogelaere algorithm, have more advantages in handling coupled equation and, therefore, are still widely used in multi-channel problem(De Vogelaere, 1954; Lester, 1968; Allison, 1970; Coleman and Mohamed, 1978).

In this chapter, we investigate some of the direct integration methods and assess their suitability for parallel computation. Since it is not easy to exploit the parallelism of complicated methods, we only choose the simple and efficient methods: Numerov method, De Vogelaere method and their exponential fitted versions.

2.2 Linear multi-step method for second order differential equation

Following Lambert(Lambert, 1973), for the initial-value problem of second-order differential equation

$$y''(r)=f(r,y), \qquad r \in [a\ ,\ b] \tag{2.2}$$

the linear multi-step formula may be written as

$$\sum_{i=0}^{k} \alpha_i y_{n+i} = h^2 \sum_{i=0}^{k} \beta_i\ f(r_{n+i},\ y_{n+i}). \tag{2.3}$$

where $\alpha_k=1$ and $|\alpha_0| + |\beta_0| >0$.

The associated operator of (2.3) is

$$L[y(r),\ h] = \sum_{i=0}^{k} \alpha_i y(r+ih) - h^2 \sum_{i=0}^{k} \beta_i\ y''(r+ih). \tag{2.4}$$

where y(r) is an arbitrary function, continuously differentiable on the interval [a, b]. If we assume that y(r) is m times continuously differentiable, then, on Taylor expanding about the point r, we obtain

$$L[y(r), h] = \sum_{i=0}^{m-1} C_i h^i y^{(i)}(r) + C_m h^m y^{(m)}(r+k\theta h) \qquad (2.5)$$

where $0< \theta <1$

and

$$C_0= \alpha_0 + \alpha_1 + \ldots + \alpha_k$$

$$C_1= \alpha_1 + 2\alpha_2 + \ldots + k\alpha_k \qquad (2.6)$$

$$C_2= \frac{1}{2!}(\alpha_1 + 2^2\alpha_2 + \ldots + k^2\alpha_k) - (\beta_0 + \beta_1 + \ldots + \beta_k)$$

$$C_i= \frac{1}{i!}(\alpha_1 + 2^i\alpha_2 + \ldots + k^i\alpha_k) - \frac{1}{(i-2)!}(\beta_1 + 2^{i-2}\alpha_2 + \ldots + k^{i-2}\alpha_k) \quad \text{for} \quad i>2$$

The linear multi-step method is said to be of order p if, in(2.5), $C_0=C_1=\ldots=C_{p+1}=0, \quad C_{p+2} \neq 0$.

or equivalently

$$L[r^i, h] = 0, \ i=1,2,\ldots p+1, \quad L(r^{p+2}) \neq 0 \qquad (2.7)$$

C_{p+2} is defined as the error constant ,and $C_{p+2}h^{p+2}y^{(p+2)}(r_n)$ the principal local truncation error at r_n. The truncation error can also be represented as

$$L[y(r_n), h] = C_{p+2}h^{p+2}y^{(p+2)}(r_n+k\theta h) \qquad (2.8)$$

where $0 < \theta < 1$.

2.3 The Numerov method

2.3.1 Derivation

Consider the two-step method

$$y_{n+1} + \alpha_1 y_n + \alpha_0 y_{n-1} = h^2(\beta_2 y''_{n+1} + \beta_1 y''_n + \beta_0 y''_{n-1}) \qquad (2.9)$$

Since there are five unknown coefficients in the equation, they can be chosen so that they the first five coefficients given in (2.7) are exact to zero.

$$C_i = 0 \qquad i = 0, 1, 2, 3, 4$$

Solving this system we obtain

$$\alpha_0 = 1, \quad \alpha_1 = 2$$

$$\beta_2 = \beta_0 = \frac{1}{12}, \quad \beta_1 = \frac{10}{12}$$

The formula (2.9) take the form

$$y_{n+1} + 2y_n + y_{n-1} = \frac{h^2}{12}(y''_{n+1} + 10y''_n + y''_{n-1}) \qquad (2.10)$$

It is also be found that

$$C_5 = 0$$

$$C_6 = -\frac{1}{240} \qquad (2.11)$$

and the local truncation error reaches the order of six. The two-step method (2.10) is known as 'Numerov method' .

2.3.2 Local and global error

According to (2.5), (2.11) , the truncation error of eq.(2.10) is

$$L[y(r_{n-1}), h] = C_6 h^6 y^{(6)}(r_{n-1}+2\theta h)$$

$$= -\frac{1}{240} h^6 y^{(6)}(r_{n-1}+2\theta h) \qquad (2.12)$$

where $0 < \theta < 1$.

When a differential equation is solved on the interval [a, b] the global error is the difference between the exact solution and the calculated value at the end point r=b. To investigate the global error of the Numerov method, we choose a stepsize h=(b-a)/N and establish an upper bound valid for all sufficiently small values of h. To make thing simple, we just consider the equation of (2.1)

$$y''=f(r)y$$

In that case, the Numerov formula is

$$y_{n+1} + 2y_n + y_{n-1} = \frac{h^2}{12}(f_{n+1}y_{n+1} + 10f_ny_n + f_{n-1}y_{n-1}) \qquad (2.13)$$

where

$f_n=f(r_n)$, $r_n=a + nh$.

Suppose $y(r_n)$ is the exact solution of the initial value problem, the global error after n steps is

$$e_n=y(r_n) - y_n$$

which satisfy the recurrence relations

$$(1 - \frac{1}{12}h^2f_{n+1})e_{n+1} - (2 + \frac{10}{12}h^2f_n)e_n + (1 - \frac{1}{12}h^2f_{n-1})e_{n-1} = L[y(r_{n-1}),h] \qquad (2.14)$$

By substituting

$$\varepsilon_n=(1 - \frac{1}{12}h^2f_n)e_n ,$$

$$a_n= \frac{f_n}{1 - \frac{1}{12}h^2f_n},$$

$$\delta_n = L[y(r_{n-1}), h]$$

Eq.(2.14) becomes

$$\varepsilon_{n+1}=(2+a_n h^2)\varepsilon_n - \varepsilon_{n-1} +\delta_n \tag{2.15}$$

Summing out the fitst n terms gives

$$\sum_{i=1}^{n} \varepsilon_{i+1} = \sum_{i=1}^{n} [(2+a_i h^2)\varepsilon_i - \varepsilon_{i-1} +\delta_i]$$

which leads to the relation

$$\varepsilon_{n+1}=\varepsilon_n + h^2 \sum_{i=1}^{n} a_i\varepsilon_i + \sum_{i=1}^{n} \delta_i + \varepsilon_1 - \varepsilon_0 \tag{2.16}$$

Let D, A the upper bounds of δ_i, a_i, respetively, and

$\eta_i=\max\{|\varepsilon_1|, |\varepsilon_2|,\ldots,|\varepsilon_i|\}$

Then

$$|\varepsilon_{n+1}| \le |\varepsilon_n|+h^2 nA\eta_n +nD +|\varepsilon_1-\varepsilon_0|$$

Notice that $\eta_{n+1}=\max\{\eta_n , \varepsilon_{n+1}\}$

so

$$\eta_{n+1} \le \eta_n+h^2 nA\eta_n +nD +|\varepsilon_1-\varepsilon_0|$$

$$=(1+h^2 nA)\eta_n+(nD +|\varepsilon_1-\varepsilon_0|)$$

The sulution of this recurrence relation is

$$\eta_n \le (1+h^2 nA)^{n-1}\eta_1+\frac{(1+h^2 nA)^{n-1}-1}{h^2 n A}(nD +|\varepsilon_1-\varepsilon_0|)$$

Now

$$(1+h^2nA)^{n-1}=(1+h(r_n-r_0)A)^{n-1}\leq e^{A(r_n-r_0)^2},$$

we obtain

$$\eta_n \leq e^{A(r_n-r_0)^2}\eta_1+ 1/A(e^{A(r_n-r_0)^2}-1)\{h^{-2}D +h^{-1}|\varepsilon_1-\varepsilon_0|/(r_n-r_0)\}$$

(2.17)

and, at the final point,

$$|\varepsilon_N| \leq \eta_N$$

$$\leq \{c_1|\varepsilon_1|+c_2h^{-1}|\varepsilon_1-\varepsilon_0| \}+c_3h^{-2}D \qquad (2.18)$$

where

$$c_1=e^{A(b-a)^2},\ c_3=(c_1-1)/A,\ c_2=c_3/(b-a)$$

If M is the upper bound of $y^{(6)}(r)$,then, in the absence of rounding error D can be chosen as

$$D=\frac{1}{240}h^6M$$

and

$$|\varepsilon_N|=O(h^4)$$

We have shown that the global error in the Numerov methods is $O(h^4)$.

2.3.3 Stability

To investigate the absolute stability of the method we apply it to the equation

$y''=ky.$

Then eq(2.10) becomes

$$y_{n+1} - ay_n + y_{n-1} = 0 \qquad (2.19)$$

where $a = 2 + \frac{q}{1-q/12}$ and $q = kh^2$.

The characteristic equation of (2.19) is

$$\lambda^2 - a\lambda + 1 = 0 \qquad (2.20)$$

and the method is absolutely stable when neither $|\lambda_1|$ nor $|\lambda_2|$ exceeds unity. Since $\lambda_1\lambda_2 = 1$, the condition required is satisfied if and only if both λ_1 and λ_2 lie on the unit circle and this happens when

$$a^2 - 4 \leq 0 \quad \text{or} \quad 0 \leq -kh^2 \leq 6.$$

2.4 Raptis/Allison method

For general second order equation without first order derivative, the Numerov method is considered as the best two-step method. However, for our problem, the solution of the Schrodinger equation exhibits sinusoidal behaviour in the case of $E>0$ or decreasing exponential behaviour in the case of $E<0$ when r is large enough. Therefore, the use of polynomial approximation is not the natural approach and several techniques based on special function approximation have been proposed. The pioneer work is the multi-step method with exponential fitting developed by Raptis and Allison(1978).

In the multi-step method with exponential fitting, the coefficients may depend on the interval h. The formula becomes

$$\sum_{i=0}^{k} \alpha_i(h) y_{n+i} = h^2 \sum_{i=0}^{k} \beta_i(h) f(r_{n+i}, y_{n+i}) \tag{2.21}$$

and the linear operator L is

$$L[y(r), h] = \sum_{i=0}^{k} \alpha_i(h) y(r+ih) - h^2 \sum_{i=0}^{k} \beta_i(h) y''(r+ih). \tag{2.22}$$

where $\alpha_2(h)=1$.

Raptis and Allison's method is a two-step formula of (2.21) which exactly integrates the solution of equation

$$y''=ky$$

If $k>0$, we let the operator L integrate exactly the functions

$$1,\ r,\ r^2,\ r^3,\ e^{\pm\omega r}$$

where $\omega^2=k$ and then the coefficients are

$$\alpha_2(h) = \alpha_0(h) = 1,\quad \alpha_1(h) = 2$$

$$\beta_0(h) = \beta_2(h) = \frac{(1 - e^{\omega r})^2 - \omega^2 h^2 e^{\omega r}}{\omega^2 h^2 (1 - e^{\omega r})^2} \tag{2.23}$$

$$\beta_1(h) = \frac{\omega^2 h^2 (1 - e^{2\omega r}) - 2(1 - e^{\omega r})^2}{\omega^2 h^2 (1 - e^{\omega r})^2}$$

In the case of $k<0$, let the operator L integrate exactly the functions

$$1,\ r,\ r^2,\ r^3,\ \sin(\omega r),\ \cos(\omega r)$$

where $\omega^2=-k$ and then the coefficients are

$$\alpha_2(h)=\alpha_0(h)=1,\quad \alpha_1(h)=2$$

$$\beta_0(h)=\beta_2(h)=\frac{\omega^2h^2-2(1-\cos(\omega h))}{2\omega^2h^2(1-\cos(\omega h))} \tag{2.24}$$

$$\beta_1(h)=\frac{2-(\omega^2h^2+2)\cos(\omega h)}{\omega^2h^2(1-\cos(\omega h))}$$

The above formulae are affected by servere concellation for small value of h and can be efficiently computed by their power series expansion

$$\alpha_2(h)=\alpha_0(h)=1,\quad \alpha_1(h)=2$$

$$\beta_0(h)=\beta_2(h)=\frac{1}{12}\{1-\frac{1}{20}z^2+\frac{1}{504}z^4+\ldots\}$$

$$\beta_1(h)=\frac{1}{6}\{5+\frac{1}{20}z^2-\frac{1}{504}z^4+\ldots\} \tag{2.25}$$

where $z^2=kh^2$

It can also be found that the leading term of the local truncation error of both cases is given by

$$=-\frac{1}{240}h^6(y^{(6)}-ky^{(4)}) \tag{2.26}$$

2.5 Numerov-like scheme: Further investigation of Raptis-Allison's method

The basis set chosen by Raptis and Allison(1978) is $\{\varphi_1,\varphi_2,1, r, r^2, r^3\}$ where φ_1,φ_2 are the linear independent solutions of equation y"=ky. One may consider that if there exists other basis sets $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$ suitable for constructing similar methods. The generalised schemes of Raptis-Allison's method should have the properties.

1. Accuracy. For an arbitrary function y, which is sufficiently differentiable, the local truncation error should be of order h^6.

2. The equation $y''=ky$ should be integrated exactly.

3. Independence. The coefficients of the formula are independent of r.

4. Symmetry. It is desirable that the coefficients $\alpha_0(h)$, $\alpha_2(h)$, $\beta_0(h)$, $\beta_2(h)$ should satisfy $\alpha_0(h)=\alpha_2(h)$, $\beta_0(h)=\beta_2(h)$ so that the formula can be reduced to the attractive form obtained by substituting $u = \{\alpha_0(h) - h^2\beta_0(h)f\}y$:

$$u_{n+1} + a_n u_n + u_{n-1} = 0 \qquad (2.27)$$

where $a = \dfrac{\alpha_1(h)-h^2\beta_1(h)f}{\alpha_0(h)-h^2\beta_0(h)f}$

Efficient parallel algorithm can be developed from eq.(2.27). Another advantage of the symmetry is that there are no odd order terms in the truncation error. Therefore, if the leading term, i.e. the sixth-order error term, of truncation error vanishes, the truncation error will be reduced to an eight-order error term.

Therefore, we choose $\alpha_0(h)=\alpha_2(h)=1$, $\beta_0(h)=\beta_2(h)$ and the formula becomes

$$y_{n+1} + \alpha_1(h)y_n + y_{n-1} = h^2(\beta_0(h)y''_{n+1} + \beta_1(h)y''_n + \beta_0(h)y''_{n-1})$$

The three unknown coefficients $\alpha_1(h)$, $\beta_0(h)$, $\beta_1(h)$ may be written in the form of power series.

$$\alpha_1(h)=a_0 +a_2(kh^2)+a_4(kh^2)^2+a_6(kh^2)^3+\ldots$$

$$\beta_1(h)=b_0 +b_2(kh^2)+b_4(kh^2)^2+b_6(kh^2)^3+\ldots \qquad (2.28)$$

$$\beta_0(h)=c_0+c_2(kh^2)+c_4(kh^2)^2+c_6(kh^2)^3+\ldots$$

Then

$$L[y(r),\ h]=y(r+h)+\alpha_1(h)y(r)+y(r-h)-h^2(\beta_0 y(r+h)+\beta_1(h)y(r)+\beta_0 y(r-h))$$

$$=F_0(y)+F_2(y)h^2+F_4(y)h^4+F_6(y)h^6+\ldots \qquad (2.29)$$

where

$$F_{2n}(y)=\frac{2}{(2n)!}y^{(2n)}+a_{2n}y \qquad \text{for } n=0,$$

$$F_{2n}(y)=\frac{2}{(2n)!}y^{(2n)}+a_{2n}y$$

$$-\{\frac{2c_0}{(2n-2)!}y^{(2n)}+\frac{2c_2}{(2n-4)!}y^{(2n-2)}+\ldots+2c_{2n-2}y^{(2)}+b_{2n-2}y^{(2)}\}$$

$$=\{\frac{2}{(2n)!}-\frac{2c_0}{(2n-2)!}\}y^{(2n)}-\frac{2c_2}{(2n-4)!}y^{(2n-2)}-\ldots-\frac{2c_{2n-4}}{2!}y^{(4)}$$

$$-(2c_{2n-2}+b_{2n-2})y^{(2)}+a_{2n}y \qquad \text{for} \quad n>0 \qquad (2.30)$$

The first four operators of $F_{2n}(y)$ are

$$F_0(y) = (a_0 + 2)y$$

$$F_2(y) = (1 - 2c_0 - b_0)y'' + a_2 k y$$

$$F_4(y) = (\frac{1}{12} - c_0)y^{(4)} - (2c_2 + b_2)ky'' + a_4 k^2 y$$

$$F_6(y) = (\frac{1}{360} - \frac{1}{12}c_0)y^{(6)} - c_2 k y^{(4)} - (2c_4 + b_4)k^2 y'' + a_6 k^3 y \qquad (2.31)$$

To give a method of $O(h^4)$, the first three term above must be zero. i.e.

$$2 + a_0 = 0,$$

$$1 - 2c_0 - b_0 = 0, \quad a_2 = 0,$$

$$\frac{1}{12} - c_0 = 0, \quad 2c_2 + b_2 = 0, \; a_4 = 0,$$

So, $a_0 = 0$, $b_0 = \frac{5}{6}$, $c_0 = \frac{1}{12}$, $a_2 = 0$, $a_4 = 0$, $2c_2 + b_2 = 0$ and

$$F_6(y) = -\frac{1}{240}y^{(6)} - c_2 k y^{(4)} - (2c_4 + b_4)k^2 y'' + a_6 k^3 y \qquad (2.32)$$

Since we expect that the formula exactly integrates solutions of the equation $y''=ky$, we obtain

$$-\frac{1}{240} - c_2 - (2c_4+b_4) + a_6 = 0 \qquad (2.33)$$

Define the operators D,I by

$$Dy=y', \quad Iy=y$$

Then

$$F_6(y) = -\frac{1}{240}h^6(D^2-kI)\{D^4+(1+240c_2)kD^2+240a_6k^2I\}y \qquad (2.34)$$

Eq.(2.34) provides sufficient information for the derivation of a new class of methods. There are two free parameters in eq.(2.34). For given c_2, a_6, there are six linear independent functions corresponding to the solutions of $F_6(y)=0$. We can prove that there exist unique coefficients $\alpha_1(h)$, $\beta_1(h)$, $\beta_2(h)$ so that the corresponding formula integrates the six linear independent solutions of $F_6(y)=0$ exactly.

Let $p_{2n}(\lambda)$ be the characteristic polynomial of the differential operator $F_{2n}(y)$. $(n>2)$

$$p_{2n}(\lambda)=\{\frac{2}{(2n)!} - \frac{2c_0}{(2n-2)!}\}\lambda^{2n} - \frac{2c_2}{(2n-4)!}\lambda^{2n-2} - \ldots - \frac{2c_{2n-4}}{2!}\lambda^4$$

$$-(2c_{2n-2}+b_{2n-2})\lambda^2+a_{2n}$$

Clearly, the condition that the method can exactly integrates the solution of $F_6(y)=0$ are that $p_6(\lambda)$ is a factor of $p_{2n}(\lambda)$ (n=4, 5, 6,...). Since there are three free coefficients c_4, $(2c_6+b_6)$, a_8 in $p_8(\lambda)$, we can uniquely determine them by letting the remainder $p_8(\lambda)/p_6(\lambda)$ be zero. Similarly, c_{2n}, $(2c_{2n}+b_{2n})$, a_{2n} can be obtained in the same way.

The algorithm is:

1. Choose c_2 , a_6.

2. calculate $2c_4+b_4$ by Eq(2.33)

3. For n=3, 4, 5, ..., calculate c_{2n-2}, $(2c_{2n}+b_{2n})$, a_{2n+2} by letting $p_6(\lambda)$ be a factor of $p_{2n+2}(\lambda)$ and then calculate b_{2n-2} from $(2c_{2n-2}+b_{2n-2})$ and c_{2n-2}.

From the algorithm, we can easily obtain the following existing methods. The algorithm generates the coefficients in the form of power series where $z=kh^2$. A REDUCE package is used to yield the explicit forms of the coefficients(Hearn, 1985).

Raptis-Allison method (1978):
This method is obtained by choosing $a_6=0$, $c_2=-\frac{1}{240}$.

The leading term of L[y,h] is $-\frac{1}{240}h^6(D^2-kI)D^4y$.

The basis set is $\{1, r, r^2, r^3, e^{\omega r}, e^{-\omega r}\}$ in which $\omega^2=k$. The series expression of the coefficients $\alpha_1(h),\beta_1(h),\beta_2(h)$ are

$\alpha_1(h) = -2$

$$\beta_0(h) = \frac{1}{12}\{1 - \frac{1}{20}z^2 + \frac{1}{504}z^4 - \frac{1}{14400}z^6 + \ldots\}$$

$$\beta_1(h) = \frac{1}{6}\{5 + \frac{1}{20}z^2 - \frac{1}{504}z^4 + \frac{1}{14400}z^6 + \ldots\}$$

where $z = kh^2$.

Ixaru and Rizea's method I (1980):

By choosing $a_6=0$, $c_2=-\frac{1}{120}$, we obtain this method. The leading term of L[y,h] is

$$-\frac{1}{240}h^6(D^2 - kI)^2D^2y$$

The basis set is $\{1, r, e^{\omega r}, e^{-\omega r}, re^{\omega r}, re^{-\omega r}\}$. The coefficients $\alpha_1(h),\beta_1(h),\beta_2(h)$ are

$\alpha_1(h)=-2$

$$\beta_0(h) = \frac{z\sinh(z)-2\cosh(z)+2}{z^3\sinh(z)}$$

$$= \frac{1}{12}\{1 - \frac{1}{10}z^2 + \frac{7}{1680}z^4 - \frac{31}{30240}z^6 \ldots\}$$

$$\beta_1(h) = 2\frac{2\cosh^2(z)-2\cosh(z)-z\sinh(z)}{z^3\sinh(z)}$$

$$= \frac{1}{6}\{5 + \frac{1}{10}z^2 + \frac{5}{336}z^4 - \frac{29}{30240}z^6 \ldots\}$$

Ixaru and Rizea's method II (1987):

This new method is generated by letting $a_6 = \frac{1}{240}$, $c_2 = -\frac{1}{80}$

The leading term of L[y,h] is

$$L[y,h]= - \frac{1}{240}h^6(D^2-kI)^3y$$

The basis set is $\{ e^{\omega r}, e^{-\omega r}, re^{\omega r}, re^{-\omega r}, r^2e^{\omega r}, r^2e^{-\omega r}.\}$

$$\alpha_1(h)= 2\frac{z\sinh^2(z)-3\sinh(z)\cosh(z)-z}{3\sinh(z)+z\cosh(z)}$$

$$=-2+ \frac{1}{240}z^6- \frac{1}{2016}z^8+ \frac{1}{11520}z^{10}\ldots$$

$$\beta_0(h)=\frac{z\cosh(z)-\sinh(z)}{z^2(3\sinh(z)+z\cosh(z))}$$

$$=\frac{1}{12}\{ 1- \frac{3}{20}z^2 + \frac{41}{1680}z^4 - \frac{1219}{302400}z^6\ldots\}$$

$$\beta_1(h)= 2\frac{z\sinh^2(z)+\sinh(z)\cosh(z)-z}{z^2(3\sinh(z)+z\cosh(z))}$$

$$=\frac{1}{6}\{ 5+ \frac{3}{20}z^2+ \frac{17}{336}z^4 - \frac{1811}{302400}z^6\ldots\}$$

Gautschi's method I(1961):

It is obtained by leting $a_6=0$, $c_2=- \frac{1}{48}$, the leading term of L[y,h] is

$$- \frac{1}{240}h^6(D^2-kI)(D^2-4kI)D^2y$$

The basis set is $\{1, r, e^{\omega r}, e^{-\omega r}, e^{2\omega r}, e^{-2\omega r}\}$. and the coefficients $\alpha_1(h),\beta_1(h),\beta_2(h)$ are

$$\alpha_1(h)=-2$$

$$\beta_0(h) =\frac{\cosh(2z)-4\cosh(z)+3}{4z^2(\cosh(2z)-\cosh(z))}$$

$$=\frac{1}{12}\{ 1- \frac{1}{4}z^2 + \frac{7}{120}z^4 - \frac{809}{60480}z^6\ldots\}$$

$$\beta_1(h) =\frac{3\cosh(2z)\cosh(z)-4\cosh(2z)+\cosh(z)}{2z^2(\cosh(2z)-\cosh(z))}$$

$$= \frac{1}{6}\{ 5+\frac{1}{4}z^2 - \frac{1}{24}z^4 - \frac{391}{60480}z^6 \ldots\}$$

Gautschi's method II:

To yield this formula, we let $a_6= \frac{3}{20}$, $c_2=- \frac{7}{120}$, the leading term of L[y,h] is

$$- \frac{1}{240}h^6(D^2-kI)(D^2-4kI)(D^2-9kI)y$$

The basis set is $\{ e^{\omega r}, e^{-\omega r}, e^{2\omega r}, e^{-2\omega r}, e^{3\omega r}, e^{-3\omega r}\}$. the coefficients $\alpha_1(h),\beta_1(h),\beta_2(h)$ are

$$\alpha_1(h)= 2\frac{5\cosh(3z)\cosh(2z)-32\cosh(3z)\cosh(z)+27\cosh(2z)\cosh(z)}{27\cosh(3z)-32\cosh(2z)+5\cosh(z)}$$

$$= -2 + \frac{3}{20}z^3 - \frac{1}{12}z^4 + \frac{49}{800}z^5 \ldots$$

$$\beta_0(h)= \frac{3\cosh(3z)-8\cosh(2z)+5\cosh(z)}{z^2(27\cosh(3z)-32\cosh(2z)+5\cosh(z))}$$

$$=\frac{1}{12}\{ 1 - \frac{7}{10}z^2 + \frac{119}{240}z^4 - \frac{53399}{151200}z^6 \ldots\}$$

$$\beta_1(h)= 2\frac{5\cosh(3z)\cosh(2z)-8\cosh(3z)\cosh(z)+3\cosh(2z)\cosh(z)}{z^2(27\cosh(3z)-32\cosh(2z)+5\cosh(z))}$$

$$=\frac{1}{6}\{ 5 + \frac{7}{10}z^2 + \frac{35}{48}z^4 - \frac{55441}{151200}z^6 \ldots\}$$

Other schemes can be derived by choose the free parameters c_2 and a_6. Here is an example:

Example : $a_6=0$, $c_2=0$:

$$L[y,h]= - \frac{1}{240}h^6(D^2-kI)(D^2+kI)D^2y$$

The basis set is $\{1, r, e^{\omega r}, e^{-\omega r}, \sin(\omega r), \cos(\omega r)\}$ in which $\omega^2=|k|$.

$\alpha_1(h)=-2$

$$\beta_0(h)=\frac{1}{12}\{1-\frac{11}{5040}z^4+\ldots\}$$

$$\beta_1(h)=\frac{1}{6}\{5-\frac{23}{1008}z^4+\ldots\}$$

Since there are numerous methods, it is essential to determine the best one, i.e. the one with the smallest truncation error for equation

$$y''=(k+V)y \tag{2.35}$$

where V and its derivatives V', V'', $V^{(3)}$, $V^{(4)}$ are significantly smaller than k. This is equivalent to finding c_2, $(2c_4+b_4)$, a_6 so that for a given solution y of (2.35), $F_6(y)$ has its smallest value.

Let

$a=240c_2$, $b=240(2c_4+b_4)$, $c=-240a_6$

then

$$F_6(y) = -\frac{1}{240}y^{(6)}- c_2ky^{(4)}-(2c_4+b_4)k^2y''+ a_6k^3y$$

$$= -\frac{1}{240}\{y^{(6)}+aky^{(4)}+bk^2y''+ ck^3y\}$$

$$= -\frac{1}{240}\{ (a+b+c-1)k^3y+(2a+b+3)k^2Vy +2(a+3)kV'y'$$

$$+k(aV^2+aV''+3V^2+7V'')y +2(3VV'+2V^{(3)})y'$$

$$+(V^3+7VV''+V^{(4)}+4V'V')y\} \tag{2.36}$$

Since $\|y'\| \approx |k|^{1/2}\|y\|$, the first three dominant terms for large k are $(a+b+c-1)k^3y$, $(2a+b+3)k^2Vy$, $2(a+3)kV'y'$

They vanish only when a, b, c satisfy

$a+b+c-1=0$, $2a+b+3=0$, $a+3=0$.

or $a=-3$, $b=3$, $c=-1$

Therefore, $a_6=\frac{1}{240}$, $c_2=-\frac{1}{80}$, $2c_4+b_4=\frac{1}{80}$ and the corresponding set is $\{ e^{\omega r}, e^{-\omega r}, re^{\omega r}, re^{-\omega r}, r^2e^{\omega r}, r^2e^{-\omega r}.\}$

In that case, we also have

$$F_6(y) =\{V^{(4)}+4V'V'+7VV''+V^3+4V''k)\}y+2(2V^{(3)} +3VV')y'$$

2.6 The De Vogelaere's Algorithm

2.6.1 Derivation

The Numerov and Numerov-like methods such as Raptis/Allison are implicit. This causes no problem when they are applied to a single channel equation of form (2.1). However, when generalize them to coupled equations, we need to do some additional computation for matrix inversion or equivalent. De Vogelaere constructed a hybrid algorithm, which involves the calculation of y', and which is explicit for the equation $y''=f(r, y)$. This algorithm does not require any matrix inversion for coupled equations.

Consider the relations

$$\sum_{i=0}^{k} \alpha_i y_{n+i} = h^2 \sum_{i=0}^{k} \beta_i y''_{n+i} + h \sum_{i=0}^{k} \gamma_i y'_{n+i} \tag{2.37}$$

$$\sum_{\iota=0}^{k} \alpha'_i y'_{n+i} = h \sum_{\iota=0}^{k} \beta'_i y''_{n+i} \tag{2.38}$$

where $\alpha_k=1$, $\alpha'_k=1$.

The associated operators are, respectively,

$$L_1[y(r), h] = \sum_{i=0}^{k} \alpha_i\, y(r+ih) - h^2 \sum_{i=0}^{k} \beta_i\, y''(r+ih) - h \sum_{i=0}^{k} \gamma_i y'(r+ih). \tag{2.39}$$

$$L_2[y'(r), h] = \sum_{\iota=0}^{k} \alpha'_i y'(r+ih) + h \sum_{\iota=0}^{k} \beta'_i y''(r+ih) \tag{2.40}$$

Eq.(2.38) is clearly the general multi-step formula. The values of y, y' at the mesh points can be calculated by the two formulae. Since we are interested in explicit method, at least one of the two formulae must be explicit. In order to balance the accuracy of the two formulae, we let the first one be explicit. In the case of k=2, the coefficients of the formula of (2.37) with highest order are

$$\alpha_2 = 1,\ \alpha_1 = -(1+ \alpha_0)\ ,$$

$$\beta_0 = \frac{1}{2}(3 - \alpha_0),\quad \beta_1 = -\frac{1}{2}(1 + \alpha_0),$$

$$\gamma_0 = \frac{1}{12}(7 - \alpha_0),\quad \gamma_1 = \frac{1}{12}(17 + \alpha_0),$$

The error constant C_5 is $(31-\alpha_0)/6!$. The formula is zero stable when α_0 lies in [-1, 1](Lambert,1973). When α_0 =1, it has the

smallest error constant. But if we choose α_0 =-1, the formula has its tidest form:

$$y_{n+2}=y_n+2hy'_n+\frac{h^2}{3}(\,4y''_{n+1}+2y''_n) \tag{2.41}$$

The best formula for (2.38) is of order $O(h^5)$ is obviously Simpson' role

$$y'_{n+2}=y'_n+\frac{h}{3}(y''_n+4y''_{n+1}+y''_{n+2}) \tag{2.42}$$

With (2.41) and (2.42), y_{n+2}, y'_{n+2} can be calculated step by step. The starting values y_1, y'_1 must be calculated by other formulae. De Vogelaere found that it is not necessary to calculate the first derivative y' at every point if y_{n+1} can be obtained from y_n, y_{n-1}, y'_n. In his method, one general step consists of two steps:

$$y_{2n+1} = y_{2n} + hy'_{2n} + \frac{h^2}{6}(4y''_{2n} - y''_{2n-1}) \tag{2.43a}$$

$$y_{2n+2} = y_{2n} + 2hy'_{2n} + \frac{h^2}{3}(\,4y''_{2n+1} + 2y''_{2n}) \tag{2.43b}$$

$$y'_{2n+2} = y'_n + \frac{h}{3}(y''_{2n} + 4y''_{2n+1} + y''_{2n+2}) \tag{2.43c}$$

The neglected terms for y_{2n+1}, y_{2n+2}, y'_{2n+2} are of order h^4, h^5, h^5 respectively. Though the local truncation error in y_{2n+1} is of order h^4, it contributes a term of order h^6 to the error in y_{2n+2} and h^5 in y'_{2n+2}. More details about the error are discussed later.

One may notice that the above method is not self-starting. To start the integration, y_{-1} can be calculated from

$$y_{-1}=y_0-hy'_0+\frac{1}{2}h^2y''_0 \tag{2.44}$$

Though the leading error of (2.44) is h^3, its contribution to y_1 is h^5 which is less important than the truncation error in y_1.

2.6.2 Local and global error

From (2.43), the linear operators for y_{2n+1}, y_{2n+2}, y'_{2n+2} are, respectively,

$$L_1[y(r),h]=\frac{3}{160}h^4y^{(4)}(r+\theta_1 h)$$

$$L_2[y(r),h]=\frac{2}{45}h^5y^{(5)}(r+\theta_2 h) \qquad (2.45)$$

$$L_3[y(r),h]=-\frac{1}{90}h^5y^{(6)}(r+\theta_3 h)$$

where $0< \theta_1,\theta_2,\theta_3<1$.

We investigate the global error in a manner similar to section 2.2.1 and fix the interval $h=(b-a)/2N$.

Let

$e_n^{(1)} = h(y(r_{2n-1}) - y_{2n-1})$, $e_n^{(2)} = y(r_{2n}) - y_{2n}$, $e_n^{(3)} = y'(r_{2n}) - y'_{2n}$

as recall that from (2.1), $y''(r_{2n}) = f_{2n}y(r_{2n})$.

From (2.37)-(2.39), $e_n^{(i)}$ (i=1,2,3) satisfy the recurrence relations

$$e_{n+1}^{(1)}=hL_1[y(r_{2n}),h] - \frac{1}{6}h^2f_{2n-1}e_n^{(1)} + h(1+\frac{2}{3}h^2f_{2n})e_n^{(2)} + h^2e_n^{(3)}$$

$$e_{n+1}^{(2)}= L_2[y(r_{2n}),h] + h(1+\frac{2}{3}h^2f_{2n})e_n^{(2)} + 2he_n^{(3)} + \frac{4}{3}hf_{2n+1}e_{n+1}^{(1)}$$

$$e_{n+1}^{(3)}= L_3[y(r_{2n}),h] + \frac{1}{3}hf_{2n}e_n^{(2)} + e_n^{(3)} + \frac{4}{3}f_{2n+1}e_{n+1}^{(1)} + hf_{2n+2}e_{n+1}^{(1)}$$

The relation may be written in matrix form as

$$A_n e_{n+1} = B_n e_n + \delta_n$$

where

$$A_n = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{4}{3}hf_{2n+1} & 1 & 0 \\ -\frac{4}{3}f_{2n+1} & -\frac{1}{3}h f_{2n+2} & 1 \end{pmatrix}$$

$$B_n = \begin{pmatrix} -\frac{1}{6}h^2 f_{2n-1} & h+\frac{2}{3}h^3 f_{2n} & h^2 \\ 0 & 1+\frac{2}{3}h^2 f_{2n} & 2h \\ 0 & \frac{1}{3}h^2 f_{2n} & 1 \end{pmatrix}$$

$\delta_n = (hL_1[y(r_{2n}),h]\ ,\ L_2[y(r_{2n}),h]\ ,\ L_3[y(r_{2n}),h]\ )^T$

Hence

$$\|e_{n+1}\|_\infty \leq \|A_n^{-1}B_n\|_\infty \|e_n\|_\infty + \|A_n^{-1}\delta_n\|_\infty$$

The sums of the absolute values of the row elements of the matrix $A_n^{-1}B_n$ are

$$h + O(h^2),\ 1 + 2h + O(h^2),\ 1 + \frac{1}{3}(4|f_{2n+1}| + |f_{2n}|)h + O(h^2)$$

and therefore for all sufficiently small h there exists a positive constant b such that

$$\|A_n^{-1}B_n\|_\infty \leq 1+bh$$

Similarly, the absolute values of the elements of the vector $A_n^{-1}\delta_n$ are

$$\frac{3}{16}h^5|y^{(4)}(r_{2n})|+O(h^6),\qquad \frac{2}{45}h^5|y^{(5)}(r_{2n})|+O(h^6),$$

$$\frac{1}{4}h^5 f_{2n+1}|y^{(4)}(r_{2n})|+\frac{1}{90}h^5|y^{(6)}(r_{2n})|+O(h^6)$$

Therefore, there exists a constant M>0 so that

$$\|A_{\bar{n}}^{-1}\delta_n\|_\infty \le h^5 M$$

so,

$$\|e_{n+1}\|_\infty \le (1+bh)\|e_n\|_\infty + h^5 M$$

$$\le (1+bh)^{n+1}\|e_0\|_\infty + \frac{(1+bh)^{n+1}-1}{bh} h^5 M$$

$$\le e^{(n+1)hb}\|e_0\|_\infty + \frac{e^{(n+1)hb}-1}{b} h^4 M \qquad (2.46)$$

If y_0, y'_0 are exact, $\|e_0\|_\infty$ is $O(h^4)$ which comes from the estimation of y_{-1} and the global errors in y_{2N}, y'_{2N} are bounded by a term proportional to h^4. The error in y_{2N-1} is, however, only to h^3 (notice that $h(y(r_{2N-1})-y_{2N-1})=O(h^4)$)

Thus the global error in De Vogelare method is $O(h^4)$.

2.6.3 Stablity

To investigate the absolute stability of the De Vogelaere's method we apply it to the equation

$$y''=ky$$

Then with $p=kh^2$

$$y_{2n+1}=y_{2n}+ hy'_{2n} + \frac{p}{6}(4y_{2n}-y_{2n-1})$$

$$y_{2n+2}=y_{2n}+2hy'_{2n}+ \frac{p}{3}(4y_{2n+1}+2y_{2n})$$

$$hy'_{2n+2}=hy'_n+ \frac{p}{3}(y_{2n}+4y_{2n+1} +y_{2n+2})$$

Let $v_n=(y_{2n-1}, y_{2n}, hy'_{2n})^T$, we obtain $v_{n+1}=Av_n$ where

$$A=\begin{pmatrix} 1 & 0 & 0 \\ -\frac{4}{3}p & 1 & 0 \\ -\frac{4}{3}p & -\frac{1}{3}p & 1 \end{pmatrix}^{-1}\begin{pmatrix} -\frac{1}{6}p & 1+\frac{2}{3}p & 1 \\ 0 & 1+\frac{2}{3}p & 2 \\ 0 & \frac{1}{3}p & 1 \end{pmatrix}$$

$$=\begin{pmatrix} -\frac{1}{6}p & 1+\frac{2}{3}p & 1 \\ -\frac{2}{9}p^2 & (1+\frac{2}{3}p)(1+\frac{4}{3}p) & 2+\frac{4}{3}p \\ -\frac{2}{9}p^2(1+\frac{1}{3}p) & p(\frac{8}{27}p^2+\frac{14}{9}p+2) & 1+2p+\frac{4}{9}p^2 \end{pmatrix}$$

The characteristic polynomial of matrix A is

$$\det(\lambda I-A)=\lambda^3-(\frac{23}{6}p^2+\frac{4}{3}p+2)\lambda^2-(\frac{2}{3}p^2+\frac{1}{3}p-1)\lambda+\frac{1}{6}p$$

It can be found that none of the eigenvalues of A exceeds unity only when p falls in [-2, 0]. Thus the region of absolute stability is [-2, 0]. (Coleman and Mohamed, 1978)

Coleman (1980) has enhanced the de Vogelaere's method by attaching another term onto the first formula. The accuracy is slightly improved since the local trancation error of the first formula is of order $O(h^5)$ rather than $O(h^4)$. However, this formula becomes implicit for the equation $y''=f(r,y)$ and the algorithm is not suitable for coupled equations. Ixaru and Berceanu (1987) applied the exponential fitting technique to Coleman's scheme. Their exponential fitting version retains the major defect of Coleman's method. In the following sections, we will directly improve the de Vogelaere's method.

2.7. Simple modification of De Vogelaere's algorithm

When we apply the algorithm to equation (2.1), we can estimate the higher order derivatives of y by Blatt's method(Blatt, 1967). That is, for sufficient large r,

$$y^{(p+2)} \approx f y^{(p)} \qquad (p = 0, 1, \ldots)$$

or $y^{(2n)} \approx f^n y$, $y^{(2n+1)} \approx f^n y'$, $(n = 1, 2, \ldots)$

Then the leading terms of the local truncation errors of (2.43a), (2.43b), (2.43c) are approximatively given by

$$R_A = \frac{3}{16} h^4 y_{2n}^{(4)} \approx \frac{3}{16} h^4 f_{2n}^2 y_{2n}$$

$$R_B = \frac{2}{45} h^5 y_{2n}^{(5)} \approx \frac{2}{45} h^5 f_{2n}^2 y_{2n}$$

$$R_c = -\frac{1}{90} h^5 y_{2n}^{(6)} \approx -\frac{1}{90} h^5 f_{2n}^3 y_{2n}$$

By adding the error corrections to the formulae, we obtain the the Modified De Vogelaere's algorithm for eq(2.1):

$$y_{2n+1} = y_{2n} + h y'_{2n} + \frac{h^2}{6}(4 f_{2n} y_{2n} - f_{2n-1} y_{2n-1}) + \frac{3}{16} h^4 f_{2n}^2 y_{2n} \qquad (2.47a)$$

$$y_{2n+2} = y_{2n} + 2h y'_{2n} + \frac{h^2}{3}(4 f_{2n+1} y_{2n+1} + 2 f_{2n} y_{2n}) + \frac{2}{45} h^5 f_{2n}^2 y_{2n} \qquad (2.47b)$$

$$y'_{2n+2} = y'_n + \frac{h}{3}(f_{2n} y_{2n} + 4 f_{2n+1} y_{2n+1} + f_{2n+2} y_{2n+2}) - \frac{1}{90} h^5 f_{2n}^3 y_{2n} \qquad (2.47c)$$

where now the leading terms of the truncation error are:

$$R_{MA} = \frac{3}{16} h^4 (y_{2n}^{(4)} - f_{2n}^2 y_{2n})$$

$$R_{MB} = \frac{2}{45} h^5 (y_{2n}^{(5)} - f_{2n}^2 y_{2n}) \qquad (2.48)$$

$$R_{MC}=-\frac{1}{90}h^5(y_{2n}^{(6)}-f_{2n}^3 y_{2n})$$

When f is a constant, all the three leading terms vanish and the truncation errors of the three formulae are of order h^5, h^6, h^6 respectively.

2.8. Exponential-fitting De Vogelaere method

Similar to Raptis/Allison method, the special function fitting technique can be also applied to the De Vogelaere's method. The three formulae can be treated individually. Since there are four coefficients in each formula, for any basis set come from a forth order homogeneous linear differential equation with constant coefficients, there exist the corresponding formulae. Consider the following formulae:

$$y_{2n+1}=\alpha_{A,1}y_{2n}+\beta_{A,1}hy'_{2n}+\frac{h^2}{6}(4\gamma_{A,1}y''_{2n}-\gamma_{A,0}y''_{2n-1}) \qquad (2.49a)$$

$$y_{2n+2}=\alpha_{B,0}y_n+2\beta_{B,0}hy'_{2n}+\frac{h^2}{3}(4\gamma_{B,1}y''_{2n+1}+2\gamma_{B,0}y''_{2n}) \qquad (2.49b)$$

$$y'_{2n+2}=\beta_{C,0}y'_n+\frac{h}{3}(\gamma_{C,0}y''_{2n}+4\gamma_{C,1}y''_{2n+1}+\gamma_{C,2}y''_{2n+2}) \qquad (2.49c)$$

Allowing all the coefficients to depend on h, we choose $\{1, r, e^{\omega r}, e^{-\omega r}\}$ and $\{e^{\omega r}, e^{-\omega r}, re^{\omega r}, re^{-\omega r}\}$ as basis sets.

Case 1: $\{1, r, e^{\omega r}, e^{-\omega r}\}$ as basis set

$\alpha_{A,1}=1$, $\beta_{A,1}=1$

$$\gamma_{A,0}=6\frac{\sinh(z)-z}{z^2\sinh(z)}$$

$$=1-\frac{7}{60}z^2+\frac{31}{2520}z^4+\ldots$$

$$\gamma_{A,1}=\frac{3}{2}\frac{2\sinh(z)\cosh(z)-\sinh(z)-z\cosh(z)}{z^2\sinh(z)}$$

$$=1+\frac{19}{120}z^2+\frac{1}{1008}z^4+\ldots$$

$\alpha_{B,0}=1,\ \beta_{B,0}=1$

$$\gamma_{B,1}=\frac{3}{4}\frac{\sinh(2z)-2z}{z^2\sinh(z)}$$

$$=1+\frac{1}{30}z^2+\frac{13}{2520}z^4+\ldots$$

$$\gamma_{B,0}=3\frac{z\cosh(z)-\sinh(z)}{z^2\sinh(z)}$$

$$=1-\frac{1}{15}z^2+\frac{2}{315}z^4+\ldots$$

$\beta_{C,0}=1$

$$\gamma_{C,0}=\gamma_{C,2}=3\frac{\sinh(z)-z}{z(\cosh(z)-1)}$$

$$=1-\frac{1}{30}z^2+\frac{1}{840}z^4+\ldots$$

$$\gamma_{C,1}=\frac{3}{2}\frac{z\cosh(z)-\sinh(z)}{z(\cosh(z)-1)}$$

$$=1+\frac{1}{60}z^2-\frac{1}{1680}z^4+\ldots \qquad (2.50)$$

where $\omega^2=k,\ z^2=kh^2$.

The leading terms of truncation errors are

$$R_{EA}=\frac{3}{16}h^4(y^{(4)}-ky''_{2n})$$

$$R_{EB}=\frac{2}{45}h^5(y^{(5)}_{2n}-ky^{(3)}_{2n}) \qquad (2.51)$$

$$R_{EC}=-\frac{1}{90}h^5(y^{(6)}_{2n}-ky^{(4)}_{2n})$$

Case 2: The basis is $\{ e^{\omega r}, e^{-\omega r}; re^{\omega r}; re^{-\omega r}\}$

$$\alpha_{A,1}=\cosh(z)\{1-\frac{z^2\tanh(z)}{\tanh(z)+z}\}$$

$$=1-\frac{1}{8}z^4+\ldots,$$

$$\beta_{A,1}=\frac{\sinh(z)}{z}\,\frac{2\tanh(z)}{\tanh(z)+z}$$

$$=1+\frac{7}{360}z^4+\ldots$$

$$\gamma_{A,0}=6\frac{z\cosh(z)-\sinh(z)}{z^2(\sinh(z)+z\cosh(z))}$$

$$=1-\frac{7}{30}z^2+\frac{71}{1260}z^4+\ldots$$

$$\gamma_{A,1}=\frac{3}{2}\frac{\cosh(z)(z^2\sinh(z)-\sinh(z)+z\cosh(z))}{z^2(\sinh(z)+z\cosh(z))}$$

$$=1+\frac{19}{60}z^2-\frac{17}{2540}z^4+\ldots$$

$$\alpha_{B,0}=1-\frac{\sinh(z)(z^2+z\sinh(z)\cosh(z)-2\sinh^2(z))}{\sinh(z)+z\cosh(z)}$$

$$=1-\frac{1}{45}z^6+\ldots$$

$$\beta_{B,0}=\frac{\sinh^2(z)(2\cosh(z)-z\sinh(z))}{z(\sinh(z)+z\cosh(z))}$$

$$=1-\frac{1}{45}z^4\ldots$$

$$\gamma_{B,1}=\frac{3}{4}\frac{2z\cosh(2z)-\sinh(2z)}{z^2(\sinh(z)+z\cosh(z))}$$

$$=1+\frac{1}{15}z^2+\frac{5}{504}z^4+\ldots$$

$$\gamma_{B,0}=\frac{3}{2}\frac{z^2\sinh(z)-z\cosh(z)(1+\cosh^2(z))+\sinh(2z)\cosh(z)}{z^2(\sinh(z)+z\cosh(z))}$$

$$=1-\frac{2}{15}z^2-\frac{1}{315}z^4+\ldots$$

$$\beta_{C,0}=1$$

$$\gamma_{C,0}=\gamma_{C,2}=3\frac{z\cosh(z)-\sinh(z)}{z^2\sinh(z)}$$

$$=1-\frac{1}{15}z^2+\frac{2}{315}z^4+\ldots$$

$$\gamma_{C,1}=\frac{3}{4}\frac{\sinh(2z)-2z}{z^2\sinh(z)}$$

$$=1+\frac{1}{30}z^2+\frac{13}{2520}z^4+\ldots \qquad (2.52)$$

The leading terms of truncation errors are

$$R_{EA}=\frac{3}{16}h^4(y^{(4)}-2ky''_{2n}+k^2y_{2n})$$

$$R_{EB}=\frac{2}{45}h^5(y^{(5)}_{2n}-2ky^{(3)}_{2n}+k^2y'_{2n}) \qquad (2.53)$$

$$R_{EC}=-\frac{1}{90}h^5(y^{(6)}_{2n}-ky^{(4)}_{2n})$$

The REDUCE package is used to generate all above coefficients(Hearn, 1985).

Chapter 3

Transputer Network

3.1 Transputer and Occam

The launch of the transputer designed by INMOS has opened the way to construct low-cost MIMD computer systems with great flexibility and enormous amounts of processing power. The transputer is a powerful 32bit reduced instruction set computer with some memory on one chip. Unlike other microprocessor chips, the transputer is designed to communicate with other transputers by means of high-speed point-to-point serial 'links' rather than the usual 'bus'. Moreover, these links are entirely implemented on the transputer chip. Present transputers have four links each, with no need for any external support logic.

A transputer network consists of a master transputer and a set of several slave transputers. The master transputer handles the user interface and responds to requests to transfer programs and data to slave transputers. Since each transputer has four links, a wide range of topologies can be configured. Fig3.1 illustrates the examples of link topologies of eight slave transputers.

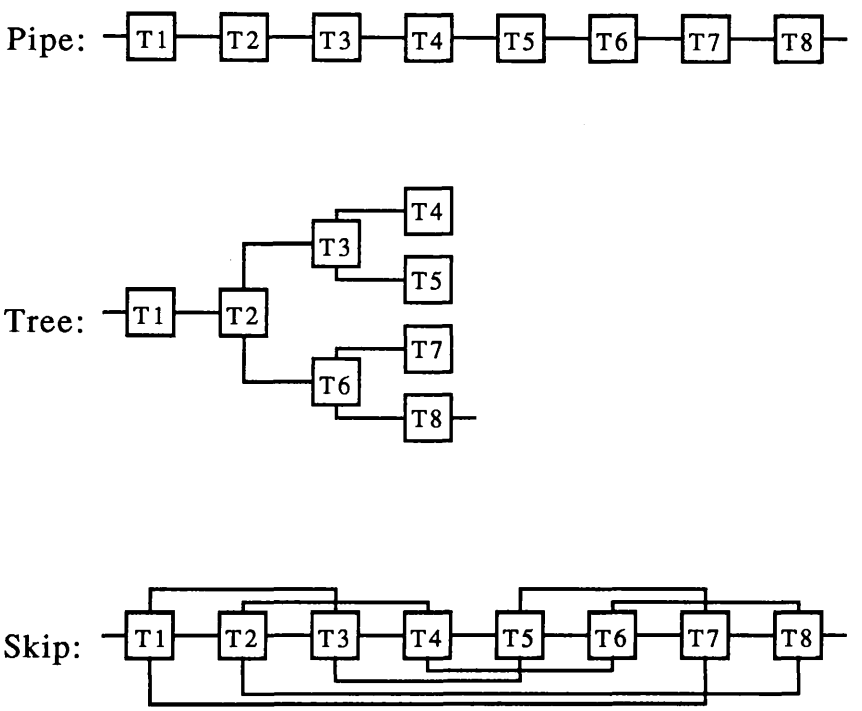
The pipe structure is the simplest one and it is easy to program. Comunication among transputers may require many steps. For example, it takes 7 steps to sent a message from T_1 to T_8. This structure is suitable to parallel algorithms requiring less communication.

In the tree structure, transputers are arranged in a hierachical structure. Communication is controlled by parent transputers.

For example, T_6 controls T_7 and T_8. The maximum length of communication, in this case, is 4.

The skip structure uses all four links of each transputer. Any communication can be completed within two steps. This structure is designed to reduce the total communication time for general algorithms.

Fig3.1



The transputer is programmed in Occam, a parallel programming language associated with the design of the transputer. The most important features of Occam is the use of processes and channels. In Occam, a process is an independent computation, with its own program and data, which can communicate with other processes executing concurrently. A channel provides a one way connection between two concurrent processes. There are three primitive processes in Occam:

v:= e *assign* expression e to various v.

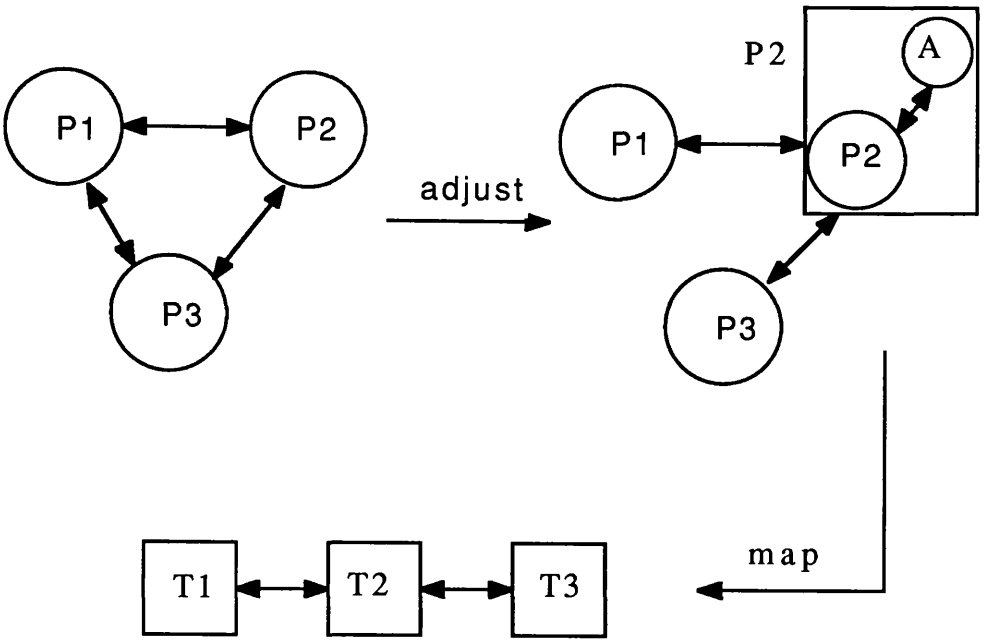
c ! e *output* the value of expression e to channel c

c ? v *input* various v from channel c.

A collection of processes is also a process. Communication between processors is synchronized. If a process sends a message to another by a channel, communication takes place when both processes are ready. The sending and receiving processes then proceed, and the message to be sent are copied from the sending process to the receiving process (INMOS 1985, 1986, Pountain 1986, Burn 1988).

To run a program on a transputer network, one has to map the processes of the program to individual processors. If there are p transputers available, the program should be written as p processes (a collection of processes is also a process), and each will be allocated to a corresponding transputer. External channels of each process would be finally placed onto the corresponding physical links of the network. Since the connecting graph of the logical network should be the subgraph of the connecting graph of the physical network, some adjustment is required. For example, suppose three concurrent processes P_1, P_2, P_3 are connected to each other and we wish to map them to a transputer network consisting of three transputers T_1, T_2, T_3 on which T1 and T3 are not connected directly. We can attach a process A to P_2 which deals with the communication between process P_1 and P_3 and is under the control of P_2. See fig 3.2.

Fig 3.2



3.2 An actual network configuration

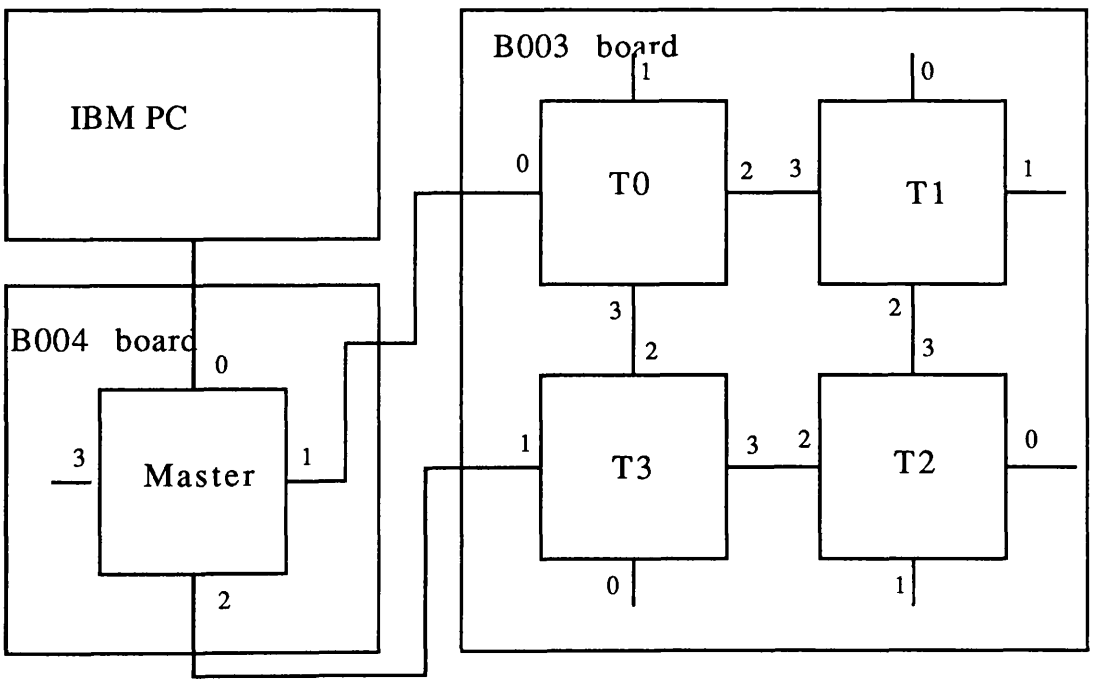
The transputer network for the implementation of parallel algorithms presented in later chapters consists of five transputers, one on board IMS B004 and four on board IMS B003. The IMS B004 is connected to an IBM PC XT which provides the access to the terminal and the filing systems. The transputer on IMS B004 serves as the master transputer which provides links for the use of multitransputer systems. The configuration of the complete system is shown in the diagram on Fig3.3. It can be seen that there are some free links which can be connected to other transputers to form a larger network.

The transputer network is run under the Transputer Development System (TDS) which provides a complete enviroment for the editing, compiling, configuring and executing of Occam2 programs.

A complete program for the network consists of two parts, the EXE part and the PROGRØM part. The EXE part runs on the master transputer and the PROGRØM part runs on the array. Only the EXE part can communicate with I/O devices. An executable unit for a transputer is an independent procedure with only commnication channels on the heading. The PROGRØM part contains several procedures and placement statements which map the procedures onto transputers and logical channels in the headings onto physical links. Procedures mapped onto T_0 or T_3 may have placement statements within them which make communication between the array and the master transputer possible.

Fig3.3

Link map of a network of transputers



3.3 Parallel algorithms for transputer networks

It is impossible to describe what a parallel program for a transputer network looks like because of the topological difference between the algorithm and the network. In general, the program comprises of a number of processes which will be assigned to different transputers. The algorithm for each processor consists of two basic phases: computation and communication. In the computation phase, the processor performs some basic computation. In the communication phase, the processor exchanges necessary results, including some information with its immediate neighbours and may have the task of transferring data to other processors. To illustrate the problems and possible ways of overcoming them we follow through the example of Jacobi iteration in same detail.

Jacobi method for the solution of simultaneous linear equations Ax=b

The Jacobi method is

$$Dx^{(n+1)}=b -(L+U)x^{(n)} ,$$

or $$x^{(n+1)}=-D^{-1}(L+U)x^{(n)} + D^{-1}b , n=0,1, \ldots$$

where A =L+D+U. The iteration is terminated when all the differences between the new values and the old values are less than some tolerance. One approach to implement the iteration is to divide it into N processes for a NxN matrix A, each processing one component of the vector x. Here we consider N=4.

The algorithm for the i-th component of x looks like:

P_i: while not converged do

begin

CPi: compute $x_i^{(n+1)}$ and a local convergent message c_i which is true if $x_i^{(n+1)} - x_i^{(n)}$ falls within the required limit required.

CMi: send $x_i^{(n+1)}$, c_i to the neighbours of P_i and receive all the necessary $x_j^{(n+1)}$, c_j (j<>i) for the next iteration.

convergence:=$c_1 \wedge c_2 \wedge c_3 \wedge c_4$ -- a simple calculation to decide whether or not the process continues.

end;

CMi depends on the practical network. It may depend on the structure of A(for sparse matrix) if we want to reduce the cost of communication. Fig3.3 gives the examples of three practical networks and we will use each in turn on our example. Suppose P_i is mapped to transputer Ti. We let L_{ij} represent the link from Ti to Tj. The link sets of network a, b, c are

a: $\{L_{12}, L_{21}, L_{23}, L_{32}, L_{34}, L_{43}\}$.

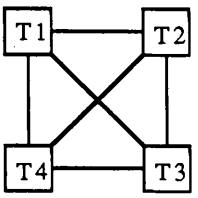
b: $\{L_{12}, L_{21}, L_{13}, L_{31}, L_{14}, L_{41}, L_{23}, L_{32}, L_{24}, L_{42}, L_{34}, L_{43}\}$.

c: $\{L_{12}, L_{21}, L_{23}, L_{32}, L_{24}, L_{42}\}$.

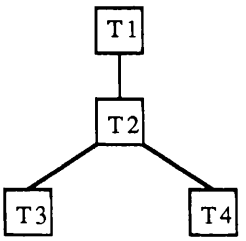
fig 3.3

a)

b)



c)



Case 1: A is a dense matrix

a) The communication can be completed in three step. CM_1 is similar to CM_4 and CM_2 to CM_3. We consider CM_1 and CM_2 :

SEQ {CM_1}	SEQ {CM_2}
1: PAR	PAR
L_{12} ! $x_1^{(n+1)}$, c_1	L_{12} ? $x_1^{(n+1)}$, c_1
L_{21} ? $x_2^{(n+1)}$, c_2	L_{21} ! $x_2^{(n+1)}$, c_2
	L_{23} ! $x_2^{(n+1)}$, c_2
	L_{32} ? $x_3^{(n+1)}$, c_3
	PAR
2: L_{21} ? $x_3^{(n+1)}$, c_3	L_{21} ! $x_3^{(n+1)}$, c_3
	L_{23} ! $x_1^{(n+1)}$, c_1
	L_{32} ? $x_4^{(n+1)}$, c_4
3: L_{21} ? $x_4^{(n+1)}$, c_4	L_{21} ! $x_4^{(n+1)}$, c_4

b) P_i can reach all the other processes and therefore all CM_i are similar. To illustrate, CM_1 can be written as

PAR

L_{12} ! $x_1^{(n+1)}, c_1$ { send to P_2 }

L_{13} ! $x_1^{(n+1)}, c_1$ { send to P_3 }

L_{14} ! $x_1^{(n+1)}, c_1$ { send to P_4 }

L_{21} ? $x_2^{(n+1)}, c_2$ { receive from P_2 }

L_{31} ? $x_3^{(n+1)}, c_3$ { receive from P_3 }

L_{41} ? $x_4^{(n+1)}, c_4$ { receive from P_4 }

c) Only P_2 can communicate with all the other processes. CM_2 may be written as

SEQ

PAR

L_{21} ! $x_2^{(n+1)}, c_2$ { send to P_1 }

L_{23} ! $x_2^{(n+1)}, c_2$ { send to P_3 }

L_{24} ! $x_2^{(n+1)}, c_2$ { send to P_4 }

L_{12} ? $x_1^{(n+1)}, c_1$ { receive from P_1 }

L_{32} ? $x_3^{(n+1)}, c_3$ { receive from P_3 }

L_{42} ? $x_4^{(n+1)}, c_4$ { receive from P_4 }

PAR {transfer data for other Processes}

L_{21} ! $x_3^{(n+1)}, c_3, x_4^{(n+1)}, c_4$ { send to P_1 }

L_{23} ! $x_4^{(n+1)}, c_4, x_1^{(n+1)}, c_1$ { send to P_3 }

L_{24} ! $x_1^{(n+1)}, c_1, x_3^{(n+1)}, c_3$ { send to P_4 }

CM_1, CM_3, CM_4 are similar and they must correspond to CM_2.

Case 2: A is a tridiagonal matrix.

if A is tridiagonal, it is not necessary to access all $x_j^{(n+1)}$ for the next iteration. However, since the convergence parameter is determined by all c_i (i=1,.. 4), every process still requires to communicate with all other processes. If the number of iteration can be estimated, then we can get rid of the communication of c_i This would be very beneficial since for a large network ~~the~~ it is time- consuming and very difficult to handle.

a) The communication can be completed in one step. CM_i only requires to communicate with its immediate neighbours once.

PAR {CM_1}
L_{12} ! $x_1^{(n+1)}$
L_{21} ? $x_2^{(n+1)}$

PAR {CM_2}
L_{12} ? $x_1^{(n+1)}$
L_{21} ! $x_2^{(n+1)}$
L_{23} ! $x_2^{(n+1)}$
L_{32} ? $x_3^{(n+1)}$

b) CM_i in a) can be used for network b)

c) Since T_3 and T_4 are not connected, the communication between them proceeds through T_2. CM_2 is

SEQ {CM_2}
PAR
L_{21} ! $x_2^{(n+1)}$
L_{23} ! $x_2^{(n+1)}$
L_{12} ? $x_1^{(n+1)}$
L_{32} ? $x_3^{(n+1)}$
PAR {transfer data for other Processes}
L_{23} ! $x_4^{(n+1)}$
L_{24} ! $x_3^{(n+1)}$

For a dense matrix of A, the number of communication steps on network a), b), c) are 3, 1, 2 respectively while they are 1, 1, 2 for a tridiagonal matrix of A. From the example we can find that a good network need not be one with small diameter.

The above examples demonstrate the level of programming thought and action required to distribute computation efficiently over a number of processors.

Chapter 4

Parallel Algorithms for an Eigenvalue Problem

4.1 Eigenvalue Problem

A common problem involves a differential system which has solutions only for some particular values of parameter occurring in the system. These particular values are the eigenvalues of the system and the corresponding solutions are the eigenfunctions. For example, the equation $y''= -\lambda y$ has the general solution

$$y=A \sin(\sqrt{\lambda}\, r) + B \cos(\sqrt{\lambda}\, r).$$

If we impose the boundary condictions $y(0)=y(\pi)=0$, we find first that $B=0$, and then that $A=0$ unless $\sqrt{\lambda}$ is integral. The system therefore has nontrivial solutions only if $\lambda=k^2$, $k=1,2,3,$ ....These are the eigenvalues and the corresponding eigenfunctions are

$$y=A \sin(kr).$$

A typical problem arising from the radial Schrodinger equation is the system:

$$y''=(\lambda+g(r))y$$

$$y(0)=0, \quad y(\infty)=0 \tag{4.1}$$

where $g(r)= \ell(\ell+1)/r^2 +V(r)$ and the potential $V(r)$ vanishes as r increases. The above system has nontrivial solutions only for certain positive values of λ, the eigenvalues. These eigenvalues correspond to the bound states in physics.

4.2 Numerical analysis

The approach for solving the eigenvalue problem numerically invokes initial-value methods. We may replace the system by one

of initial type, for which λ is estimated and ultimately adjusted until the solution satisfies the boundary conditions. If we impose another initial condition, say, y'(0)=1, the actual solution, for a given λ, will increase quickly in the region between the origin and the inner turning point (i.e. the first zero of $\lambda+g(r)$). Then it will oscillate in the region in which $\lambda+g(r)$ is negative. After the outer turning point, it exhibits exponential behaviour. For large r, the solution is a combination of $\exp(\sqrt{\lambda}\ r)$ and $\exp(-\sqrt{\lambda}\ r)$. If λ is an eigenvalue, the positive exponential must not be present and the solution tends to a negative exponential function. Otherwise, the solution will not satisfy the boundary condition.

Since the solution is a continuous function of λ, if the solution for given λ_1 increases exponentially for large value of r and the solution for another value λ_2, say, grows to the opposite sign, there must be at least one eigenvalue between λ_1 and λ_2. The eigenvalue can be calculated by a binary search technique.

The disadvantage of the above method is that although we can obtain accurate eigenvalues, the calculated eigenfuntions are poor. The reason is that even if λ is the exact eigenvalue, any minor error (rounding error or truncation error) will introduce a component of the increasing solution and will lead to a divergent solution. If we only integrate forward, we can not get rid of the unwanted increasing solution in the outer region.

For small r, the solution is a combination of $r^{l+1}p_1(r)$ and $r^{-l}p_2(r)$ in which $p_1(r)$ and $p_2(r)$ are polynomials. If we simply integrate backward, the numerical solution will diverge at the orgin since we can not suppress the term of $r^{-l}p_2(r)$. The similar phenomenon will occur. Integration over the whole range in either direction is unsatisfactory.

4.3 Matching method

To tackle this problem, the technique of "matching in the middle" can be used(Fox, 1962). That is that we integrate both forward from the origin and backward from a large value of r and then let the two solutions meet at a reasonable point in the middle. The λ is adjusted until the forward solution and backward solution agree at the matching point.

THe choice of the matching point is not critical for a single equation of this type. However, the matching point should be normally between the inner turning point (the first zero of $\lambda+g(r)$) and the outer turning point (the last zero of $\lambda+g(r)$). The most convenient matching point is the point at which g(r) reaches its minimum because at that point $\lambda+g(r)$ is always negative for any possible eigenvalue.

Since g(r) is singular at the origin and very large and positive for small value of r, the forward solution will increase rapidly and numerical integration near the origin is impossible. We prefer to choose a small value of r_0 as starting point rather than the origin. For the forward solution, we can take the initial conditions $y_f(r_0)=0$, $y_f(r_1)=h$. For the backward solution, we start at a large value r_N of r with the conditions are $y_b(r_N)=t$, $y_b(r_{N-1})=te^{\sqrt{\lambda}\,h}$. The t cannot be fixed arbitrarily, and in fact its value must be calculated in the iterative process, which attempts to match the solutions y_f, y_b at some common point r_A. "Matching" is equivalent to the relations

$$y_f = y_b\,, \quad y_f' = y_b' \qquad \text{at} \quad r= r_A \tag{4.2}$$

Since we are concentrating on Numerov-like methods, we prefer the equivalent approach of matching the two solutions at two adjacent points

$$y_f = y_b, \qquad \text{at} \quad r = r_A$$

$$y_f = y_b, \qquad \text{at} \quad r = r_B \tag{4.3}$$

Obviously, y_f is a function of λ and y_b is a function of both λ and t. Eq(4.3) can be written as a set of non-linear equations about λ and t:

$$y_f(r_A, \lambda) - y_b(r_A, \lambda, t) = 0$$

$$y_f(r_B, \lambda) - y_b(r_B, \lambda, t) = 0 \tag{4.4}$$

The above matching equations are conventionally solved by a Newton process, which suggests changes $\delta\lambda$, δt derived from the simultanous equations.

$$\delta\lambda\frac{\partial}{\partial\lambda}(y_f(r_A,\lambda) - y_b(r_A,\lambda,t)) + \delta t\frac{\partial}{\partial t}(y_f(r_A,\lambda) - y_b(r_A,\lambda,t))$$
$$+ y_f(r_A,\lambda) - y_b(r_A,\lambda,t) = 0$$

$$\delta\lambda\frac{\partial}{\partial\lambda}(y_f(r_B,\lambda) - y_b(r_B,\lambda,t)) + \delta t\frac{\partial}{\partial t}(y_f(r_B,\lambda) - y_b(r_B,\lambda,t))$$
$$+ y_f(r_B,\lambda) - y_b(r_B,\lambda,t) = 0 \tag{4.5}$$

These functions are obtained by solving initial-value problems. The quantities $z=\frac{\partial y}{\partial\lambda}$, $T=\frac{\partial y}{\partial t}$ satisfy the systems

$$z_f'' = f(r)z_f + y_f, \qquad z_f(r_0)=0,\ z_f(r_1)=0 \tag{4.6}$$

$$z_b'' = f(r)z_b + y_b, \qquad z_b(r_N)=t,\ z_f(r_{N-1})=te^{\sqrt{\lambda}\,h}/(2\sqrt{\lambda}) \tag{4.7}$$

$$T_b'' = f(r)T_f, \qquad T_b(r_N)=1,\ T_b(r_{N-1})=e^{\sqrt{\lambda}\,h} \tag{4.8}$$

where $f(r) = \lambda + g(r)$

From (4.8),we have

$$T_b=t^{-1}y_b \tag{4.9}$$

Eqs(4.5) then reduce to the form

$$\delta\lambda(z_f(r_A,\lambda) - z_b(r_A,\lambda,t)) - \delta t t^{-1}y_b(r_A,\lambda,t) + y_f(r_A,\lambda) - y_b(r_A,\lambda,t)=0$$

$$\delta\lambda(z_f(r_B,\lambda) - z_b(r_B,\lambda,t)) - \delta t t^{-1}y_b(r_B,\lambda,t) + y_f(r_B,\lambda) - y_b(r_B,\lambda,t)=0$$

The Newton process requires integration in both directions twice for each iteration: the first for the solution of y_f, y_b at matching points and the second for z_f, z_b. The z_f and z_b satisfy the same equation, though the initial conditions are different. Therefore, we can use the same method for both forward and backward solutions. Applying Numerov method to the equations about y and z, we get

$$(1- \tfrac{1}{12}h^2f_{n+1})y_{n+1}-(2+ \tfrac{10}{12}h^2f_n)y_n+(1- \tfrac{1}{12}h^2f_{n-1})y_{n-1} = 0 \tag{4.10}$$

$$(1- \tfrac{1}{12}h^2f_{n+1})z_{n+1}-(2+ \tfrac{10}{12}h^2f_n)z_n+(1- \tfrac{1}{12}h^2f_{n-1})z_{n-1} = \tfrac{1}{12}h^2(y_{n+1}+10y_n+y_{n-1}) \tag{4.11}$$

Let $$a_n= \frac{12}{1-h^2f_n/12} - 10 = 2+ \frac{h^2f_n}{1-h^2f_n/12}$$

$$b_n= \frac{h^2}{(1-h^2f_n/12)^2} \tag{4.12}$$

and substitute (Allison, 1970)

$$u_n= (1- \tfrac{1}{12}h^2f_n)y_n\ , \tag{4.13}$$

$$v_n = (1 - \frac{1}{12}h^2 f_n)z_n - \frac{1}{12}h^2 y_n \ , \qquad (4.14)$$

where $f_n = \lambda + g(r_n)$, in to Eqs(4.10),(4.11), we obtain

$$u_{n+1} = a_n u_n - u_{n-1} \qquad (4.15)$$

$$v_{n+1} = b_n u_n + a_n v_n - v_{n-1} \qquad (4.16)$$

The integrations can be carried out efficiently by the recurrence formulae (4.15) and (4.16). Each step for both u and v requires 14 arithimetic operations: 8 for coefficients a_n and b_n, 6 for recurrence formulae (4.15) and (4.16). We only calculate u_n and v_n during the integrations. The values of y and z at matching points are finally derived from u and v through relations (4.13) and (4.14).

It is found that almost all the computing time is spend on the computation of these integrations. The total time for each iteration can be sharply reduced by using parallel integration algorithms. The following sections give four parallel algorithms for the calculation of the integrations.

4.4 Method 1: four processes method

For the Newton method, we notice two facts: the forward integration and the backward integration are independent; in each direction, y and z can be coped with in parallel. We can divide the integrations into four concurrent processes P_1, P_2, P_3, P_4, which compute u_f, v_f, u_b, v_b respectively. Since the computation of v involves the value of u, P_1 and P_3 are required to pass the values of u_f, u_b to P_2 and P_4. The n^{th} step for P_1 and P_2 are:

P_1: temp := 1.0 - h_1*(λ+g_n) ; { $h_1 = \frac{1}{12}h^2$ }

a_n := 12.0/temp -10.0;

sent temp, a_n , u_n to P_2 ;

u_{n+1} : = a_n*u_n - u_{n-1} ;

P_2: receive temp, a_n , u_n from P_1 ;

b_n := 12.0*h_1/(temp*temp) ;

v_{n+1} := b_n*u_n + a_n*v_n - v_{n-1}

P_3 and P_4 are are similar to P_1 and P_2. The four processes of P_i can be mapped to four transputers of a transpter network and implement in parallel. This method is the most efficient provided the match points are near the middle of the range of integration. In practice, the cost of communication, which is required by each integration step, should be taken into account and the efficiency cannot reach 100 percent.

4.5 Method 2: 4x4 matrix formalism

The above algorithm is only suitable for a system with four processors. For a system with an arbitrary number of processor, we should try other approaches.

Let

$$D_n = \begin{bmatrix} a_n & 0 & -1 & 0 \\ b_n & a_n & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad 1 \le n \le N-1$$

$$W_{f,n} = \begin{bmatrix} u_{f,n+1} \\ v_{f,n+1} \\ u_{f,n} \\ v_{f,n} \end{bmatrix}, \qquad 0 \le n \le M$$

$$W_{b,n}=\begin{bmatrix} u_{f,n-1} \\ v_{f,n-1} \\ u_{f,n} \\ v_{f,n} \end{bmatrix}, \qquad M+1 \le n \le N$$

where r_M is the first matching point ($r_M = r_A$, $r_{M+1} = r_B$)

Hence eq.(4.15) + eq.(4.16) become

$$W_{f,n} = D_n W_{f,n-1} \qquad 0 \le n \le M$$

$$W_{b,n} = D_n W_{b,n+1} \qquad M+1 \le n \le N$$

Finally,

$$W_{f,A} = D_A D_{A-1} \ldots D_2 D_1 W_{f,0}$$
$$W_{b,B} = D_B D_{B+1} \ldots D_{N-2} D_{N-1} W_{b,N} \qquad (4.17)$$

The solutions at matching points can be obtained from $W_{f,A}$ and $W_{b,B}$ by (4.13) and (4.14). Expressions (4.17) are ideal forms for parallel calculation. Since the two expressions are similar, we just consider the forward solutions.

Suppose there are p (A>>p) processors available. We can let processor i calculate the matrix product:

$$E_i := D_{e_i} D_{e_i-1} .. D_{s_i+1} D_{s_i}$$

and finally let the first processor calculate the result of $W_{f,A}$:

$$W_{f,A} := E_p E_{p-1} \ldots E_1 W_{f,0}$$

s_i and e_i may be decided in this way:

$L=\lfloor M/p \rfloor$, $s_1=1$, $e_1=M-(p-1)L$,

$s_{i+1} = e_i + 1$, $e_{i+1} = e_{i+1} + L-1$ $\quad i=2, 3, \ldots p$

The algorithm for P_i ($i > 1$) is

$E_i := I$;
for n:= s_i to e_i do
    begin
    calculate a_n , b_n ;
    $E_i := D_n E_i$;
    end;
send E_i to P_1 through communication;

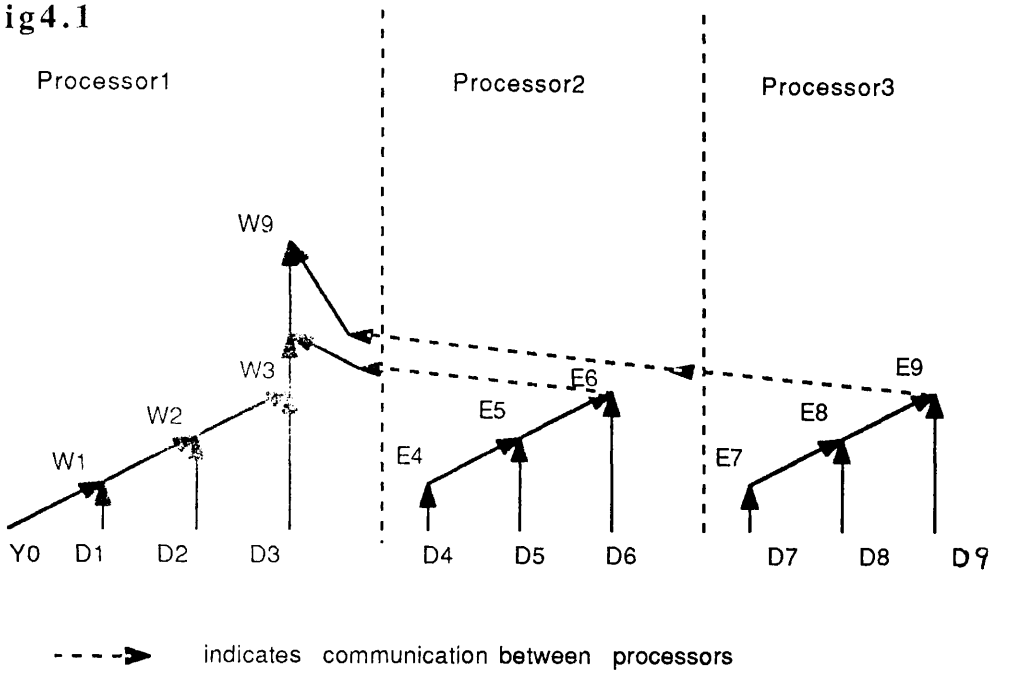
The algorithm for P_1 is

$W_{f,A} := W_{f,0}$;
for n:= s_1 to e_1 do
    begin
    calculate a_n , b_n ;
    $W_{f,A} := D_n W_{f,A}$;
    end;
for i:= 2 to p do
    begin
    receive E_i ;
    $W_{f,A} := E_i W_{f,A}$;
    end;

Fig 4.1 illustrate the algorithms through an example in which p=3, A=9.

The matrix D_n is sparse and the number of arithmetic operations for each matrix multiplication, together with the calculation of a_n and b_n, is 24 and the efficiency of this method might approach **0.58** (14/24).

Fig4.1



4.6 Method 3 : 2x2 matrix formulism

One of the disadvantage of method 2 is that the 4x4 matrix multiplication requires more arithmetic operations. The alternative approach is that we calculate y firstly at all pivotal points and then calculate the values of v at matching points. Instead of 4x4 matrix multiplications, we only form 2x2 matrices.

Let

$$D_n = \begin{bmatrix} a_n & -1 \\ 1 & 0 \end{bmatrix}, \quad C_n = \begin{bmatrix} b_n y_n \\ 0 \end{bmatrix} \qquad 1 \le n \le N-1$$

$$Y_{f,n} = \begin{bmatrix} y_{f,n+1} \\ y_{f,n} \end{bmatrix}, \quad V_{f,n} = \begin{bmatrix} v_{f,n+1} \\ v_{f,n} \end{bmatrix}, \qquad 1 \le n \le M$$

$$Y_{b,n} = \begin{bmatrix} y_{b,n-1} \\ y_{b,n} \end{bmatrix}, \quad V_{b,n} = \begin{bmatrix} v_{b,n-1} \\ v_{b,n} \end{bmatrix}, \qquad M+1 \le n \le N$$

then Eqs(4.15) and (4.16) become

$$Y_n = D_n Y_{n-1} \tag{4.18}$$

$$V_n = D_n V_{n-1} + C_n \tag{4.19}$$

From (4.18) we get

$$Y_{f,n} = D_n D_{n-1} \ldots D_k Y_{f,k-1} \qquad 1 \le k \le n \le M \tag{4.20}$$

We let processor i tackle $Y_{f,n}$ where $s_i \le n \le e_i$. The algorithm for P_i (i<>1) is:

P_iY:

```
calculate a_{s_i} ;
E_{s_i} := D_{s_i} ;
for n :=s_i +1 to e_i do
        begin
        calculate a_n ;
        E_n := D_n E_{n-1} ;
        end;
receive Y_{f,s_i-1} from P_{i-1};
Y_{f,e_i} := E_{e_i} Y_{s_i-1};
if i<>p then send Y_{f,e_i} to P_{i+1};
for n :=s_i to e_i -1;
        Y_{f,n} := E_n Y_{f,e_i-1};  { In practice, we only calculate the
                                       first component of Y_{f,n} }
```

For P_1, it is

P_1Y:

```
for n :=s_1 to e_1 do
        begin
        calculate a_n ;
        Y_{f,n} := D_n Y_{f,n-1} ;
        end;
send Y_{e_1} to P_2 ;
```

After all the $Y_{f,n}$ have been calculated, we can calculate $V_{f,A}$. Following Eq(4.18), we have the general relation:

$$V_{f,n} = D_n V_{f,n-1} + C_n$$

$$= D_n D_{n-1} V_{f,n-2} + C_n + D_n C_{n-1}$$

$$= D_n D_{n-1} \ldots D_k V_{f,k-1} + C_n + D_n C_{n-1} + \ldots + D_n D_{n-1} \ldots D_{k+1} C_k$$

Let $k = s_i$, $n = e_i$, then

$$Y_{f,e_i} = E_{e_i} Y_{f,s_i-1} + T_i$$

where

$$T_i = C_{e_i} + D_{e_i} C_{e_i-1} + \ldots + D_{e_i} D_{e_i-1} \ldots D_{s_i+1} C_{s_i} \qquad i > 1 \qquad (4.21)$$

and finally,

$$V_{f,A} = E_{e_p} E_{e_{p-1}} \ldots E_{e_2} V_{f,e_1} + T_p + E_{e_p} T_{p-1} + \ldots + E_{e_p} E_{e_{p-1}} \ldots E_{e_3} T_2 \qquad (4.22)$$

E_{e_i} (i=2,..., p) have been available in the calculation for Y_n. T_i can be calculated in processor i independently. The algorithm for processor i is:

$P_i V$:

```
calculate b_{s_i} ;
T_i := C_{s_i} ;
for n := s_i + 1 to e_i do
        begin
        calculate b_n ;
        T_i := C_n + D_n T_i;
        end;
send E_{e_i}, T_i to P_1 ;
```

P_1V:

$V_{f,A} := V_{f,0}$
for n :=s_1 to e_1 do
begin
calculate b_n ;
$V_{f,A} := D_n V_{f,A} + C_n$
end;
for i :=2 to p do
begin
receive E_{e_i}, T_i ;
$V_{f,A} := E_{e_i} V_{f,A} + T_i$;
end;

This method requires 19 arithmetic operations for each integration step and therefore the efficiency is estimated to be **0.74** (14/19).

The fig4.2 and fig4.3 show how P_iY and P_iV work.

Fig4.2

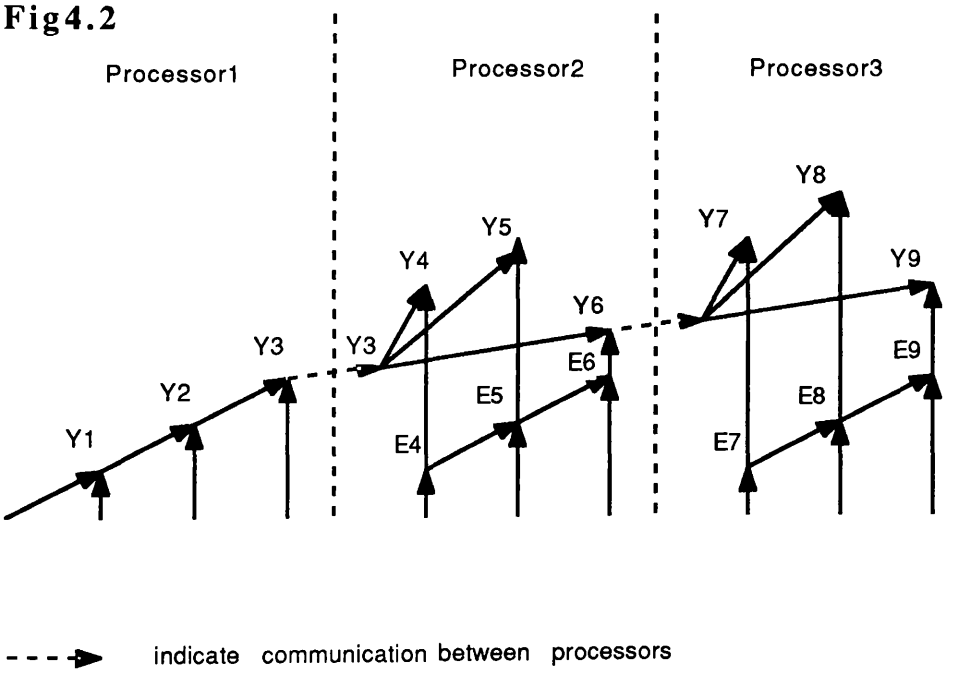
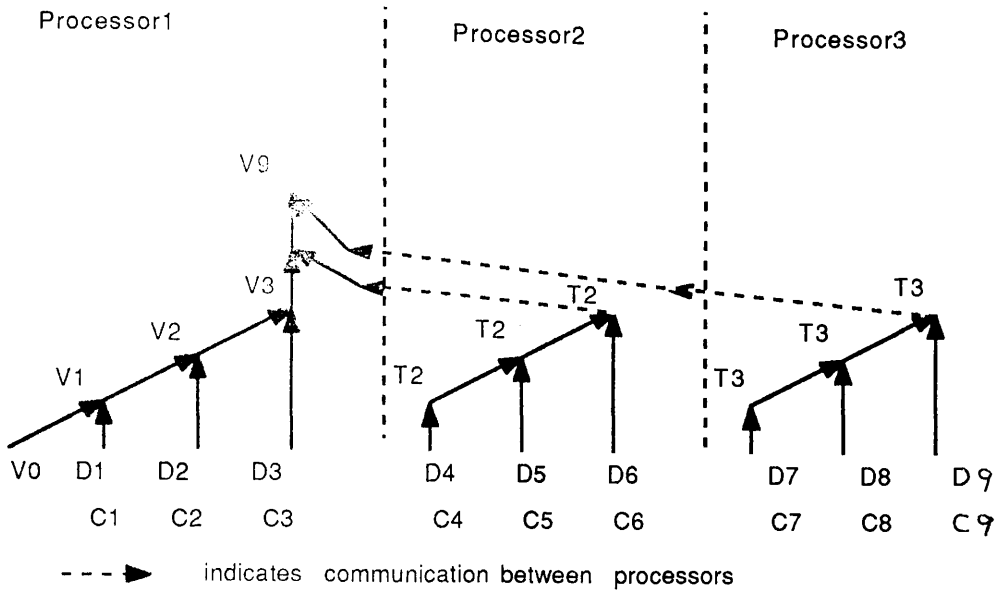


Fig4.3



4.7 Method 4: secant method with 2x2 matrix formulism

It can be seen that the values of y at the matching points can be easily calculated in parallel by 2x2 matrix formulism but the calculation of v (e.g. $\frac{\partial}{\partial\lambda}y$) is far more complicated. To calculate v, we not only have to solve a nonlinear equation, but calculate y at all pivotal points. This additional calculation takes a large proportion of the computing time and makes programming in parallel more difficult. However, the values of v are not required to high accuracy and can be calculated by simple approximation method. For example, we can obtain them approximately from the values of current and previous y at matching points.

Notice that

$$v_f(r, \lambda_n) = \frac{\partial}{\partial\lambda}y_f(r, \lambda_n)$$

$$= [y_f(r, \lambda_n) - y_f(r, \lambda_{n-1})]/(\lambda_n - \lambda_{n-1}) + O(|\lambda_n - \lambda_{n-1}|) \qquad (4.23)$$

and that y_b satisfies:

$$y_b(r, \lambda, t_u) = t_u/t_v y_b(r, \lambda, t_v) \tag{4.24}$$

From the Taylor series

$$y_b(r, \lambda_{n-1}, t_{n-1})$$

$$= \frac{t_{n-1}}{t_n} y_b(r, \lambda_{n-1}, t_n)$$

$$= \frac{t_{n-1}}{t_n}\{y_b(r, \lambda_n, t_n) + (\lambda_{n-1} - \lambda_n)\frac{\partial}{\partial\lambda} y_b(r, \lambda_n, t_n) + O(|\lambda_{n-1} - \lambda_n|^2)$$

So,

$$\frac{\partial}{\partial\lambda} y_b(r,\lambda_n,t_n)$$

$$=\{y_b(r,\lambda_n,t_n) - t_n/t_{n-1} y_b(r,\lambda_{n-1},t_{n-1})\}/(\lambda_n-\lambda_{n-1}) + O(|\lambda_{n-1}-\lambda_n|) \tag{4.25}$$

Let $TA_n(r)=[y_f(r,\lambda_n) - y_f(r,\lambda_{n-1})]/(\lambda_n - \lambda_{n-1})$

$$TB_n(r)=[y_b(r,\lambda_n,t_n) - t_n/t_{n-1} yb(r,\lambda_{n-1},t_{n-1})]/(\lambda_n - \lambda_{n-1}) \tag{4.26}$$

The iterative scheme is now

$$(\lambda_{n+1} - \lambda_n)(TA_n(r_A) - TB_n(r_A)) - (t_{n+1} - t_n)/t_n y_b(r_A,\lambda_n,t_n)$$

$$+ y_f(r_A,\lambda_n) y_b(r_A,\lambda_n,t_n) = 0$$

$$(\lambda_{n+1}- \lambda_n)(TA_n(r_B) - TB_n(r_B)) - (t_{n+1} - t_n)/t_n y_b(r_B,\lambda_n,t_n)$$

$$+ y_f(r_B,\lambda_n) - y_b(r_B,\lambda_n,t_n) = 0 \tag{4.27}$$

Scheme(4.27) is actually the secant method. Its order of convergence is 1.6 (approx.), compared with the second order Newton method. However, it only requires the values of y_f, y_b at

matching points. The parallel algorithm is similar to that in method2, but only uses 2x2 matrix multiplication.

Let

$$D_n = \begin{bmatrix} a_n & -1 \\ 1 & 0 \end{bmatrix}, \qquad 1 \le n \le N-1$$

$$Y_{f,n} = \begin{bmatrix} u_{f,n+1} \\ u_{f,n} \end{bmatrix}, \qquad 1 \le n \le M$$

$$Y_{b,n} = \begin{bmatrix} u_{b,n-1} \\ u_{b,n} \end{bmatrix}, \qquad M+1 \le n \le N$$

Hence

$$Y_{f,n} = D_n Y_{f,n-1} \qquad 0 \le n \le M$$

$$Y_{b,n} = D_n Y_{b,n+1} \qquad M+1 \le n \le N$$

Finally,

$$Y_{f,A} = D_A D_{A-1} \ldots D_2 D_1 Y_{f,0}$$

$$Y_{b,B} = D_B D_{B+1} \ldots D_{N-2} D_{N-1} Y_{b,N} \qquad (4.28)$$

Details of the parallel algorithm for (4.28) can be found in method 2.

The iteration cannot start until the second integration is completed. Therefore, we have to provide two starting values λ_0 and λ_1 for the first two integrations. The choice of starting values of λ is more restrictive than the Newton method.

The algorithm requires 9 arithmetic operations for each integration step while the sequential algorithm needs 7. The efficiency is expected to be **0.78** (7/9).

4.8 Model Problem

To illustrate how the four algorithms work, we take the Morse potential

$$V_0(r) = D\{e^{-2\alpha(r-r_e)} - 2e^{-\alpha(r-r_e)}\} \tag{4.29}$$

where D = 0.18349, α = 1.435 and r_e = 2.31.
The equation is

$$y'' = B(-e + V_0(r))y \tag{4.30}$$

where B = 29156.0 and e is the unknown energy. Since the minimum value of of $V_0(r)$ is -D, the possible values of e should fall in (-D, 0).

In this problem, the difference between e and the minimum value of the potential $V_0(r)$, e+D, is what we actually require. In practice, it is conventional to multiply by a physical constant, which transform the eigenvalue from atomic units into c.g.s. units. So, the eigenvalue is actually E =cm(D+e) where cm = 219474.62. Eq.(4.30) is rewritten as

$$y'' = B(D - E/cm + V_0(r))y \tag{4.31}$$

E is the required eigenvalue. For the Morse potential analytic solutions are known and given, in this case, by

$$E_k = c_1(k + 0.5) - c_2(k + 0.5)^2, \quad k=0, 1, \ldots \tag{4.32}$$

where c_1 = 1580.1868088..., c_2=15.501016...

Values for the first 11 eigenvalues are listed as E_{exact} within following tables.

To transform the equation into standard form, we let λ = B(D - E/cm) , g(r) = $BV_0(r)$, then E can be obtained from λ by

$$E = cm(D - \lambda/B). \tag{4.33}$$

4.9 Implementation

We only consider the first eleven results and choose r_0=1.5 as starting point and r_N=3.5 as end point. r_e=2.31 is the best choice for matching point. We tolerate relative errors up to 0.1×10^{-4} for the iteration. In the secant method, we choose $\lambda_1=\lambda_0$- 0.1 as the second starting value of λ. For comparison, three different stepsizes h=0.01, 0.005, 0.0025 are considered. The corresponding numbers of integration are 200, 400, 800, respectively. All real variables in the programs are double precision.

Tables 4.1&4.2 display the numerical results and errors of the first eleven (0 to 10) eigenvalues calculated by the Newton method and the secant method respectively. There is no significant difference in accuracy between the two methods. The second columns of both tables indicate the initial values for iteration, which are chosen close to the exact eigenvalues.

The times, in seconds, required by the four parallel methods for the calculation of the three eigenvalue E_0, E_5 and E_{10} are shown in Table 4.3 through Table 4.6 (The times spent on the calculation of the potential are not taken into account). The speedup and efficiency are calculated by comparison with the

corresponding sequential algorithms with the same parameters. We find that in principle they are consistent e with the estimation.

For method 1, the speedup and efficiency are largely dependent of the choice of matching point. The ideal matching point for which the load is well balanced over the 4 processors is in the middle of the integration region. For comparison, the algorithm are tested on two different choices of matching points. One is the general choice $r=r_e$ and another is in the middle i.e. $r=(r_0+r_N)/2$. Results are shown in Table 4.3.1 and Table 4.3.2 respectively.

All the other three methods can be run on any number of processors. Here we run them on 2, 3 and 4 four processors. Results are shown in Table 4.4 to Table 4.6. Each table consists of 3 sub-tables which correspond to three different stepsizes.

The final calculation of eigenvalue is inherent sequential. This will slightly affect the efficiency and speedup of all the four methods.

All the algorithms in method 1 to 4 are developed from Numerov formula. As mentioned in chapter 2, exponential fitting Numerov formulae can be efficiently applied to these parallel algorithms with little modification. For an exponential fitting Numerov formula with the symmetric form

$$y_{n+1} + \alpha_1(h)y_n + y_{n-1} = h^2(\beta_0(h)y''_{n+1} + \beta_1(h)y''_n + \beta_0(h)y''_{n-1}) \qquad (4.34)$$

we define

$$a_n = \frac{-\alpha_1(h)+h^2\beta_1(h)f_n}{1-h^2\beta_0(h)f_n},$$

$$b_n = \frac{h^2(-\alpha_1(h)\beta_0(h)+\beta_1(h))}{(1-h^2\beta_0(h)f_n)^2} \tag{4.35}$$

and substitute

$$u_n = (1- \beta_0(h)h^2f_n)y_n \tag{4.36}$$

$$v_n = (1- \beta_0(h)h^2f_n)z_n - \beta_0(h)h^2y_n \tag{4.37}$$

We can still obtain eqs.(4.15) and (4.16). i.e.

$$\begin{aligned} u_{n+1} &= a_n u_n - u_{n-1} \\ v_{n+1} &= b_n u_n + a_n v_n - v_{n-1} \end{aligned} \tag{4.38}$$

Therefore, the formula (4.34) can be applied to all the four parallel methods in a similar fashion to the Numerov formula.

In eigenvalue problems, the potential varies in integration interval and the coefficients should be adjusted at each step. Since the calculation of the coefficients at each step by explicit formulae is time consuming, we prefer the power series formulae. We take the Raptis and Allison algorithm as the example (Raptis and Allison, 1978). In this case, the coefficients satisfy

$$\alpha_1(h)= -2, \qquad \beta_1(h)=1-2\beta_0(h) \tag{4.39}$$

From (4.35)

$$a_n = \frac{-\alpha_1(h)+h^2\beta_1(h)f_n}{1-h^2\beta_0(h)f_n} = 2+ \frac{h^2f_n}{1-h^2\beta_0(h)f_n}$$

$$b_n = \frac{h^2(-\alpha_1(h)\beta_0(h)+\beta_1(h))}{(1-h^2\beta_0(h)f_n)^2} = \frac{h^2}{(1-h^2\beta_0(h)f_n)^2}$$

where $\beta_0 = \frac{1}{12}(1 - \frac{1}{20}h^2 f_n + \ldots)$ and , in practice, we only consider the first two terms.

In parallel implementation, The Raptis and Allison method requires three more arithmetic operations on the calculation of coefficients at each step than the standard Numerov method, but it is far more accurate. The comparison in accuracy is shown in Table 4.7. Three different stepsizes h=0.01, 0.005, 0.0025 are used. Table 4.8 shows the times required by method 3 with Raptis and Allison's formula for the calculation of eigenvalues.

Table 4.1

Numerical results of Newton process(method 1-3), h is the stepsize and Error = E - E_{exact}

k	$E_{initial}$	E_{exact}	h=0.01		h=0.005		h=0.0025	
			E	Error	E	Error	E	Error
0	700.0	786.2182	786.2175	-0.0007	786.2181	-0.0000	786.2182	0.0000
1	2400.0	2335.4029	2335.3984	-0.0045	2335.4027	-0.0003	2335.4029	0.0000
2	3800.0	3853.5857	3853.5703	-0.0153	3853.5847	-0.0010	3853.5856	-0.0001
3	5400.0	5340.7664	5340.7295	-0.0368	5340.7641	-0.0023	5340.7662	-0.0001
4	6700.0	6796.9451	6796.8731	-0.0720	6796.9406	-0.0045	6796.9448	-0.0003
5	8300.0	8222.1217	8221.9989	-0.1228	8222.1141	-0.0077	8222.1212	-0.0005
6	9700.0	9616.2963	9616.1054	-0.1910	9616.2844	-0.0119	9616.2956	-0.0007
7	10900.0	10979.4689	10979.1917	-0.2772	10979.4517	-0.0173	10979.4678	-0.0011
8	12400.0	12311.6395	12311.2574	-0.3820	12311.6157	-0.0238	12311.6380	-0.0015
9	13700.0	13612.8080	13612.3029	-0.5051	13612.7766	-0.0314	13612.8060	-0.0020
10	14850.0	14882.9745	14882.3285	-0.6460	14882.9343	-0.0401	14882.9720	-0.0025

Table 4.2

Numerical results of Secant process(method 4), h is the stepsize and Error = E - E_{exact}

k	$E_{initial}$	E_{exact}	h=0.01		h=0.005		h=0.0025	
			E	Error	E	Error	E	Error
0	700.0	786.2182	786.2175	-0.0007	786.2181	-0.0000	786.2181	-0.0000
1	2400.0	2335.4029	2335.3985	-0.0045	2335.4027	-0.0002	2335.4030	0.0000
2	3800.0	3853.5857	3853.5704	-0.0153	3853.5847	-0.0009	3853.5856	-0.0000
3	5400.0	5340.7664	5340.7295	-0.0369	5340.7641	-0.0023	5340.7662	-0.0002
4	6700.0	6796.9451	6796.8731	-0.0720	6796.9405	-0.0046	6796.9447	-0.0004
5	8300.0	8222.1217	8221.9979	-0.1238	8222.1131	-0.0086	8222.1202	-0.0015
6	9700.0	9616.2963	9616.1036	-0.1928	9616.2827	-0.0137	9616.2938	-0.0025
7	10900.0	10979.4689	10979.1939	-0.2750	10979.4540	-0.0149	10979.4702	-0.0013
8	12400.0	12311.6395	12311.2564	-0.3831	12311.6146	-0.0248	12311.6369	-0.0026
9	13700.0	13612.8080	13612.3021	-0.5059	13612.7759	-0.0321	13612.8053	-0.0027
10	14850.0	14882.9745	14882.3285	-0.6460	14882.9343	-0.0402	14882.9720	-0.0025

Table 4.3.1

Time (in seconds) for method 1 with 4 processor. re=2.31 as matching point

k\h	h=0.01	h=0.005	h=0.0025
0	0.3719 (0.7882)	0.6874 (1.5295)	1.3125 (3.0138)
5	0.3740 (0.7897)	0.5505 (1.5302)	1.0544 (2.2650)
10	0.4487 (0.9863)	0.6881 (1.9137)	1.3194 (3.7740)
speedup	**2.12**	**2.23**	**2.30**
efficiency	**0.53**	**0.56**	**0.57**

*figures in brackets indicate the time required by the corresponding sequential program with the same matching point

Table 4.3.2

Time (in seconds) for method 1 with 4 processor. choose $r=(r_0+r_N)/2$ as matching point.

k\h	h=0.01	h=0.005	h=0.0025
0	0.3428 (0.9853)	0.6532 (1.9137)	1.2733 (3.7681)
5	0.3433 (0.7892)	0.6522 (1.1485)	1.2739 (2.2645)
10	0.3437 (0.7891)	0.6515 (1.5311)	1.2707 (3.0208)
speedup	**3.06**	**3.26**	**3.39**
efficiency	**0.76**	**0.82**	**0.85**

*figures in brackets indicate the time required by the corresponding sequential program with the same matching point

Table 4.4.a h=0.01

Time (in seconds) for method 2

k\p	1	2	3	4
0	0.7882	0.7207	0.5219	0.4327
5	0.7897	0.7216	0.5226	0.4337
10	0.9863	0.9004	0.6513	0.5401
speedup		**1.09**	**1.49**	**1.82**
efficiency		**0.55**	**0.50**	**0.45**

Table 4.4.b h=0.005

Time (in seconds) for method 2

k\p	1	2	3	4
0	1.5295	1.3738	0.9613	0.7614
5	1.5302	1.3734	0.9611	0.7596
10	1.9137	1.7153	1.1986	0.9472
speedup		**1.11**	**1.59**	**2.00**
efficiency		**0.56**	**0.53**	**0.50**

Table 4.4.c h=0.025

Time (in seconds) for method 2

k\p	1	2	3	4
0	3.0138	2.6901	1.8364	1.4156
5	2.2650	2.0173	1.3762	1.0607
10	3.7740	3.3578	2.2895	1.7644
speedup		**1.12**	**1.64**	**2.13**
efficiency		**0.56**	**0.55**	**0.53**

Table 4.5.a h=0.01

Time (in seconds) for method 3

k\p	1	2	3	4
0	0.7882	0.5862	0.4171	0.3354
5	0.7897	0.5866	0.4168	0.3363
10	0.9863	0.7320	0.5201	0.4185
speedup		**1.34**	**1.89**	**2.35**
efficiency		**0.67**	**0.63**	**0.59**

Table 4.5.b h=0.005

Time (in seconds) for method 3

k\p	1	2	3	4
0	1.5295	1.1188	0.7750	0.6033
5	1.5302	1.1167	0.7724	0.6001
10	1.9137	1.3976	0.9641	0.7490
speedup		**1.37**	**1.97**	**2.54**
efficiency		**0.68**	**0.66**	**0.63**

Table 4.5.c h=0.0025

Time (in seconds) for method 3

k\p	1	2	3	4
0	3.0138	2.1852	1.4850	1.1358
5	2.2650	1.6389	1.1123	0.8500
10	3.7740	2.7377	1.8548	1.4122
speedup		**1.38**	**2.03**	**2.65**
efficiency		**0.69**	**0.68**	**0.66**

Table 4.6.a h=0.01

Time (in seconds) for method 4

k\p	1	2	3	4
0	0.4989	0.3400	0.2452	0.2013
5	0.3992	0.2730	0.1963	0.1616
10	0.6982	0.4745	0.3411	0.2799
speedup		**1.47**	**2.03**	**2.48**
efficiency		**0.73**	**0.68**	**0.62**

Table 4.6.b h=0.005

Time (in seconds) for method 4

k\p	1	2	3	4
0	0.9565	0.6494	0.4534	0.3575
5	0.7630	0.5188	0.3617	0.2844
10	1.3350	0.9033	0.6298	0.4951
speedup		**1.47**	**2.11**	**2.68**
efficiency		**0.74**	**0.70**	**0.67**

Table 4.6.c h=0.0025

Time (in seconds) for method 4

k\p	1	2	3	4
0	1.8709	1.2709	0.8678	0.6675
5	1.4954	1.0141	0.6922	0.5308
10	2.6153	1.7665	1.2034	0.9253
speedup		**1.47**	**2.16**	**2.80**
efficiency		**0.74**	**0.72**	**0.70**

Table 4.7

The deviations of numerical results from the exact ones for Numerov scheme and Raptis/Allison scheme. 0.0000 indicates the error is less than 0.00005.

k	$E_{initial}$	E_{exact}	Error (h=0.01)		Error (h=0.005)		Error (h=0.0025)	
			N	R/A	N	R/A	N	R/A
0	700.0	786.2182	-0.0007	-0.0007	-0.0000	-0.0000	-0.0000	-0.0000
1	2400.0	2335.4029	-0.0045	-0.0021	-0.0003	-0.0001	0.0000	-0.0000
2	3800.0	3853.5857	-0.0153	-0.0034	-0.0010	-0.0002	-0.0001	-0.0000
3	5400.0	5340.7664	-0.0368	-0.0047	-0.0023	-0.0003	-0.0001	-0.0000
4	6700.0	6796.9451	-0.0720	-0.0060	-0.0045	-0.0003	-0.0003	-0.0000
5	8300.0	8222.1217	-0.1228	-0.0073	-0.0077	-0.0004	-0.0005	-0.0000
6	9700.0	9616.2963	-0.1910	-0.0086	-0.0119	-0.0005	-0.0007	-0.0000
7	10900.0	10979.4689	-0.2772	-0.0101	-0.0173	-0.0005	-0.0011	-0.0000
8	12400.0	12311.6395	-0.3820	-0.0117	-0.0238	-0.0006	-0.0015	-0.0000
9	13700.0	13612.8080	-0.5051	-0.0134	-0.0314	-0.0007	-0.0020	-0.0000
10	14850.0	14882.9745	-0.6460	-0.0154	-0.0401	-0.0008	-0.0025	-0.0001

Table 4.8.a h=0.01

Time (in seconds) for method 3 with R/A formula

k\p	1	2	3	4
0	0.9555	0.6636	0.4719	0.3784
5	0.9571	0.6645	0.4720	0.3798
10	1.1950	0.8297	0.5894	0.4735
speedup		**1.44**	**2.02**	**2.53**
efficiency		**0.72**	**0.67**	**0.63**

Table 4.8.b h=0.005

Time (in seconds) for method 3 with R/A formula.

k\p	1	2	3	4
0	1.8547	1.2655	0.8771	0.6817
5	1.3923	1.2644	0.8746	0.6781
10	2.7814	1.5719	1.0909	0.8468
speedup		**1.46**	**2.10**	**2.71**
efficiency		**0.73**	**0.70**	**0.68**

Table 4.8.c h=0.0025

Time (in seconds) for method 3 with R/A formula.

k\p	1	2	3	4
0	3.6534	2.4745	1.6812	1.2849
5	2.7446	1.8552	1.2594	0.9614
10	4.5729	3.0907	2.0939	1.5983
speedup		**1.48**	**2.17**	**2.84**
efficiency		**0.74**	**0.72**	**0.71**

Chapter 5

Parallel Algorithms for Phase Shift Problem

5.1 Phase shift problem

The radial form of the Schrodinger equation with positive energy may be written as

$$y'' = f(r)y \tag{5.1}$$

where $f(r)=-(k^2 - \ell(\ell+1)/r^2 - V(r))$ and $V(r)$ vanishes for large r. The boundary condition imposed at origin is

$$y(r) = 0 \quad \text{at} \quad r = 0\,, \tag{5.2}$$

From eq.(5.1) it can be seen that for large r we have

$$y'' \sim -k^2 y. \tag{5.3}$$

Therefore the solution of eq.(5.1) has the asymptotic form

$$y(r) \sim C\sin(kr - \ell\pi/2 + \delta) \tag{5.4}$$

where δ is the "phase shift".

To calculate the phase shift, we can impose the second initial condition $y'(0) = t$ and integrate to sufficiently large r for which the contribution to δ of the term $\ell(\ell+1)/r^2 + V(r)$ can be ignored. The phase shift δ can then be obtained by comparing the solution with (5.4).

In practice $V(r)$ converges to zero much faster than $\ell(\ell+1)/r^2$ so the latter is the dominant term for large r. It is well known that the two linear independent solutions of the equation

$$y'' = -(k^2 - \ell(\ell+1)/r^2)y \tag{5.5}$$

are $krj_\ell(kr)$ and $krn_\ell(kr)$ where $j_\ell(kr)$ and $n_\ell(kr)$ are the spherical Bessel and Neumann functions respectively.

The asymptotic solution of eq(5.1) may take the form

$$y(r) \sim kr(Aj_\ell(kr) + Bn_\ell(kr)) \tag{5.6}$$

which is valid as soon as the effect of V(r) can be neglected.

Since

$$j_\ell(kr) \sim \frac{\sin(kr - \ell\pi/2)}{kr}, \quad n_\ell(kr) \sim \frac{\cos(kr - \ell\pi/2)}{kr}$$

(5.6) may be rewritten as

$$y(r) \sim A\sin(kr - \ell\pi/2) + B\cos(kr - \ell\pi/2) \tag{5.7}$$

and the phase shift can be determined uniquely by

$$\delta = \arctan(B/A) \tag{5.8}$$

If r_a, r_b are sufficiently large, $y(r_a)$, $y(r_b)$ can be represented by the asymptotic form (5.6):

$$y(r_a) = kr_a(Aj_\ell(kr_a) + Bn_\ell(kr_a)),$$

$$y(r_b) = kr_b(Aj_\ell(kr_b) + Bn_\ell(kr_b))$$

so $\delta = \arctan(B/A)$

$$= \arctan\left(\frac{y(r_b)r_a j_\ell(kr_a) - y(r_a)r_b j_\ell(kr_b)}{y(r_b)r_a n_\ell(kr_a) - y(r_a)r_b n_\ell(kr_b)}\right), \tag{5.9}$$

Therefore, to calculate a phase shift, it is necessary to solve the equation (5.1) from the origin to the asymptotic region in which

the V(r) becomes negligible. The phase shift is then obtained by (5.9).

5.2 Parallel algorithm

We have seen the phase shift problem can be reduced to the solution of eq.(5.1). Applying the Numerov algorithm to it, we have

$$(1-\frac{1}{12}h^2f_{n+1})y_{n+1}-(2+\frac{10}{12}h^2f_n)y_n+(1-\frac{1}{12}h^2f_{n-1})y_{n-1} = 0 \qquad (5.10)$$

where $f_n=f(r_n)$.

The technique of matrix formalism used ~~in the~~ in the previous chapter can be applied to the problem. In the eigenvalue problem, the integration region is under restriction and the parallel algorithms given in the previous chapter do not involve changing the stepsize. However, in the phase shift problem, the integration region is semi-infinite, though in numerical computation we can stop the integration at sufficient large r. Since f(r) is large and positive for small r, we must start the integration with a rather small stepsize. Unless we are prepared to change the stepsize as the f(r) decreases, we will waste a lot of machine time in the region where f(r) is small in absolute value. Changing the stepsize in the integration is imperative. At present, the best approach for stepsize control is to monitor the global error and change the stepsize automatically if the estimated error satisfies some conditions. In parallel computation, this method would be very difficult to realize. To change the stepsize, the simple way we can use in parallel computation is to arrange the stepsize in advance according the behaviour of the function f [Blatt, 1967]. That is, we integrate n_1 steps at the stepsize h_1, then n_2 at the stepsize h_2,... and

finally n_m at h_m. (We only consider doubling the stepsize i.e. $h_2=2h_1$, ..., $h_m=2h_{m-1}$ and assume $n_i>2$ for i=1,2,..m.) The stepsize is changed m-1 times and the points at which the stepsize is changed are r_{t_j} where $t_j = \sum_{i=1}^{j} n_i$ (j=1,2,...m-1). The total number of integration steps is $N=\sum_{i=1}^{m} n_i$

Let $w_n(h)=1-\frac{1}{12}h^2f_n$ (5.11)

$$u_n = w_n(h)y_n\ ,\quad a_n=12/w_n(h)\ -10,\quad h=r_n-r_{n-1} \tag{5.12}$$

Then if $n\neq t_j$ (0<j<m), the stepsize is not changed (e.g. $r_{n+1}-r_n=h$, $r_n-r_{n-1}=h$) and eq.(5.10) can be written as

$$u_{n+1}= a_nu_n - u_{n-1}$$

or
$$\begin{bmatrix} u_{n+1} \\ u_n \end{bmatrix} = \begin{bmatrix} a_n & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_n \\ u_{n-1} \end{bmatrix}, \tag{5.13}$$

If the $n=t_j$ (0<j<m) and h is the new stepsize, that is $r_{n+1}-r_n=h_{j+1}=h$, $r_n-r_{n-1}=r_{n-1}-r_{n-2}=h_j=h/2$. (5.10) become

$$(1-\tfrac{1}{12}h^2f_{n+1})y_{n+1}-(2+\tfrac{10}{12}h^2f_n)y_n+(1-\tfrac{1}{12}h^2f_{n-2})y_{n-2} = 0 \tag{5.14}$$

Or $w_{n+1}(h)y_{n+1} - (12-10w_n(h))y_n + w_{n-2}(h)y_{n-2} = 0$

Notice that from (5.12),(5.13)

$$u_n=w_n(h/2)y_n,\ u_{n-2}=w_{n-2}(h/2)y_{n-2}$$

$$u_{n-2}= a_{n-1}u_{n-1} - u_n$$

Hence

$$u_{n+1} - \frac{12-10w_n(h)}{w_n(h/2)}u_n + \frac{w_{n-2}(h)}{w_{n-2}(h/2)}(a_{n-1}u_{n-1} - u_n)=0$$

Rearrange the equation and we have

$$u_{n+1}= \{a_n \frac{w_n(h)}{w_n(h/2)}+\frac{w_{n-2}(h)}{w_{n-2}(h/2)} \}u_n - \frac{w_{n-2}(h)}{w_{n-2}(h/2)} a_{n-1}u_{n-1}$$

Similarly, we have

$$u_{n+2}= a_{n+1}u_{n+1} - \frac{w_n(h)}{w_n(h/2)} u_n$$

Let

$$b_n= \frac{w_n(h)}{w_n(h/2)}, \; c_n= \frac{w_{n-2}(h)}{w_{n-2}(h/2)}, \tag{5.15}$$

Then

$$\begin{bmatrix} u_{n+1} \\ u_n \end{bmatrix} = \begin{bmatrix} a_n b_n+c_n & -a_{n-1}c_n \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_n \\ u_{n-1} \end{bmatrix},$$

$$\begin{bmatrix} u_{n+2} \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} a_{n+1} & -b_n \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_{n+1} \\ u_n \end{bmatrix} = \begin{bmatrix} a_{n+1} & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & b_n \end{bmatrix} \begin{bmatrix} u_{n+1} \\ u_n \end{bmatrix},$$

Therefore, we define matrix D_n by

$$D_n= \begin{bmatrix} a_n & -1 \\ 1 & 0 \end{bmatrix}, \qquad n \neq t_j \; (0<j<m) \tag{5.16a}$$

and

$$D_n= \begin{bmatrix} 1 & 0 \\ 0 & b_n \end{bmatrix} \begin{bmatrix} a_n b_n+c_n & -a_{n-1}c_n \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} a_n b_n+c_n & -a_{n-1}c_n \\ b_n & 0 \end{bmatrix} \qquad n=t_j \; (0<j<m) \tag{5.16b}$$

in which a_n, b_n, c_n are defined by (5.12),(5.15).

The values of u at the final points r_N and r_{N-1} can be represented by

$$\begin{bmatrix} u_N \\ u_{N-1} \end{bmatrix} = D_{N-1}D_{N-2}...D_1 \begin{bmatrix} u_1 \\ u_0 \end{bmatrix} \tag{5.17}$$

Similar to method 4 of last chapter, the N-1 matrix multiplications can be carried out in parallel by *p* processors. To do this, we divide the N-1 matrices into p groups, each consists of the average number of matrices. Let the i^{th} processor P_i calculate the product of

$$E_i := D_{e_i}D_{e_i-1}..D_{s_i+1}D_{s_i} \tag{5.18}$$

where s_i, e_i can be determined by the following scheme

$L=\lfloor N-1/p \rfloor$,

$s_1=1$, $e_1=(N-1)-(p-1)L$,

$s_{i+1} = e_i + 1$, $e_{i+1} = e_{i+1} + L-1$ $\quad i=2, 3, \ldots p$

The E_i produced by P_i is then sent to P_1 on which u_N and u_{N-1} are generated by

$$\begin{bmatrix} u_N \\ u_{N-1} \end{bmatrix} = E_pE_{p-1}...E_1 \begin{bmatrix} u_1 \\ u_0 \end{bmatrix} \tag{5.19}$$

The phase shift can be finally calculated from u_N and u_{N-1}.

Since the stepsize is changed at r_{t_j} ($0<j<m$), the processor P_i should know not only the parameters are k, l, the potential $V(r_n)$ ($n=s_i, s_i+1,\ldots,e_i$), but the starting point r_{s_i} and starting stepsize. It also requires the parameters for changing the stepsize. To do this,

we let PR_i, PH_i be the starting point r_{s_i} and the starting stepsize,respectively, $POINTER_i$ the pointer to the first stepsize changing point from s_i to e_i. To illustrate, we take the example of p=3, m=5, $n_1=n_2=n_3=20$, $n_4=30$, $n_5=50$, $r_0=0.0$, $h_1=0.01$. Then N=140 and we have the following tables:

j	1	2	3	4
t_j	20	40	60	90

i	1	2	3
s_i	1	48	94
e_i	47	93	139
PR_i	0.01	0.92	3.8
PH_i	0.01	0.04	0.16
$POINTER_i$	1	3	0

We can find from the above tables that $t_3=60$ is the first stepsize changing point handled by processor P_2 since $POINTER_i=3$.

These parameters can be generated by a simple algorithm on the host processor and are sent to all other processors.

The skeleton of the algorithm for P_i can be written as

```
r:=PRi;
h:=PHi;
Ei:=I;
j:= POINTERi;
If j=0 then ITURN:=0 else ITURN:= tj;
for n:= si to ei do
    begin
    if n= ITURN then
        begin
        j:=j+1;
        ITURN:= tj;
        h:=2*h;
```

```
        Ei:=DnEi;              {Dn is formed by (5.16b) }
        end
    else
        Ei:=DnEi;              {Dn is formed by (5.16a) }
    r:=r+h;
    end;
```

At each normal point, the calculation of a_n requires 8 arithmetic operations (for $\ell \neq 0$) and therefore each integration step takes 13 arithmetic operations to complete. Compared with the corresponding sequential algorithm which requires 11 arithmetic operations, the efficiency of the parallel algorithm is expected to be 0.85 (11/13).

5.3 Implementation

The parallel algorithm has been tested on a well-known example (Bernstein,1968; Raptis and Allison, 1978) where in eq.(5.1),

$$V(r) = 500\left(\frac{1}{r^{12}} - \frac{1}{r^{6}}\right),$$

We choose $r_0=0.7$ as starting point and $r_N=8.7$ as the final matching point. The algorithm has the ability to handle different arrangement of stepsize, here we test the following three examples.

a: 800 steps at 0.01;

b: 400 steps at 0.01, 200 steps at 0.02;

c: 200 steps at 0.01, 100 steps at 0.02, 100 steps at 0.04.

The total number of integration steps of the three cases are 800, 600 and 400, respectively.

The results, shown in Table 5.1, for the three cases are the same if only three decimal place accuracy is considered. The times, in seconds, required by the algorithm on one to four processors to calculate a phase shift are shown in Table 5.2 which also indicates the speedup and the efficiency of the the algorithm.

From Table 5.2, one may note that the overall speedup and efficiency of the algorithm are not as good as we expect. This is because the final calculation of phase shift, eq.(5.9), is treated sequentially. In fact, the four function values of $j_l(kr_a)$, $n_l(kr_a)$, $j_l(kr_b)$, $n_l(kr_b)$ in eq.(5.9) can be calculated independently. Even each Bessel function can be calculated in parallel since the Bessel function is based on a recurrence formula. However, this has not been done. The original subroutine for Bessel functions returns the values of both Bessel and Neumann functions. This package was designed and written to be efficient when run on a sequential computer. Modern parallel consideration would dictate that we should separate the calculation of Bessel function from that of Neumann function so that the four function values of $j_l(kr_a)$, $n_l(kr_a)$, $j_l(kr_b)$, $n_l(kr_b)$ can be calculated in different processors. Thus we should rewrite the original package. However, the final calculation of phase shift takes up a small proportion of total computation so the improvement gained by parallelising the final calculation of phase shift will be small. Since Occam language is tedious, it is unlikely that parallelising the final calculation of phase shift is necessary.

The expected speedup can be achieved if we ignore the calculation of the phase shift as can be seen in Table 5.3.

Table 5.1
Phase shifts obtained by the algorithm

k	ℓ	δ
3	0	-0.590
3	2	-1.289
3	4	-0.144

Table 5.2.a

Times, in second, required by the algorithm to calculate a phase shift (800 steps at 0.01)

k, ℓ\p	1	2	3	4
3 0	0.8385	0.5072	0.3502	0.2712
3 2	0.9522	0.5661	0.3900	0.3019
3 4	0.9551	0.5700	0.3936	0.3053
average	**0.9153**	**0.5478**	**0.3779**	**0.2930**
speedup		**1.67**	**2.42**	**3.12**
efficiency		**0.84**	**0.81**	**0.78**

Table 5.2.b

Times, in second, required by the algorithm to calculate a phase shift (400 steps at 0.01, 200 steps at 0.02)

k, ℓ\p	1	2	3	4
3 0	0.6388	0.3905	0.2727	0.2131
3 2	0.7252	0.4361	0.3037	0.2373
3 4	0.7290	0.4385	0.3075	0.2405
average	**0.6977**	**0.4217**	**0.2946**	**0.2303**
speedup		**1.65**	**2.37**	**3.03**
efficiency		**0.83**	**0.79**	**0.76**

Table 5.2.c

Times, in second, required by the algorithm to calculate a phase shift (200 steps at 0.01, 100 steps at 0.02, 100 steps at 0.04)

k, ℓ\p	1	2	3	4
3 0	0.4405	0.2722	0.1942	0.1542
3 2	0.4989	0.3028	0.2157	0.1714
3 4	0.5027	0.3060	0.2189	0.1742
average	**0.4807**	**0.2937**	**0.2096**	**0.1666**
speedup		**1.64**	**2.29**	**2.89**
efficiency		**0.82**	**0.76**	**0.72**

Table 5.3.a

Times, in second, required by the algorithm to integrate from r_0 to r_N. N=800. (800 steps at 0.01)

k, ℓ\p	1	2	3	4
3 0	0.8104	0.4759	0.3188	0.2399
3 2	0.9188	0.5313	0.3552	0.2670
3 4	0.9192	0.5326	0.3562	0.2678
average	**0.8761**	**0.5133**	**0.3434**	**0.2582**
speedup		**1.72**	**2.57**	**3.42**
efficiency		**0.86**	**0.86**	**0.85**

Table 5.3.b

Times, in second, required by the algorithm to integrate from r_0 to r_N. N=600. (400 steps at 0.01, 200 steps at 0.02)

k, ℓ\p	1	2	3	4
3 0	0.6088	0.3593	0.2416	0.1821
3 2	0.6921	0.4015	0.2692	0.2028
3 4	0.6933	0.4015	0.2704	0.2035
average	**0.6647**	**0.3874**	**0.2604**	**0.1961**
speedup		**1.72**	**2.55**	**3.39**
efficiency		**0.86**	**0.85**	**0.85**

Table 5.3.c

Times, in second, required by the algorithm to integrate from r_0 to r_N. N=400. (200 steps at 0.01, 100 steps at 0.02, 100 steps at 0.04)

k, ℓ\p	1	2	3	4
3 0	0.4109	0.2414	0.1635	0.1235
3 2	0.4662	0.2687	0.1815	0.1373
3 4	0.4674	0.2694	0.1815	0.1374
average	**0.4482**	**0.2598**	**0.1755**	**0.1327**
speedup		**1.72**	**2.55**	**3.38**
efficiency		**0.86**	**0.85**	**0.84**

Chapter 6

Parallel Algorithms for Solving Coupled Differential Equations

6.1 Coupled Equations

The coupled equations arising from the Schrodinger equation may be transformed into the following form

$$\{ \frac{d^2}{dr^2} + k_i^2 - \frac{l_i(l_i+1)}{r^2} - V_{ii} \} y_{ii} = \sum_{\substack{k=1 \\ k \neq i}}^{n} V_{ik} y_{kj} \qquad (6.1)$$

where $1 \leq i, j \leq N$.

The boundary conditions imposed are

$$y_{ij} = 0 \quad \text{at } r=0, \qquad (6.2)$$

$$y_{ij} \rightarrow k_i r \, j_{l_i}(k_i r)\delta_{ij} + \left(\frac{k_i}{k_j}\right)^{\frac{1}{2}} R_{ij} \, k_i r \, n_{l_i}(k_i r) \quad \text{at } r \rightarrow \infty, \qquad (6.3)$$

where $j_l(x)$ and $n_l(x)$ are the spherical Bessel and Neumann functions, respectively.

In most applications, the R matrix is what we require. As usual, the boundary condition problem may be satisfied by solving an initial value problem. It is obvious that if w is the solution which satisfies (6.2), $w \cdot c$ is also a solution satisfying (6.2) for any arbitrary constant matrix c. If w is non-singular(i.e, the columns of w are linearly independent), $w \cdot c$ is the general solution and we can find a suitable matrix c such that $w \cdot c$ matches to the correct asymptotic form.

For the problem (6.1) in which the matrix elements V_{ij} have no singularities of order two or higher at the origin, the solutions for small r that satisfy (6.2) are given by

$$w_{ij}=\alpha_{ij}r^{l_i+1} \tag{6.4}$$

where α is a constant matrix. Therefore, the second starting value can be given by eq.(6.4) providing the given matrix α is non-singular. The corresponding solutions will not, in general, satisfy the asymptotic boundary conditions (6.3). However, if the columns of the solutions of (6.1) are linearly independent, a suitable linear combination of the solutions can be matched to the correct asymptotic form. That means

$$y_{ij} = \sum_{k=1}^{n} w_{ik}c_{kj} \ , \tag{6.5}$$

or $y=w\cdot c$

The solution y can be matched to the boundary conditions at two values of r large enough so that the terms V_{ij} are negligible. Then, defining the following matrices,

$$R'_{ij} = \left(\frac{k_i}{k_j}\right)^{1/2} R_{ij} \ ,$$

$$M_{ij}=k_i r J_{l_i}(k_i r)\delta_{ij} \ ,$$

$$N_{ij}=k_i r n_{l_i}(k_i r)\delta_{ij} \ ,$$

we find that the asymptotic condition (6.3) can be written as

$$y\rightarrow M + NR' \quad \text{at } r\rightarrow\infty, \tag{6.6}$$

Further for large value of r, the matrix w can be written

$$w=MA+NB, \tag{6.7}$$

and a comparison of (6.6) and (6.7) will lead to the relations

$$c=A^{-1}, \quad R'=BA^{-1} \tag{6.8}$$

The following is the algorithm for the matrix R':

1. Choose the second initial value y_1 at r_0+h. For example, let $\alpha_{ij}=\delta_{ij}$.

2. Integrate from r_0 out to two matching points r_a and r_b in the asymptotic region N times, each time obtaining one column of solutions corresponding to a different column of α.

3. From (6.7) we have $w_a=M_aA+N_aB$, $w_b=M_bA+N_bB$. Notice that M, N are both diagonal and NM=MN. Therefore

$$A=-(N_by_a-N_ay_b)(N_bM_a-N_aM_b)^{-1}$$

$$B=(M_by_a-M_ay_b)(N_bM_a-N_aM_b)^{-1}$$

$$R'=BA^{-1} \tag{6.9}$$

6.2 Numerical Integration Methods

Eqs.(6.1) can be rewritten in matrix form as

$$y''+Fy=0 \tag{6.10}$$

where $$F_{ij}=\{k_i^2 - \frac{l_i(l_i+1)}{r^2}\}\delta_{ij} + V_{ij} \tag{6.11}$$

and $V_{ij}\to 0$ as $r\to\infty$.

Generalizing the Numerov formula, we get

$$y_{n+1}=(I+\frac{1}{12}h^2F_{n+1})^{-1}\{(2I-\frac{10}{12}h^2F_n)y_n+(I+\frac{1}{12}h^2F_{n-1})y_{n-1}\}\ , \qquad (6.12)$$

where I is a unit matrix.

At each integration step, the inversion of the matrix

$$I+\frac{1}{12}h^2F_{n+1},$$

is required. The matrix is strongly diagonally dominant for small r if the step size is chosen properly and tends to a diagonal as r increases. Therefore, an iterative method would rapidly converge and Allison (1970) has proposed the iterative method by substituting

$$Y_{i,n}=(I+\frac{1}{12}h^2f_{i,n})y_{i,n}-\frac{1}{12}h^2g_{i,n}$$

into Eq.(6.12), which becomes

$$Y_{i,n+1}=2Y_{i,n}+h^2(g_{i,n}-f_{i,n}y_{i,n})-Y_{i,n-1}\ , \qquad (6.13)$$

where

$$y_{i,n}=\frac{Y_{i,n}+1/12h^2g_{i,n}}{1+1/12h^2f_{i,n}} \qquad \text{for } 1\le i\le n. \qquad (6.14)$$

and, in the notation of Eq.(6.1),

$$f_{i,n}=(k_i^2-\frac{l_i(l_i+1)}{r^2}-V_{ii})_n\ , \qquad (6.15)$$

$$g_{i,n}=\sum_{\substack{k=1\\k\neq i}}^{n}V_{ik}y_{kj} \qquad (6.16)$$

Eqs.(6.14) and (6.15) may be rewritten as

$$y_{i,n+1}^{(m+1)}=\frac{Y_{i,n}+1/12h^2g_{i,n+1}^{(m+1)}}{1+1/12h^2f_{i,n}} \qquad (6.17)$$

and

$$g_{i,n+1}^{(m)} = \sum_{k>i} V_{ik} y_{k,n+1}^{(m)} + \sum_{k<i} V_{ik} y_{k,n+1}^{(m+1)} \tag{6.18}$$

to define an iterative scheme that converges to the solution. To start the iteration, $y_{i,n+1}^{(0)}$ may be chose as

$$y_{i,n+1}^{(0)} = \frac{Y_{i,n}}{1 - 1/12 h^2 f_{i,n}} \tag{6.19}$$

To estimate the error of the iteration, let

$$V'_{ij} = V_{ij}(1-\delta_{ij})$$

$$g'^{(m)}_{i,n+1} = \sum_{\substack{k=1 \\ k\neq i}}^{n} V_{ik} y_{k,n+1}^{(m)}$$

$$D = \mathrm{diag}(1-h^2 f_{1,n}/12,\ 1-h^2 f_{2,n}/12,\ \ldots,\ 1-h^2 f_{N,n}/12) \tag{6.20}$$

or in matrix form

$$g'^{(m)}_{n+1} = V' y_{n+1}^{(m)} \tag{6.21}$$

In general, we have

$$\|g-g^{(m)}\|_\infty \le \|g-g'^{(m)}\|_\infty = \|V'(y-y^{(m)})\|_\infty \le \|V'\|_\infty \|(y-y^{(m)})\|_\infty$$

Hence, we have

$$\|y_{n+1}-y_{n+1}^{(0)}\|_\infty \le \frac{1}{12}h^2 \|D^{-1}\|_\infty \|g'\|_\infty = \frac{1}{12}h^2 \|D^{-1}\|_\infty \|V'\|_\infty \|y_{n+1}\|_\infty \tag{6.22}$$

$$\|y_{n+1}-y_{n+1}^{(m+1)}\|_\infty \le \frac{1}{12}h^2 \|D^{-1}\|_\infty \|g-g_{n+1}^{(m)}\|_\infty$$

$$\le \frac{1}{12}h^2 \|D^{-1}\|_\infty \|V'\|_\infty \|y_{n+1}-y_{n+1}^{(m)}\|_\infty$$

$$\le \left(\frac{1}{12}h^2 \|D^{-1}\|_\infty \|V'\|_\infty\right)^m \|y_{n+1}-y_{n+1}^{(0)}\|_\infty$$

$$\leq (\frac{1}{12}h^2)^{m+1}||D^{-1}||_\infty^{m+1}||V'||_\infty^{m+1}||y_{n+1}||_\infty \qquad (6.23)$$

Since the truncation error of Numerov formula is $O(h^6)$, at most two corrections are actually required. In that case, the error of the iteration is significantly smaller than the truncation error. In practice, one correction is enough near the asymptotic region. The integration over the entire range is repeated N times, corresponding to N different columns of the matrix α and the resultant solution matrices used in the match process.

The De Vogelaere's method can also applied to coupled equations(Lester,1968; Allison,1970). Generalizing to eq.(6.10), De Vogelaere's algorithm becomes

$$\begin{aligned} y_{n+1/2} &= y_n + \frac{1}{2}h\,y'_n + \frac{h^2}{24}(4y''_n - y''_{n-1/2}), \\ y_{n+1} &= y_n + hy'_n + \frac{h^2}{6}(y''_n + 2y''_{n+1/2}), \\ y'_{n+1} &= y'_n + \frac{h}{6}(y''_n + 4y''_{n+1/2} + y''_{n+1}), \end{aligned} \qquad (6.24)$$

where

$$y''_{j/2} = -F_{j/2}y_{j/2} \quad j=-1,0,1,2,.. \qquad (6.25)$$

The N equations given by(6.10) may be integrated to the asymptotic region N times, each time corresponding to a different column of the matrix α which may be regarded as a linearly independent set of initial derivatives. To start the integration, there is a extra work to calculate $y''_{-1/2}$, which is readily obtained from $y_{-1/2}$ given to sufficient accuracy by

$$y_{-1/2} = y_0 - \frac{1}{2}h y'_0 + \frac{h^2}{8}y''_0 \qquad (6.26)$$

6.3 Parallel algorithm

It is clear that each column of y in eqs.(6.1) can be solved completely independently. However, what we are interested in here is to solve one column of y in parallel and therefore the algorithm is suitable for a set of general coupled equations. The great advantage of both the Iterative Numerov method and De Vogelaere's method is that each equation in the coupled equations can be handled relatively independently. If the coupled equations consist of N equations, we give each of them to a different process. The N processes will finally be distributed to p actual processors--transputers. For the Iterative Numerov method, the coupling only enters through the term $g^{(m)}_{i.n+1}$ in eq.(6.18). In the case of the De Vogelaere's method, the coupling is encountered when calculating the term $y''_{j/2}$ in eq.(6.25). To handle the coupling, each process has to access the results produced by others and the communication among processes is required. On the transputer network, this can be done by channel communication. For the Iterative Numerov method, there is a extra work to exchange the convergence parameters which decide whether or not the iteration is completed.

Distribution

In practical implementation, the number of equations of a set of coupled equations is varied. Suppose we need to solve a set of coupled equations with N equations on a network with p transputers. To distribute the N processes for the N equations

into p transputers, we can map the first n_1 processes onto the first transputer, n_2 processes onto the second one, and n_p onto the last one. n_i can be chosen by

$$n_i = \begin{cases} \lfloor N/p \rfloor + 1 & i < (N \bmod p) \\ \lfloor N/p \rfloor & \text{else} \end{cases} \qquad (6.24)$$

To illustrate, if p=5 and n=16 , the series of n_i is 4, 3, 3, 3, 3. It is one of the most efficient division because whatever division used, at least one transputer handles more than 3 processes.

Communication

A transputer has four links for communication. They can connect to any other transputers. To minimize the cost of communication, it is desirable to form a network with minimum diameter. However, it is very difficult for a programmer to write a procedure for a complicated network. To simplify the problem, we use a pipeline or a loop structure. In each transputer, a buffer Y is declared for the communication of y. i.e. Y[i] is used to store y_i.

In the case of a loop structure with p transputers, communication can be completed in $\lfloor p/2 \rfloor$ steps. Here is the procedure for communication of y for the i^{th} transputer P_i:

```
PROC COM([]REAL64 Y, CHAN from.left, to.left, from.right, to.right)
  INT i:
  SEQ j=0 FOR ⌊p/2⌋
    PAR
      from.left  ?    [Y FROM S[n.left[j+1]] FOR L[n.left[j+1]]]
      to.left    !    [Y FROM S[n.right[j]] FOR L[n.right[j]]]
      from.right ?    [Y FROM S[n.right[j+1]] FOR L[n.right[j+1]]]
      to.right   !    [Y FROM S[n.left[j]] FOR L[n.left[j]]]
```

:

In the procedure, S[k] indicates the index of the first equation assigned to processor P_k and L[k] gives the number of equations assigned P_k. So $L[k]=n_k$ and $S[1]=1$, $S[2]=n_1+1$,... ,$S[k]=S[k-1]+n_k$.

For the i^{th} processor P_i, n.right[j] indicates the processor j steps to the left of P_i while n.left[j] indicates the processor j steps to the right. They can be determined by

$$\text{n.right[j]}=\begin{cases} i+j & \text{for } i+j \le p \\ i+j-p+1 & \text{for } i+j>p \end{cases}$$

$$\text{n.left[j]}=\begin{cases} i-j & \text{for } i-j>0 \\ i-j+p & \text{for } i+j \le 0 \end{cases}$$

For example, in the case of p=5, N=16, S[k], L[k] are given by

k	1	2	3	4	5
L[k]	4	3	3	3	3
S[k]	1	5	8	11	14

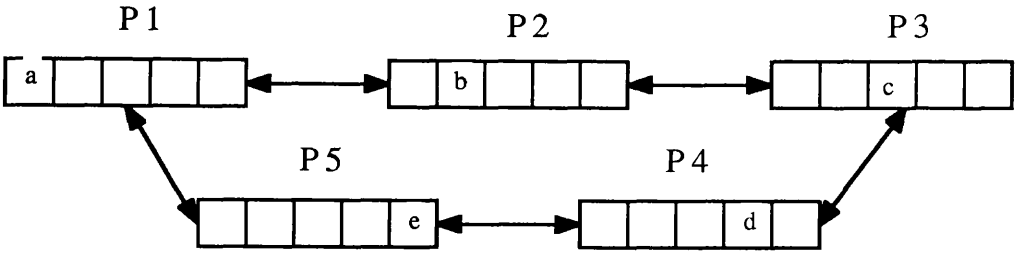
For P_2, n.left and n.right are given by

j	0	1	2
n.left[j]	2	1	5
n.right[j]	2	3	4

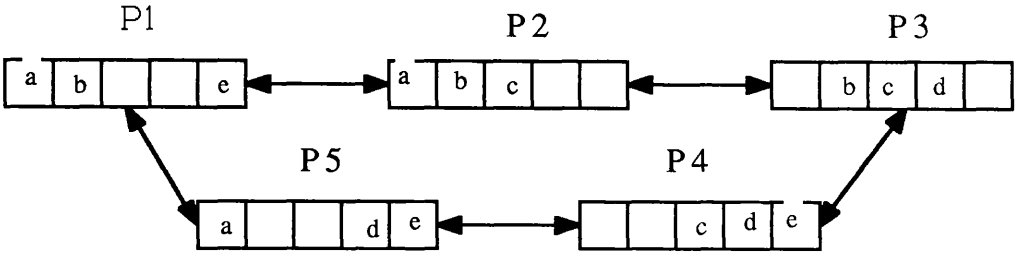
The communication takes two steps to complete. At the first step, P_2 sends y_5, y_6, y_7 to both its neighbours P_1 and P_3, while at the same time it receives y_1, y_2, y_3, y_4 from P_1 and y_8, y_9, y_{10} from P_3. At the second step, it sends all it receives from P_1 in the first step to P_3 and all it receives from P_3 to P_1. It also receives

y_{11}, y_{12}, y_{13} from P_3 and y_{14}, y_{15}, y_{16} from P_1. Details are illustrated in Fig.6.1, in which a,b,c,d,e represent the messages for exchange (For instance, a represents the package of y_1, y_2, y_3, y_4).

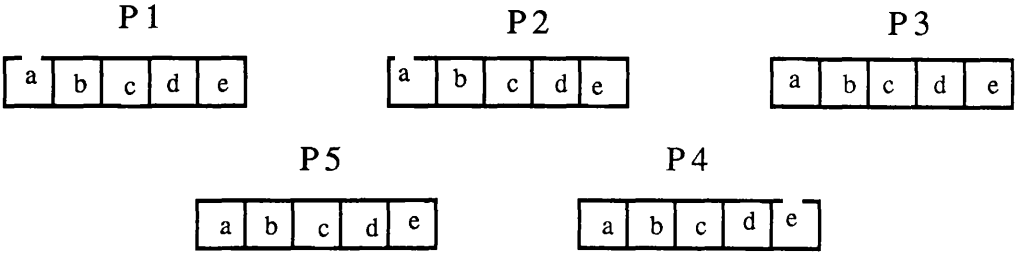
Fig.6.1
Step 1:



Step 2:



Final Results:



In the case of a pipeline structure with p transputers, completed communication can be completed in p-1 steps. The procedure for communication of y for the i^{th} processor P_i is:

```
PROC COM([]REAL64 Y, CHAN from.left, to.left, from.right, to.right)
  INT  j, left.in.c, left.out.c, right.in.c ,left.in.c:
  SEQ
    left.in.c:=left.in.no
    left.out.c:=left.out.no
    right.in.c:=right.in.no
    right.out.c:=right.out.no
    PAR j=0 FOR p-1
      IF
        left.out.c<>0
          SEQ
            to.left  ! [Y FROM S[i+j] FOR L[i+j]]
            left.out.c:=left.out.c-1
        TRUE
          SKIP
      IF
        left.in.c<>0
          SEQ
            from.left  ! [Y FROM S[(i-j)-1] FOR L[(i-j)-1]]
            left.in.c:=left.in.c-1
        TRUE
          SKIP
      IF
        right.in.c<>0
          SEQ
            from.right  ! [Y FROM S[(i+j)+1] FOR L[(i+j)+1]]
            right.in.c:=right.in.c-1
        TRUE
          SKIP
      IF
        right.out.c<>0
          SEQ
            to.right  ! [Y FROM S[i-j] FOR L[i-j]]
            right.out.c:=right.out.c-1
        TRUE
          SKIP

:
```

In the procedure, the four parameters left.in.no, left.out.no, right.in.no, right.out.no indicate the required numbers of communication steps for the by the corresponding channels. For the p^{th} processor, P_i, they can be calculated by

left.in.no = i-1,

right.in.no = p-1,

$$\text{left.out.no} = \begin{cases} 0 & \text{for } i=1 \\ \text{right.in.no}+1 & \text{else} \end{cases}$$

$$\text{right.out.no} = \begin{cases} 0 & \text{for } i=p \\ \text{left.in.no}+1 & \text{else} \end{cases}$$

For the same example of p=5, N=16, the complete communication requires 4 steps. Fig6.2 illustrates the whole process.

Take the example of the second processor P_2.

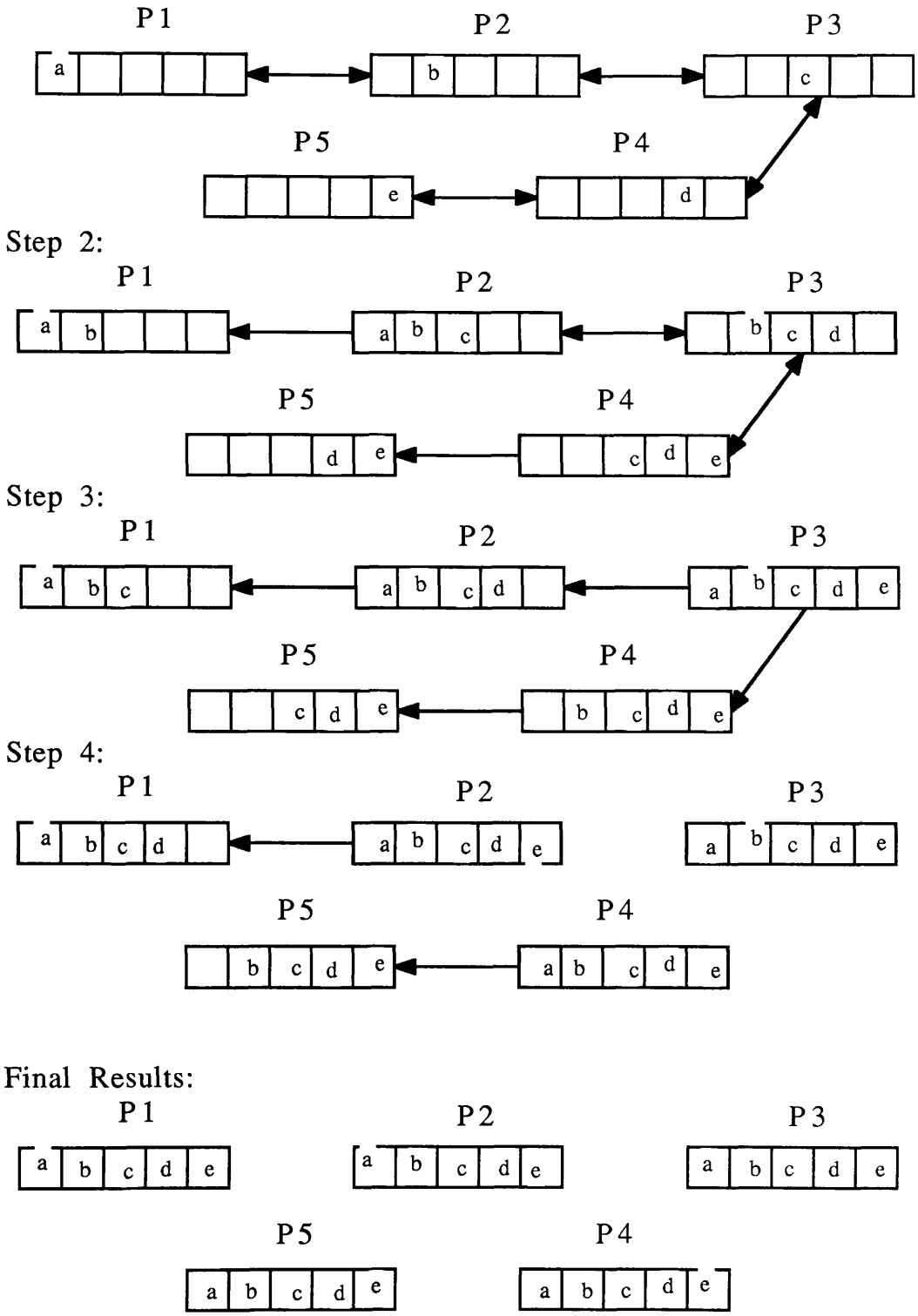
Step 1: P_2 sends y_5, y_6, y_7 to both its neighbours P_1 and P_3, while at the same time it receives y_1, y_2, y_3, y_4 from P_1 and y_8, y_9, y_{10} from P_3.

Step 2: P_2 sends y_1, y_2, y_3, y_4 to P_3 and y_8, y_9, y_{10} to P_1 while it receives y_{11}, y_{12}, y_{13} from P_3.

Step 3: P_2 sends y_{11}, y_{12}, y_{13} to P_1 and it receives y_{14}, y_{15}, y_{16} from P_3. After this step, P_2 has received all y, but it still has to transfer y_{14}, y_{15}, y_{16} for P_1.

Step 4: P_2 sends y_{14}, y_{15}, y_{16} to P_1.

Fig.6.2


Step 1:
P1
P2
P3
a
b
c
P5
P4
e
d
Step 2:
P1
P2
P3
a b
a b c
b c d
P5
P4
d e
c d e
Step 3:
P1
P2
P3
a b c
a b c d
a b c d e
P5
P4
c d e
b c d e
Step 4:
P1
P2
P3
a b c d
a b c d e
a b c d e
P5
P4
b c d e
a b c d e
Final Results:
P1
P2
P3
a b c d e
a b c d e
a b c d e
P5
P4
a b c d e
a b c d e

6.4 Implementation

The parallel algorithms of both the Iterative Numerov method and the De Vogelaere's method are implemented on a network of 5 transputers(T414). The test example comes from the coupled equations which arise from the rotational excitation of a diatomic molecule by neutral particle impact(Allison,1970). Using the notation of the literature(Arthurs and Dalgarno,1960), denoting the entrance channel by the quantum numbers(j, l), the exit channels by (j', l'), and the total angular momentum by J=j+l=j'+l', the equation can be written by

$$\{ \frac{d^2}{dr^2} + k^2_{j'j} - \frac{l'(l'+1)}{r^2} \} y^{Jjl}_{j'l'}(r) = \sum_{j''} \sum_{l''} \langle j'l';J|V|j''l'';J\rangle y^{Jjl}_{j''l''}(r) \qquad (6.26)$$

where

$$k^2_{j'j} = \frac{2\mu}{h^2}\left[E + \frac{h^2}{2I}\{j(j+1) - j'(j'+1)\}\right],$$

The coupling matrix element are given by

$$\langle j'l';J|V|j''l'';J\rangle = \delta_{j'j''}\delta_{l'l''}V_0(r) + f_2(j'l', j''l''; J)V_2(r)$$

where the formulae for the coefficients can be found in the literature(Arthurs and Dalgarno,1960). The scattering matrix S for the problem can be obtained from the R matrix by the relation

$$S=(I+iR)(I-iR)^{-1}$$

We choose the physical parameters and numerical parameters from the literature (Allison,1970). The parameters are

$$\frac{2\mu}{h^2} = 1000.0, \qquad \frac{\mu}{I} = 2.351, \qquad E = 1.12,$$

and $V_0(r)=r^{-12} - 2r^{-6}$, $V_2(r)=0.2283V_0(r)$.

We take J=6 and consider excitation of the rotor from the j=0 state to levels up j'=2,4,6 and 8 giving rise to sets of 4,9,16 and 25 coupled equations, respectively.

The range of integration was chosen to be

r 0	0.75
100 steps at 0.007	0.7
350 steps at 0.014	4.9
Final matching point	6.35

The OCCAM2 programs are mainly translated from Allison's Fortran program. Only the integration sections, which take a large proportion of time, are implemented in parallel.

For comparison, we choose two different convergence parameters $\epsilon=1$ and $\epsilon=10^{-4}$ for iterative Numerov method. $\epsilon=1$ means that we required only one correction at each step. The matrices $|S|^2$ for N=4 with the two convergence parameters are shown in Table 6.1 and Table 6.2. The De Vogelaere's method does not require any iterations, the matrix $|S|^2$ for N=4 calculated from it is given in Table 6.3.

The times, in seconds, required by the iterative Numerov method and the De Vogelaere's method to calculate the square of the modulus of the S matrix are shown in Table 6.4 through Table 6.7 where N is the size of set and p the number of processor. The computation can be divided into four phases, they are: preparation the elements of coupling matrix and potential at mesh points, integrating to matching points, calculating the values of Bessel functions at matching points, the final calculation of S matrix. We

have distributed the computation of the integration which takes the majority of total computing times. Though the computation of the other phases can be done in parallel, we do not do so since little improvement is expected. The sum of the times spent on the there phases is listed as 'ELSE'. Table 6.4 and Table 6.5 are due to Numerov method with loop structure and pipeline structure, respectively. Table6.6 and Table 6.7 show the Similar results for the De Vogelare's method.

Since the loop structure requires less steps in communication than the pipeline structure, it should be more efficient than the latter. But the difference is very small. The efficiency for both structures can be very high. For example, using the De Vogelaere's method on pipeline structure for N=25, p=5, the efficiency reaches 0.99 in the integration phase while the overall efficiency is 0.96.

The most important factor affecting the efficiency is the load balance. Take the De Vogelaere's method on pipeline structure as the example. For N=16, the computation on 5 transputers take as long as on 4 transputers. When p=4, each transputer handles the same number of equations ,i.e. 4 equations. When p=5, one transputer handles 4 equations and the other handle 3. Implementation, in this case, requires the same time on 5 transputers as on 4 transputers. That is why some figures look unusual.

Table 6.1

$|S|^2$ Calculated by iterative Numerov method for N=4 with ε=1.0 (i.e. one correction)

		Values of j and l			
j'	l'	0 6	2 4	2 6	2 8
0	6	0.4133	0.1890	0.1518	0.2460
2	4	0.1891	0.6630	0.1140	0.0340
2	6	0.1517	0.1139	0.6740	0.0599
2	8	0.2459	0.0340	0.0599	0.6601

Table 6.2

$|S|^2$ Calculated by iterative Numerov method for N=4 with $\varepsilon=10^{-4}$

		Values of j and l			
j'	l'	0 6	2 4	2 6	2 8
0	6	0.4134	0.1890	0.1517	0.2460
2	4	0.1890	0.6631	0.1140	0.0340
2	6	0.1516	0.1139	0.6744	0.0599
2	8	0.2459	0.0340	0.0599	0.6601

Table 6.3

$|S|^2$ Calculated by De Vogelaere's method for N=4

		Values of j and l			
j'	l'	0 6	2 4	2 6	2 8
0	6	0.4136	0.1888	0.1516	0.2457
2	4	0.1890	0.6632	0.1139	0.0339
2	6	0.1517	0.1140	0.6745	0.0599
2	8	0.2460	0.0340	0.0600	0.6602

Table 6.4

Times, in second, to calculate $|S|^2$, iterative Numerov method for integration. processors are connected in **loop** structure. Upper entries $\varepsilon=10^{-4}$, lower entries $\varepsilon=1.0$.

	INTEGRATION			ELSE
N\p	1	3	5	
4	26.203	13.625	7.768	1.310
	22.359	11.562	6.526	
9	173.547	60.992	41.268	2.765
	138.190	47.840	32.399	
16	716.800	278.396	186.639	8.525
	530.113	202.686	135.467	
25	2115.068	827.251	457.800	25.425
	1552.157	569.106	314.677	

Table 6.5

Times, in second, to calculate $|S|^2$, iterative Numerov method for integration. processors are connected in **pipeline** structure. Upper entries $\varepsilon=10^{-4}$, lower entries $\varepsilon=1.0$.

	INTEGRATION					ELSE
N\p	1	2	3	4	5	
4	26.203	13.505	13.773	7.686	7.996	1.310
	22.359	11.460	11.671	6.445	6.669	
9	173.547	98.529	61.443	61.972	41.878	2.765
	138.190	70.901	48.200	48.598	32.815	
16	716.800	374.182	279.385	187.266	188.125	8.525
	530.113	273.031	203.338	135.946	136.599	
25	2115.068	1185.535	828.684	648.489	461.006	25.425
	1552.157	822.941	570.040	445.160	316.654	

Table 6.6

Times, in second, to calculate $|S|^2$, De Vogelaere's method for integration. processors are connected in **loop** structure.

	INTEGRATION			ELSE
N\p	1	3	5	
4	37.477	19.177	10.504	2.186
9	239.097	82.688	55.244	3.641
16	938.468	358.698	238.912	9.401
25	2793.089	1025.405	564.169	26.301

Table 6.7

Times, in second, to calculate $|S|^2$, De Vogelaere's method for integration. processors are connected in **pipeline** structure.

	INTEGRATION					ELSE
N\p	1	2	3	4	5	
4	37.477	19.047	19.274	10.408	10.696	2.186
9	239.097	134.453	83.020	83.438	55.646	3.641
16	938.468	483.779	359.098	239.169	239.849	9.401
25	2793.089	1481.057	1025.098	799.243	566.219	26.301

Chapter 7

Conclusion

Several effective parallel algorithms for some applications arising from Schrodinger equations have been developed and implemented on transputer network. Numerov's method and De Vogelaere's method, from which the parallel algorithms are developed, have been investigated.

Developing parallel algorithms is a interesting and challenging task. It is unlikely that there are universal methods for developing parallel algorithms. Though it may obviously exist for some applications, such as solving a coupled equations, parallelism has to be exploited by some special transformations in most cases.

Parallel algorithms are different from serial ones. Here are some facts discovered in developing parallel algorithms:

1. A serial algorithm may be generally applied to several applications. In contrast, a corresponding parallel algorithm is only suitable for few particular applications. For example, the Numerov algorithm can be used for all three applications while we use matrix formalism for a single equation and communications for coupled equations.

2. Even for one application, there may be several different parallel algorithms, which are developed from the same serial algorithm. Their efficiency may depend on the practical limits, such as the number of processors. To illustrate, in the bound state

problem, methods 1-3 are based on the same serial algorithm. Method 1 is the most efficient, but it is only suitable for a parallel machine with four processors.

3. Some algorithms, which are considered poor and ignored in serial computing, would perhaps be more easily and more efficiently parallelised. The secant method for the bound state problem gives the evidence.

4. We cannot expect that an effective parallel algorithm will be as flexible as the corresponding serial one. For example, we arrange the interval sizes in advance rather than changing them automatically according to the estimated error.

The efficiency of a parallel algorithm can never reach 100 percent. The factors affecting the overall efficiency are

1. Complexity: A parallel algorithm may have greater complexity than a serial one. This decides the upper limit of the parallel algorithm.

2. Distribution: A task can not always be decomposed into subtasks with the same size. The efficiency will be decreased because of the imbalance of distribution.

3. Communication: The cost of communication depends on the physical environment. On a transputer network, the time for a floating-point arithmetic operation is very much longer than the time to pass a real datum between processors. For a small network, the cost of communication is relatively low.

4. Sequential factor: Some operations are strongly sequential. Because of their effects, the efficiency of the algorithm will fall With the increasing of the number of processors.

References

Allison, A. C., *J. Comput. Phys.* **6** (1970) 378.

Allison, A. C., *Advances in Atomic and Molecular Physics*, **25**(1988)

Arthurs, A.M. and Dalgarno, A., *Proc. Roy. Soc. Ser.*, A**256**(1960) 540.

Ahmed, H.S., *Tutorial on Parallel Processing*, edited by R.H. Huhn and D.A. Padua (IEEE, 1981) 412.

Baylis, W.E. and Peel, S.J., *Comp. Phys. Comm.* **25** (1982) 7.

Blatt, J.M., *J. Comput. Phys.* **1** (1967) 382.

Burns, A., "*Programming in Occam 2*," Addison-Wesley, 1988.

Coleman, J. P. and Mohamed, J., *Math. Comp.* **32** (1978) 751.

Coleman, J. P., *Comp. Phys. Comm.* **19** (1980) 185.

De Vogelaere, R., *J. Res. Nat. Bur. Standards*, **55** (1955) 119.

Flynn, M.J., *IEEE Trans. Comptr*, C-**21**, (Sept.1972) 948.

Fox, L. and Mayer, D. F., "*Computing Methods for Scientists and Engineers*," Clarendon, Oxford, 1968.

Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., Walker, D. "*Solving Problems on Concurrent Processors*," Vol. **1**. Prentice-Hall, 1988.

Gautschi, W., *Numer. Math.* **3** (1961) 381.

Gordon, R.G., in "*Methods in computational Physics*," Vol.10, edited by B.Alder et al. Academic Press, New York, 1971.

Hearn, A. C., *Reduce User's Manual*, Rand Publication CP78, 1985.

Henrich, P., "*Discrete Variable Methods in Ordinary Differential Equations*," Wiley, New York, 1962.

Hey, A.J.G., *J. Comput. Phys.* **50** (1988) 23.

INMOS , "*Transputer* ," 1985.

INMOS , "*the Transputer Family*," 1986.

Ishibashi, K., Takada, H., Sakae, T., Matsumoto, Y., Katase, A., *J. Comput. Phys.* **80** (1989) 17.

Ixaru, L. G. and Rizea, M., *Comp. Phys. Comm.* **19** (1980) 23.

Ixaru, L. G. and Berceanu, S., *Comp. Phys. Comm.* **44** (1987) 11.

Ixaru, L. G. and Rizea, M., *J. Comput. Phys.* **73** (1987) 306.

Jesshope, C., "*Major Advances in Parallel Processing*," Gower Technical Press, England, 1987.

Lambert, J. D., "*Computational Methods in Ordinary Differential Equations*," Wiley, New York, 1973.

Lester, W.A., *J. Comput. Phys.* **3** (1968) 322.

Lyche, T., *Numer. Math.* **19** (1972) 65.

Pountain, D., "a Tutorial Instroduction to Occam Program," INMOS Ltd, 1986.

Raptis, A. D., Ph.D. Thesis, Glasgow Univ (1977)

Raptis, A. D. and Allison, A. C., *Comp. Phys. Comm.* **14** (1978) 1.

Raptis, A. D. and Cash, J. R., *Comp. Phys. Comm.* **44** (1987) 95.

Schendel, U. "*Introduction to numerical Methods for Parallel Computers*," Ellis Horwood, Chichester, 1984.

Wallach,Y., "*Lecture Notes in Computer127: Alternating Sequential/Parallel Processing*," Springer-Verlay, New York, 1982.