Quantum wave modeling on highly parallel distributed memory machines

Naresh Nayar
Iowa State University
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Quantum wave modeling on highly parallel distributed memory machines

Nayar, Naresh, Ph.D.

Iowa State University, 1992
Quantum wave modeling on highly parallel distributed memory machines

by

Naresh Nayar

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved: Members of the Committee:

Signature was redacted for privacy. Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1992
DEDICATION

To Sunita, my friend and wife.
TABLE OF CONTENTS

DEDICATION ......................................................... ii

ACKNOWLEDGMENTS ................................................. x

1. INTRODUCTION .................................................. 1

2. PERFORMANCE ISSUES ON DISTRIBUTED MEMORY MA-
   CHINES ............................................................. 5
   2.1 Shared Memory Versus Distributed Memory Machines .... 5
   2.2 SIMD versus MIMD ............................................. 6
       2.2.1 nCUBE and Maspar ..................................... 7
   2.3 Characteristics of Distributed Memory Machines .......... 8
   2.4 Grain Size of Machines and Algorithms .................... 9
   2.5 Communication-Efficient Algorithms ....................... 11

3. A 3-DIMENSIONAL PROBLEM .................................... 12
   3.1 Introduction .................................................. 12
   3.2 Chemistry and Physics of Particle-Particle Interaction .. 12
   3.3 General Parallel Approach .................................. 15
       3.3.1 Matrix and Vector Operations ....................... 15
   3.4 MasPar Version ............................................... 16
3.5 Matrix Decomposition ........................................ 17
   3.5.1 Vector Decomposition .................................. 19
   3.5.2 Matrix-Vector Multiply Algorithm ......................... 20
3.6 nCUBE Version .............................................. 22
   3.6.1 Vector Decomposition and nlocn,i ........................ 23
   3.6.2 Matrix Decomposition .................................. 24
   3.6.3 Matrix-Vector Multiply Algorithm ......................... 24
3.7 Results .................................................... 27
3.8 Conclusion .................................................. 29

4. THEORY OF DISTRIBUTED APPROXIMATING FUNCTIONS
   (DAFs) .......................................................... 31
4.1 Importance of Scattering Calculations .......................... 31
4.2 DAF Theory ................................................ 35
   4.2.1 Interpolation vs. Approximation .......................... 35
   4.2.2 Dynamic Behavior of Hermite Functions ................. 38
   4.2.3 Derivation of DAF Theory ................................ 39
4.3 Banded Nature of the DAF Propagator .......................... 45

5. IMPLEMENTATION ISSUES ...................................... 47
5.1 Generation of Weights ....................................... 47
5.2 Wave Propagation .......................................... 48
   5.2.1 Algorithm for Wave Propagation .......................... 49
5.3 Comparisons with FFT Methods ................................ 50
   5.3.1 Comparison of Operation Counts ......................... 50
   5.3.2 Comparison of Communications Requirements ............... 51
| Table 7.1: | Potential parameters for short and long time propagation | 79 |
| Table 7.2: | Initial packet and propagation parameters for short and long time propagations | 80 |
| Table 7.3: | Comparison of the real and imaginary parts of the wavepacket after 400 time steps for the 1-D double barrier potential | 81 |
| Table 7.4: | Converged reflection and transmission probabilities for electron scattering off the double barrier potential | 82 |
LIST OF FIGURES

Figure 3.1: Decomposition of an 8 x 8 matrix on a 4x4 PE array ....... 18
Figure 3.2: Decomposition of a vector of size 8 on a 4x4 PE array ....... 19
Figure 3.3: Vector consolidation ........................................... 20
Figure 3.4: Broadcast .......................................................... 21
Figure 3.5: Local Dot-Products .............................................. 21
Figure 3.6: Local-sum collapse .............................................. 21
Figure 3.7: Transpose and distribute vector ............................... 22
Figure 3.8: Data decomposition on a 4-node hypercube .................. 25
Figure 3.9: Consolidation of a vector on a 4-node hypercube .......... 27
Figure 3.10: Local dot-products (Only node 0 shown) ................... 28
Figure 3.11: MasPar MP-1 and nCUBE 2 .................................. 28
Figure 3.12: MasPar and nCUBE communications. Fraction of time spent
in communication ............................................................... 29

Figure 4.1: Accuracy of fitting vs. number of Hermites (for different \( \sigma(0) \)) 42
Figure 4.2: Accuracy of fitting vs. number of Hermites (for different func-
tions) ................................................................. 43
Figure 4.3: A DAF ............................................................... 44
Figure 4.4: Graphical Representation of Equation (1) ...................... 45
Figure 6.1: Shape Polynomial for $\sigma(0) = 0.98$ and $M = 8$ 65
Figure 6.2: DAF for $\sigma(0) = 2.36$ and $M = 54$ 66
Figure 6.3: Optimum value of $\sigma(0)$ versus $M$ 67
Figure 6.4: Gaussian envelope, the shape polynomial and their product, the DAF (for $\tau = 0$) 68
Figure 6.5: The Gaussian envelope and the modulus of the DAF at times $\tau = 0$ and $\tau = 0.25$ 69
Figure 6.6: The Gaussian envelope and the real part of the oscillatory factor of the DAF 70
Figure 6.7: The real part of the exact free propagator and the real part of the DAF (M=6) 71
Figure 6.8: The real part of the exact free propagator and the real part of the DAF (M=12) 72
Figure 7.1: Comparison of FFT and DAF methods for a double barrier + linear potential (real part) 83
Figure 7.2: Comparison of FFT and DAF methods for a double barrier + linear potential (absolute value) 84
Figure 7.3: Comparison of FFT and DAF methods for a double barrier + cosine time-dependent potential (real part) 86
Figure 7.4: Comparison of FFT and DAF methods for a double barrier + cosine time-dependent potential (absolute value) 87
Figure 7.5: Comparison of FFT and DAF methods for a double barrier + linear + cosine time-independent potential (real part) 88
Figure 7.6: Comparison of FFT and DAF methods for a double barrier + linear + cosine time-independent potential (absolute value) . 89
ACKNOWLEDGMENTS

This work was performed in part at Ames laboratory under contract number W-7405-eng-82 with the U.S Department of Energy. The United States government has assigned the DOE Report number IS-T-1599 to this thesis.

I would like to thank my advisor, Dr. G. M. Prabhu, for his guidance, and patience, with emphasis on the word patience. The super-bowl parties were fun.

I am thankful that Dr. John L. Gustafson came to Ames Laboratory and helped to start the parallel-processing effort at Iowa State University. He took me on as a Research Assistant at the laboratory, so that I could concentrate on research. He provided valuable assistance in research, and help in preparing transparencies and presentations.

Dr. David K. Hoffman was the one who presented the thesis problem, and with whom I worked closely. He very patiently presented and explained the issues to me, and forced me to organize my work. He presented many new ideas during the course of our research.

A special thanks to Dr. Donald Kouri, and his students, Xin-Ma, and Omar Sharafeddin, from the University of Houston, for providing us with data that helped to validate our research.

I would like to thank the staff at Ames Laboratory and Computer Science De-
partment for their assistance. A special thanks to my officemate Michael B. Carter
for answering all my questions on Unix. It was fun working with him on a couple
of projects at the laboratory. Mark Arnold provided a great deal of insight on the
problem and the solution with his knowledge of applied mathematics.

Finally, I thank my parents, Bimla and Sita Ram Nayar, and my wife, Sunita
Nayar, for their love, support, and encouragement.
1. INTRODUCTION

A computer is a man-made device that has been developed to solve certain classes of problems. As the problems have become larger and larger, computers have been refined, gaining continuously in speed and data storage capabilities. Unfortunately, there appears to be no realistic approach to substantially increasing the performance of individual computers in the future; technology is already nearing limits set by the speed of light and quantum physics effects. There is general agreement that the only route to significantly increased performance is through parallel processing - the use of many computers together to solve the same problem [9].

Parallel computers have found major applications in almost all scientific and engineering disciplines, as well as computer science. Fox [9] provides a list of fields in which high performance computing is of particular importance. To solve problems on such computers, we have to use parallel algorithms. Sometimes the parallel algorithm is a straightforward adaptation of the sequential algorithm, many times it is a different approach to the solution of the problem. The approach one uses depends upon the problem and the architecture of the machine on which a person is implementing an algorithm to solve the problem.

An interesting area that has received attention is Quantum Scattering and many researchers [27] have proposed algorithms for studying this phenomenon. Algorithms
for studying Quantum Scattering are computation intensive and are suitable for massively parallel machines. The focus of this research is to develop algorithms suitable for parallel processing for studying time-dependent Quantum Scattering.

Scattering processes are at the core of a modern understanding of the physical and chemical processes of matter and its interaction with radiation. In order to probe a system, one brings it into interaction with some other entity, and by appropriate analysis of changes in the probe and/or the system, one can obtain the structure and interactions of the constituents of the system and/or the probe. Furthermore, the most profound description of bulk matter is in terms of the quantum mechanical behavior of microscopic matter combined with statistical ideas. By using the law of large numbers, the properties of bulk matter are revealed as consequences of the properties of atomic, molecular, and sub-atomic particles. Because of the ubiquitous character of scattering or collision processes, the ability to carry out accurate quantum calculations for scattering of complex systems is essential.

The study of Quantum Scattering has important applications in the study of:

1. Electron Tunnelling for making fast “nano-devices”.
2. Biological proton transfer problems.
3. Reactive molecular/surface scattering.
4. Time-dependent potentials.
5. Inelastic and reactive gas-phase molecular scattering.

The focus of this dissertation is the development of efficient parallel methods for studying the phenomenon of time-dependent quantum-wave scattering. The methods
described belong to the class of integral equation methods, which involve the application of a repeated sequence of very short time step propagations. Free propagation of a wavepacket is most easily handled in the so-called momentum representation whereas the effect of the potential is most easily obtained in the coordinate representation. The two representations are Fourier Transforms of each other. The algorithm presented eliminates the computation of FFTs by performing the propagation totally within the coordinate representation. The communication required is only with the nearest neighbors and is load balanced, thus making the algorithm suitable for distributed memory parallel machines.

As described above, the state of the art methods (developed for uni-processors) use FFT methods. FFT methods are communication bound on parallel machines: We will examine the motivation for developing alternate methods in Chapter 2. In Chapter 3, we will describe the reduction of a 3-dimensional wave scattering of two atoms to a one dimensional problem and describe its implementation on two massively parallel machines, an nCUBE and a MasPar using Fourier transform methods. In Chapter 4, starting from the Schrödinger Wave Equation, we will develop formalisms for Distributed Approximating Functions (DAFs), to be used in our scattering calculations for one dimensional problems on the infinite line. Such problems are useful for diagnostic purposes and also provide a simple model for some physical systems. Chapter 5 will provide implementation details for the nCUBE 2. We also present the analysis of the communication and computational complexity of our wave-propagation algorithm. We will compare our results against the running time of FFT methods.

In Chapter 6 of the thesis, we give an alternate derivation of the DAF formalism
and study the important properties of the DAF operator. Finally, in Chapter 7, we demonstrate the computational effectiveness of our methods by applying them to challenging 1-D problems and comparing results against standard, FFT methods. We conclude in Chapter 8.
2. PERFORMANCE ISSUES ON DISTRIBUTED MEMORY MACHINES

In this chapter we give the underlying principles which motivated our research. We give a brief description of high-performance architectures and discuss their performance-related characteristics.

2.1 Shared Memory Versus Distributed Memory Machines

The nature or topology of the interconnection between the processing elements is an important issue in the architecture of parallel machines. An important class of concurrent computers makes use of shared memory. These machines feature a common memory that can be accessed by all the processing elements. The simplest design of this class uses a common bus, or communication channel, to allow the individual processing elements to access the shared memory. This design is particularly appropriate if \( N \), the number of processing elements is small. However, these type of machines are not scalable; neither are algorithms designed for shared-memory architectures. The limiting factor is the bandwidth of the common bus. A more sophisticated shared memory design involves a switch connecting the processing units to the shared memory. The omega network switch has a communication time that increases in proportion to \( \log N \) with increasing system size [18]. Shared memory
machines have an attractive generality, but it can be a nontrivial challenge to arrange the algorithm or hardware to avoid memory "conflicts", or bottlenecks caused by the need for several processors to access a common memory unit [10] simultaneously.

A different class of machines is characterized by a distributed memory. In pure distributed machines, the basic processing element includes local memory to the exclusion of shared or global memory. We are faced with the choice between topologies like the two- or three-dimensional mesh, in which the number of channels per node is independent of N, or alternately, architectures like the hypercube in which the number of channels per node grows logarithmically. Distributed memory machines go hand-in-hand with the message passing model for concurrent computation.

In spite of the aforementioned difficulties in designing algorithms for shared memory machines, it is easier to develop concurrent software for shared memory machines than for distributed memory machines [41]. Further, it is easier to automatically balance the load on shared memory machines. Despite these advantages of shared memory machines, it is our view that distributed memory machines provide a much more cost-effective high performance than do shared memory machines, and we will restrict our discussion to distributed memory machines and performance issues regarding them.

2.2 SIMD versus MIMD

A Distributed memory machine will be of one of the two types:

**SIMD** (Single Instruction Multiple Data) [23]

In this type of architecture all processors are given the same instruction to operate on. Each processor operates on a different datum and depending on
its local state, may “sit out” a sequence of instructions issued from the control unit.

**MIMD** (Multiple Instruction Multiple Data)

In this architecture each processor runs its own instruction sequence. Each processor communicates data to other processors—a processor may have to wait for other processors to access data.

In SIMD architectures, a single control unit fetches and decodes instructions and broadcasts them to a collection of processing elements (PEs). These PEs operate synchronously but their local memories have different contents. Depending on the complexity of the ACU (array control unit), the processing power and addressing methods of the PEs, and the interconnection facilities between the PEs, we can distinguish between pipeline, or vector, processors, array processors, processing ensembles and associative processors.

In MIMD architectures, several processors operate in parallel in an asynchronous manner.

The next section describes the two machines (representatives of SIMD and MIMD architectures) which are available in the Scalable Computation Laboratory (SCL) at Iowa State University.

**2.2.1 nCUBE and Maspar**

SCL is the home of MasPar (SIMD) and nCUBE (MIMD) machines. In its present configuration, the nCUBE is a 128 processor machine connected together as a hypercube. 64 processors have memory modules of 4 Mbytes each, while 64 of them have memory modules of 1 Mbyte each. Node 0 is a “fat” node and has a memory module
of 16 Mbytes. Each individual processor is a 32 bit machine. The front end for the machine is a Sun sparc work-station.

The Maspar is an example of a massively parallel architecture. The present configuration has 16K PEs connected together in a square grid. In addition to the N (North), S (South), E (East), and W (West) connections, the processors are interconnected in the NE, NW, SE and SW directions as well. The communication grid is collectively known as the Xnet. The grid is really a torus, i.e. the grid connections wrap around. The PEs are controlled by the ACU. In addition to the Xnet, PEs can communicate with each other through a global mechanism called the router. The router implements generalized communication—any PE can send a message to any other PE, a PE can receive a message from more than one PE. The router is really a MIMD feature which enhances the power of the machine. The PE itself is a 4 bit CPU with 16K bytes of local memory and an array of registers (16) available for the user. The front end for the machine is a VAX station.

### 2.3 Characteristics of Distributed Memory Machines

Two parameters are of particular importance in the analysis of distributed machine performance.

$t_{calc}$ This is the time required to perform a "typical" calculation. For scientific problems, this can be taken as a floating point calculation. $t_{comm}$ The time required to communicate a single word in the hardware topology.

Let $T_{calc}$ be the total time an algorithm spends in doing computation and $T_{comm}$ be the total time an algorithm spends in doing communication. Then the
fractional communication overhead $f_c$ can be expressed as $T_{\text{comm}}/T_{\text{calc}}$, and is an important measure of the performance of the algorithm. For many algorithms (those with regular communication and computation structures), $f_c$ can be expressed in terms of $t_{\text{calc}}$ and $t_{\text{comm}}$, i.e.

$$f_c = \frac{T_{\text{comm}}}{T_{\text{calc}}} = \frac{\text{constant } t_{\text{comm}}}{g(N) t_{\text{calc}}}$$  \hspace{1cm} (2.1)$$

where $N$ is the problem size and $g$ is some function.

In the design of algorithms for distributed machines, we want $f_c$ to be as small as possible, therefore we want the ratio $t_{\text{comm}}/t_{\text{calc}}$ to be as small as possible. For a given machine, this ratio is fixed and the computation complexity and the communication structure of the algorithm determine the efficiency of an algorithm.

For the two machines in SCL the ratio $t_{\text{comm}}/t_{\text{calc}}$ is very different. Xnet communication on the Maspar is very fast, and since the processing element is a 4 bit unit, $t_{\text{calc}}$ is high. Conversely in the nCUBE each node is a 32 bit processor, leading to a relatively low $t_{\text{calc}}$, but communication latency is high (70 – 100μsec), leading to a high $t_{\text{calc}}$. This would lead one to believe that the MasPar would out-perform the nCUBE, but that is not the case. We still have to examine the function $g(N)$, which embodies the communication pattern, and more important, the data decomposition used in the algorithm. The data decomposition issue leads to discussion of grain size in the next section.

2.4 Grain Size of Machines and Algorithms

The idea of grain size is not central to the thesis, but nevertheless is an interesting topic of research. We outline some of the issues regarding grain size in this section.
Some of our ideas on grain size are outlined in Appendix x.

We could reduce $f_c$ to zero by simply using only one processor to solve the problem, thereby getting rid for any communication. But that would defeat the purpose of using parallel processors. Also, in a distributed machine, a single processor seldom has enough memory to hold data for the entire problem. If it has, the problem should not be solved using a parallel machine anyway.

Once we have determined that a problem is "big" enough to be solved on a parallel machine, we have to determine the data-decomposition for the problem. For example, if we are dealing with a two-dimensional grid, how many points of the grid should be assigned to each processor? In other words what should be the grain size of the algorithm? The choice of grain size of an algorithm is heavily influenced by the grain size of the machine. The MasPar is a small grain-size machine with 16K bytes memory per node while the nCUBE is a large grain-size machine with upward of one Mega bytes of memory per node [39]. There is some confusion about the terms fine grain and coarse grain as applied to parallel computers—the above definition is not the only that appears in the literature. Some authors use the terms to characterize the power of the individual processing elements. Some others use the terms to differentiate between ensembles with a small number of processors and those with a large number.

It is clear a quantitative definition of grain-size would eliminate confusion. Given the process graph, we would like also like to automatically determine the grain size of an algorithm. Some work has been in reported in [36]. In a SIMD architecture, this point is moot because one cannot "save" processors for doing "other" work. Each processor either executes an instruction or sits idle. The problem size has to be big
enough to match the size of the machine on which it is being solved.

2.5 Communication-Efficient Algorithms

For a fixed ensemble and processor size, the speed-up $S$ is given by

$$ S = \frac{N}{1 + f_c} \quad (2.2) $$

where $N$ is the number of processors in the ensemble.

In chapter 4 we will see that the state of the art scattering algorithms use FFTs, which have high communication overheads. The quest for a lower $f_c$, and hence a higher speed-up, provides the motivation for designing algorithms which have efficient communication patterns. In the next chapter, we will introduce the ideas of using DAFs (Distributed Approximating Functions) as a basis for communication efficient algorithms for quantum scattering.
3. A 3-DIMENSIONAL PROBLEM

3.1 Introduction

In this chapter we describe how we reduce a 3-dimensional wave scattering problem for two-atom scattering to a one dimensional problem. We make use of spherical coordinates to take advantage of radial symmetry when two nuclei interact with each other.

Section 3.2 briefly describes the equations involved in computing the interaction of two particles. Sections 3.4 and 3.5 introduce and describe the MP-1 and nCUBE 2 as well as address machine-specific implementation issues. Results from the MP-1 and nCUBE 2, and their respective performance analyses can be found in Section 3.6.

3.2 Chemistry and Physics of Particle-Particle Interaction

We are developing the equations for the quantum dynamics for the collision of two atoms. For the purpose of developing the equations, the velocity of the electrons is so much greater than that of the nuclei of the two atoms that the electrons can be considered only to provide a potential field for the nuclear motion (Born-Oppenheimer approximation). In this way we have finessed out of the problem any detailed consideration of the electron motion. We use center of mass and relative
coordinate vectors. In the absence of external forces, the center of mass dependence can be rigorously separated. This reduces the problem of particle-particle collisions to the problem of a particle with a reduced mass \( \mu = \frac{m_1 m_2}{m_1 + m_2} \) scattering off a central potential. By "central" we mean that the potential is a function only of the radial coordinate, \( R \).

An approximate equation for the short-time propagation of the wave packet, which describes the particle, is \([24]\):

\[
\Psi(t + \tau) = \left(1 + \frac{i \tau H_0}{2\hbar}\right) e^{-i \frac{V \tau}{\hbar}} \left(1 - \frac{i \tau H_0}{2\hbar}\right) \Psi(t)
\]

where:

- \( i \) is the imaginary number \( \sqrt{-1} \)
- \( \hbar \) is Planck's constant
- \( V \) is the potential function
- \( \tau \) is the time step size
- \( H_0 \) is the free particle Hamiltonian
- \( \Psi \) is the wave function
- \( t \) is time

This particular form of the short-time propagator is referred as the "potential-referenced modified Cayley approximation." For a spherical potential, such as we are considering, the angular momentum of the system is conserved, and hence the scattering does not mix angular momentum states. For a fixed angular momentum state the effective free-particle Hamiltonian is given by:
where:

\( \mu \) is the reduced mass

\( L \) is the angular momentum quantum number

The wave function then takes on the form \( \Psi(t) = \Psi_L(t)Y_{L,M} \), where \( Y_{L,M} \) is a spherical harmonic (a function of the polar angles) which specifies the angular momentum state.

Equation 3.1 then reduces to an equation for \( \Psi_L(t) \) which depends only on the radial coordinate variable, \( R \) and the angular momentum number \( L \). Dealing with the angular momentum this way thus reduces the scattering to an effective one-dimensional, time-dependent problem in the radial coordinate \( R \).

After putting equation 3.1 into \( K \) (momentum) and \( R \) (space) representation and discretizing, the form of the propagation equation becomes:

\[
\Psi_L(K_i, t + \tau) = 2 \pi \left( 1 + \frac{\mu \tau K_i^2}{4} \right) \left( \sum_{m=1}^{N} \Delta RR_m j_l(K_i R_m) \right) * \left( e^{-iV(R_m) \frac{\tau}{\hbar}} \left( \sum_{n=1}^{N} \Delta KK_n^2 j_l(K_n R_j) \right) \left( 1 - \frac{\mu \tau K_n^2}{4} \Psi(K_n, t) \right) \right) \tag{3.3}
\]

The spherical Bessel function evaluated on the \( K \) and \( R \) grids is a symmetric matrix, and is used in the spherical Bessel transform. This transform takes the wave function between \( K \) space and \( R \) space. The transform and its inverse are described by the following two expressions:

\[
G(K) = \frac{2}{\pi} \int_{0}^{\infty} R^2 j_l(KR) F(R) dR \tag{3.4}
\]

\[
F(K) = \frac{2}{\pi} \int_{0}^{\infty} K^2 j_l(KR) G(R) dK \tag{3.5}
\]
The corresponding discrete forms of equations and are:

\[
G(K_i) = \frac{2}{\pi} \sum_{j=1}^{N} \Delta RR_j^2 j_{l}(K_iR_j)F(R_j)
\]  
(3.6)

\[
F(R_i) = \sum_{j=1}^{N} \Delta KK_j^2 j_{l}(K_jR_i)G(K_j)
\]  
(3.7)

Equations (5a) and (5b) form the kernel of the implementation for this application. Computational issues for the nCUBE and MasPar versions are discussed in sections 3.3, 3.4 and 3.5.

### 3.3 General Parallel Approach

#### 3.3.1 Matrix and Vector Operations

At each time-step, the spherical Bessel transform is used four times as described by equations 3.6 and 3.7. The equations reveal that the transform is a matrix-vector product, which runs well on virtually all scientific computers. The transform is fine-grained, highly parallel, load balanced, and vectorizable. In other words, it can be made to run at a high fraction of theoretical peak speed on virtually any scientific computer, serial or parallel.

This control flow for the application is described by the following steps:

1. Set up the transform matrix and initial wave packet.

2. Calculate the wave function and tangent phase-shift for analysis at each time step until wave packets are sufficiently far apart.

3. Perform final analysis (Time-dependent amplitude density).
3.4 MasPar Version

We chose to implement this problem on the MasPar MP-1 first, since considerable expertise already existed for this machine. A strictly top-down approach was used in the software development process which resulted in an extremely clean and well-structured code. As on the more conventional supercomputers, specific vector operations were implemented as stand-alone calls and sequestered into a machine-specific vector library separate from the application code. As we shall see, separation of machine specific routines from machine-independent routines helped enormously in porting to the nCUBE 2.

The MasPar MP-1 is a single-instruction stream multiple-data stream (SIMD), or "data parallel", computer. In short, a single instruction issued by a central control unit and executed on each of the processing elements (PEs) operates on a different datum on each PE. There exist control structures to "turn off" selected PEs to give the programmer more control over which data is operated upon [39].

All PEs are connected in a high-speed toroidal 8-way mesh. To give the reader a feel for the actual speed, a double-precision multiply is ten times more expensive than sending the product to an adjacent PE. Clearly, algorithms must take advantage of this high-speed interconnect to achieve maximum performance. This is in stark contrast to the usual parallel programming stricture of "avoid communication unless absolutely necessary." On the MP-1, it is computation that is to be avoided unless absolutely necessary!

Communications is not an insignificant part of programs on the MP-1, however. Since the MP-1 necessarily has a smaller problem grain size than most other machines, proportionally more communication is needed for a given algorithm. It is a matter
of experience, though, that the fraction of time spent in communication for a given algorithm on the MP-1 is roughly the same as what is seen in the nCUBE or other parallel machines.

On many parallel computers, one can make use of a subset of the processor ensemble. This is especially true of hypercubes and shared-memory machines. Such is not the case on the MP-1. Using a subset of the PE's would imply not only a dynamic rerouting of the communication network, but also the existence of another array control unit which is the antithesis to the very notion of the SIMD architecture.

The MasPar MP-1 configuration currently available at the Ames Lab Scalable Computing Facility is an 16,384 PE machine with 16 Kbytes memory per PE. Peak speed is rated at 556 single-precision MFLOPS (Million Floating-point Operations Per Second), 259 double-precision MFLOPS, and 10.2 BIPS (Billion Instructions Per Second). The 8192 PEs are (logically) arranged in a 128 by 64 grid. Nominal cost/MFLOPS (single-precision) is about $560.

3.5 Matrix Decomposition

As usual on parallel computers, we must involve as many of the processors as possible in computations to achieve maximum performance. This is doubly important on the MP-1. Since all PEs execute the same instruction stream, load balancing is a critical issue.

In this vein, we have chosen to distribute matrix elements to the PEs using scattered decomposition (Figure 3.1). In the left-hand side of Figure 3.1, each box, with its corresponding ordered pair, represents the ij'\textsuperscript{th} matrix element. Each box in the right-hand side represents a single PE which owns the four matrix elements
Figure 3.1: Decomposition of an 8 x 8 matrix on a 4x4 PE array

shown. PEs differ by at most two matrix elements in the storage and computation load they must bear. Formally, matrix element $a_{ij}$ is mapped to PE $(x,y)$ by the following relations:

$$x = j \mod nxproc$$

$$y = i \mod nyproc$$

(3.8)

where $nxproc$ is the number of PEs in a row, and $nyproc$ is the number of PEs in a column (128 and 64, respectively).

The scattered decomposition has a number of desirable properties. First, it evenly distributes the data to be stored among the processors. This even distribution of data usually gives rise to a good load balance. Furthermore, a given row of a matrix is stored entirely on an easily computed row of PEs in the PE array. The same holds true for columns.
Linear vector decomposition

\[
\begin{array}{cccc}
V_1 & V_2 & V_3 & V_4 \\
V_5 & V_6 & V_7 & V_8 \\
\end{array}
\]

\( nxproc=nyproc=4 \)

vector length = 8

Figure 3.2: Decomposition of a vector of size 8 on a 4x4 PE array

3.5.1 Vector Decomposition

All vectors in this application are of length equal to the dimensions of the spherical Bessel transformation matrix. The amount of memory consumed by this matrix per PE is:

\[
M_{sb} = \left\lceil \frac{n}{nxproc} \right\rceil \left\lceil \frac{n}{nyproc} \right\rceil \text{sizeof(double)}
\]

Since the largest possible Msb is 16384 bytes per PE (the maximum available memory per PE), n must be less than 4096. Thus the maximum vector length we could possibly make use of would be 4096, which is less than the number of PEs. In this case we may store one element of each vector on each of the first 4096 PEs in the array. In Figure 3.2, a vector of length 8 is linearly-distributed over a 4 x 4 PE array.

Although this decomposition allows us to perform simple vector operations in one floating-point operation time period, it leaves at least half of the PEs idle. However, as we shall see, the total time spent executing these vector operations is not the dominant operation in the algorithm —matrix-vector multiply is.
3.5.2 Matrix-Vector Multiply Algorithm

As stated earlier, the kernel of this application is the matrix-vector multiply operation. Since the spherical Bessel matrix is symmetric, and is the only matrix in this application, vectors may be thought of as either columns or rows. We choose the row interpretation here so that the C compiler will use contiguous storage.

In order to (pre)multiply a 1 x n row vector by an n x n matrix, we must transpose the vector into a column vector, and compute dot products with each of the n columns of the matrix. Since the spherical Bessel matrix is symmetric, we can interchange the dot-product and vector transposition operations. To perform these operations on the (scattered) matrix and (linearly-distributed) vector stored on the MP-1 PE array, we use the following algorithm:

1. Consolidate vector from linearly-distributed decomposition to a scattered decomposition on a single PE row (Figure 3.3).

2. Broadcast consolidated vector to all PE rows (Fig. 4).

3. Broadcast consolidated vector to all PE rows (Figure 3.4).

4. All PEs perform partial dot-products with the matrix
5. All PEs perform partial dot-products with the matrix (Figure 3.1) and vector elements they have (Figure 3.5).

6. Partial sums on all rows are added up to give a resultant column vector (Figure 3.6). The summation is performed logarithmically, but is not shown in the figure. Only two PEs are shown in the figure.

7. Resultant column vector is transposed and moved into the standard linearly
Figure 3.7: Transpose and distribute vector distributed format (Figure 3.7).

Note that all but step 3 in the above algorithm imply a considerable amount of communication between PEs. Such a communications burden is the price one pays for such a fine-grained problem decomposition.

3.6 nCUBE Version

The nCUBE 2 was the second parallel machine on which we ran the scattering problem. Porting the scattering code to the nCUBE was largely a matter of rewriting the vector and matrix library. In fact, the wave-propagation loop transferred and ran unaltered on the nCUBE 2. Only the machine-dependent code had to be rewritten, not the application-specific code. This observation is perhaps the first step toward debunking the myth that it is categorically difficult to port an application from one parallel architecture to another; especially between different classes of parallel architectures (i.e. SIMD to MIMD).

The nCUBE 2 is a multiple-instruction stream, multiple-data stream (MIMD) parallel computer. Each processing element, or "node", executes its own program asynchronously of the other nodes. Nodes coordinate their activities by sending messages to one another. Nodes are connected by 2.2 MByte/sec bidirectional links into a hypercube configuration. There is a 60 $\mu$sec delay for each message until data
is actually transmitted due to software overhead [35].

The nCUBE 2 system configuration available at the Ames Laboratory is a 64 node system. Node 0 has 4 MBytes of memory; the others, 1 MByte each. Peak speed is rated at 171 single-precision MFLOPS, 130 double-precision MFLOPS, and 640 MIPS (Million Instructions Per Second). Nominal cost per MFLOPS (single-precision) is about $1700.

3.6.1 Vector Decomposition and $nlocn,i$

Here we develop a mathematical construct which will be of considerable value in describing the decomposition of both matrices and vectors on the nCUBE 2. While a scattered decomposition works well on the MP-1, it is not desirable on the MIMD nCUBE where communication is not as efficient as on the MP-1.

Vectors are more easily managed if they are partitioned into nearly equal segments. In cases where the length of a vector is a multiple of the number of processors, the length of a segment is simply the vector length divided by the number of processors. For nonintegral multiple sizes, a slightly modified formula must be used. First, we define a function which gives the total number of elements owned by all processors whose processor numbers are less than that of the processor number in question.

$$cum_{n,i} = \left\lfloor \frac{ni}{nproc} \right\rfloor$$  \hspace{1cm} (3.9)

where:

$n$ is the length of the vector.

$i$ is the processor number in question.

$nproc$ is the total number of processors.
Then, we define our desired function which tells us how many elements a given processor has in terms of \( \text{cumen}_i \).

\[ n\text{loc}_n,i = \text{cumen}_n,i + 1 - \text{cumen}_n,i \]  

(3.10)

We now have an easily evaluated function which yields the number of elements a particular processor possesses even for vector sizes which are nonintegral multiples of the number of processors.

### 3.6.2 Matrix Decomposition

Matrices are decomposed in horizontal strips, or groups of rows. Thus, for an \( n \times n \) matrix, processor \( i \) will store rows \( \text{cumen}_n,i \rightarrow \text{cumen}_n,i+1-1 \), inclusive. A given processor stores the same rows of a matrix as it does elements of a vector (Figure 3.8). Although this decomposition is only efficient for systems with fewer processors than there are rows in the matrix, it is sufficient for this application and our current nCUBE configuration.

### 3.6.3 Matrix-Vector Multiply Algorithm

Recall the vector consolidation step from the matrix-vector multiply algorithm on the MP-1. The vector is quickly and easily consolidated onto a single row of PEs because of the very fast communication links. This operation takes place with the number of communication steps proportional to the vector length. On the MP-1, both the time to initiate a message and the time to send a datum are much faster than the time to perform a floating point operation.
Figure 3.8: Data decomposition on a 4-node hypercube
When we return to the realm of MIMD computers, such communication performance is not generally typical. The nCUBE 2 is no exception. Message startup time alone is over 100 times more expensive than a single floating point operation. Once data actually starts moving, the relative cost figure drops to about four times. Clearly, long messages are much preferred over many short ones on the nCUBE 2.

Now, we return to the task of vector consolidation from a scattered decomposition. Let us accept for the moment that the entire vector must be assembled on every node in the hypercube. We must devise a decomposition which allows us to put the vector together with the minimum number of messages between nodes, and with the least amount of useless data movement. If the vector were distributed in the scattered decomposition, much unshuffling would be necessary to put the elements in proper order once they were consolidated onto a single node. If each node stores contiguous elements of the vector, then no unshuffling will be necessary.

The above discussion solves the problem of unnecessary data movement, but what about number of messages? Hypercubes have been around for a number of years now, and many efficient algorithms have been developed that utilize the hypercube interconnect to its full potential. Of interest to us here is the use of the interconnect to consolidate a distributed vector to a single one on every processor. We shall refer to this operation as vector consolidation. Pairs of nodes exchange data across each hypercube dimension in turn. This scheme will consolidate a vector in a number of communication steps equal to the log2 of the number of processors—the absolute minimum number of messages possible.

Matrix-vector multiplication is less complicated with a stripwise matrix decomposition since a single node contains complete rows of the matrix. Below is the
algorithm used to perform the multiplication:

1. Consolidate vector from segmentwise decomposition onto every node in the hypercube (Figure 3.9). Note that the hypercube collapse is used at this step; the scheme implied by the arrows is for illustration purposes only.

2. Perform complete dot products between consolidated vector and each matrix row that a node owns. Store dot products into result vector (Figure 3.10). Note that the result is already decomposed segmentwise!

3.7 Results

Figure 3.11 shows a measure of the scattering program's performance versus problem size on a per-iteration basis for both the MP-1 and nCUBE 2.

We have measured the percentage of time consumed in communications for each machine in matrix-vector multiply. This will help assess the suitability of the matrix-
Figure 3.10: Local dot-products (Only node 0 shown)

Figure 3.11: MasPar MP-1 and nCUBE 2
Figure 3.12: MasPar and nCUBE communications. Fraction of time spent in communication vector multiply algorithm for the architectures in question. Figure 3.12 shows these curves:

Note that the nCUBE spends a significantly greater fraction of time in communication for low problem sizes because of its expensive message startup time. In spite of this, however, the nCUBE manages to achieve greater overall efficiency that the MP-1 at all problem sizes.

3.8 Conclusion

We have shown that a simple quantum scattering problem can be successfully run on both SIMD and MIMD parallel computers. But, this is no surprise. More important than this is the ease with which the application ported not only from serial code to parallel code, but from one class of parallel architectures to another. We wrote generic driver code that embodied the structure of the algorithm with no ties to computer architecture type of any kind. All machine- and architecture-dependence
was sequestered into simple and efficient vector library calls.

We have described an implementation of the equations which describe the interaction of two atoms. In the next chapter we will a formalism which will allow us to propagate wavepackets in the coordinate representation.
4. THEORY OF DISTRIBUTED APPROXIMATING FUNCTIONS (DAFs)

4.1 Importance of Scattering Calculations

Scattering processes are at the core of a modern understanding of the physical and chemical processes of matter and its interaction with radiation. In order to probe a system, one brings it into interaction with some other entity, and by appropriate analysis of changes in the probe and/or the system, one can obtain the structure and interactions of the constituents of the system and/or the probe. Indeed, even the so-called “stationary” states of a bound system can be viewed as (infinitely) long lived collisions which are manifested as singularities of a scattering matrix (e.g., for an $^1$H atom which consists of an electron interacting with a proton). Furthermore, the most profound description of bulk matter is in terms of the quantum mechanical behavior of microscopic matter combined with statistical ideas. By using the law of large numbers, the properties of bulk matter are revealed as consequences of the properties of atomic, molecular, and sub-atomic particles. Because of the ubiquitous character of scattering or collision processes, the ability to carry out accurate quantum calculations for scattering of complex systems is essential. Recently, great attention has been focused on use of the time dependent Schrodinger equation (TDSE) as a basis for the development of powerful new tools for calculating scattering information [40].
One reason for this is the fact that the TDSE is a first order differential equation in the time variable. This implies that the time evolution of the wavefunction is deterministic in the sense that its state at any later time \( t \) is completely determined by its state at some earlier initial time \( t = 0 \); i.e. it is an initial value problem. Such problems are much easier to deal with than boundary value problem, which the time independent Schrodinger equation (TISE) is. Some of the consequences of the initial value character of the TDSE are

1. Solutions can be obtained for the relatively small subset of possible initial conditions that are required to simulate the typical crossed molecular beam experiment.

2. Algorithms which generate the solutions for a single initial condition will scale at worst as \( N^2 \), where \( N \), is the number of basis functions needed to obtain an accurate solution of the TDSE.

3. It is possible, by use e.g. of Fast Fourier Transforms (FFTs) and/or a rotating coordinate system to devise algorithms which scale much more slowly than \( N^2 \).

4. It is possible to develop algorithms such that the nondiagonal matrix which produces the time evolution of the wavepacket has a highly banded structure. This also leads to much slower scaling of the computational effort to calculate the desired scattering information (i.e. a \( nN \) scaling, where \( n \) is the band width of the propagation matrix.

It is this last idea that we have chosen to pursue, since it should lend itself naturally to algorithms well suited to implementation on a massively parallel computing environment. In solving the TDSE, one usually at some stage, evaluate the
action of the quantum mechanical kinetic energy operation $K$ (or its exponential $\exp(-iK\tau/\hbar)$, where $\tau$ is a time increment). In the coordinate representation, this involves computing spatial derivatives of the wavepacket, and it is commonly done by use of finite differencing. This leads to a highly banded structure, both for $K$ and for $\exp(-iK\tau/\hbar)$. However, in order to attain high accuracy, a very fine grid mesh must be used, so that $N$ is very large, leading to heavy computational demands. An attractive alternative is to evaluate the action of $K$ or $\exp(-iK\tau/\hbar)$ on the wavepacket in the *momentum representation*, where $K$ and $\exp(-iK\tau/\hbar)$ are diagonal matrices. To do this, one transforms the wavepacket from the coordinate to the momentum representation, applies $K$ or $\exp(-iK\tau/\hbar)$ (involving $N$ multiplications), and transforms back to the coordinate representation (in which the potential $V$, or its exponential $\exp(-iV\tau/\hbar)$, is diagonal. If cartesian coordinates are employed, the coordinate-to-momentum and momentum-to-coordinate representation transforms are discretized Fourier transforms, which can be effected by FFTs. This leads to scaling by $2N' \log_2 N'$, where the number of grid points $N'$ is generally much smaller than the $N$ required for finite differencing. The price paid to achieve the logarithmic scaling of the FFTs is that one is no longer deals with a banded propagator matrix. The FFT approach has proven very successful for treating a variety of collision problems.

In our approach [22], we propose to continue to utilize the coordinate representation in order to take advantage of the possibility of banding the matrix representation of the operator $\exp(-iK\tau/\hbar)$. This will enable us to take advantage of massively parallel computer architectures in a relatively straightforward way. However, we abandon the use of finite differencing as the means of obtaining the banded representation
of \( \exp(-iK \tau/\hbar) \). Instead, we shall take advantage of the fact that the action of \( \exp(-iK \tau/\hbar) \) on a Gaussian function, and also on Hermite functions, (i.e., a Gaussian generating function times a Hermite polynomial obtained from the generating function by differentiation) can be obtained analytically. This is the consequence of the fact that for Cartesian coordinates the kinetic energy operator \( K \) commutes with the differential operator that generates the Hermite polynomials as derivatives of a Gaussian function [42]. We then develop a procedure for accurately representing a general wavepacket in terms of discrete approximating functions (DAF's) which are themselves expressed as a sum of Hermite functions.

The coefficients of the DAF's are simply the values of the wavepacket at the discrete grid points. The accurate fitting of a general wavepacket, on a discrete grid of \( N \)-points in terms of the DAF functions is done without requiring that the fit be exact on the grid points. That is, we do not require the fit to be an interpolation formula and thus the \( j \)th DAF, evaluated at the \( j' \)th grid point is not a kronnecker delta. This additional flexibility enables us to require that the value of the wavepacket at the grid-point \( j \) not contribute to the grid point \( j' \) very far away. This implies that the value of the \( j \)th DAF at grid point \( j' \) decreases rapidly with \( j - j' \). These ideas will be discussed more explicitly later in the paper.

The free evolution of the wavepacket by a time step \( \tau \) results from applying \( \exp(-iK \tau/\hbar) \) to the wavepacket at any time \( t \). This evolves the DAF's forward by time \( \tau \), and yields an expression for the wavepacket at time \( t + \tau \) in terms of a sum over DAF's (now at time \( t + \tau \)) times the grid values of the original wavepacket at time \( t \). Thus, the grid values of the time evolved DAF's are, in fact, a matrix representation \( F_{j'j}(\tau) \) of the free evolution operator for time step \( \tau \), with the property that \( F_{j'j}(\tau) \)
decreases rapidly as \(|j' - j|\) increases. This is, of course, simply the condition that the matrix \(F_{j'j}(\tau)\) be highly banded. We shall find that like the FFT procedure, the coordinate grid can be much coarser with the DAF's than is possible with finite differences (so \(N'\) is much smaller than \(N\), but unlike the FFTs, the effort required to apply \(\exp(-iK\tau/\hbar)\) in the discretized coordinate representation is now \(n'N'\), where \(n'\), is the (very small) band width of the DAF-effective discretized free operator.

The chapter is organized as follows. In the next section, we give a brief review of the basic notions of quantum mechanics, both from the time independent and time dependent points of view. In Section 4.2 we give the details of DAF theory. Finally in Section 4.3 we present our conclusions and set the stage for presenting details of implementations.

\section*{4.2 DAF Theory}

\subsection*{4.2.1 Interpolation vs. Approximation}

The approach to scattering we pursue in this paper is to discretize the wavepacket on a grid of coordinate points. In so doing we need to be able to approximate the wavepacket everywhere by the values on the discretized set of grid points. The approaches employed generally divide themselves into two classes: namely, \textit{interpolation} and \textit{approximation} methods. In the former, the approximating function is required to reproduce the exact, known value of the true function on the grid points. In contrast, for approximating methods the fit is required to be in some sense uniformly good everywhere (e.g., in a least square sense), but not to exactly fit the true function anywhere. There are, of course, many numerical procedures of each class. The
requirement that an interpolation function exactly reproduce the function on the grid can lead to a highly pathological behavior between grid points (as is, for example, well known for very large order Lagrangian polynomial approximation). This difficulty is typically avoided by piece-wise interpolation, using a relatively few points for each segment, or by Fourier interpolation which effectively uses all the grid point values to approximate the function at any point in the domain. The piece-wise approach is inadequate for our purposes because it does not yield an analytic approximation. As we shall see, Fourier interpolation is also not very satisfactory since the approximation is not localized. For these reasons, the numerical procedure we develop is based on an approximation method.

For the purpose of our present discussion we will consider the problem of a particle scattering off a potential field on the infinite line. Such a system is useful diagnostically, and can also be useful as a simple model for certain physical scattering systems (as for example electron-scattering in nano-structure devices).

Typically the numerical solution of the scattering problem in the coordinate representation involves computing the wave function on a grid of points, which are spaced so that the value of the function at points not on the grid can be obtained by interpolation (e.g. by Fourier Series), or by approximation (e.g. by Gram polynomials). Let us begin by assuming an (infinite) grid of equally spaced points on the infinite line. We wish to develop a "fitting formula" for a wavepacket-type function function $f(x)$ of the form

$$f(x) = \sum_{j=-\infty}^{\infty} a_j(x) f(x_j)$$  

(4.1)
where, for interpolation

\[ a_j(x_k) = \delta_{j,k} \]  \hspace{1cm} (4.2)

on the grid, Eq 4.2 does not hold exactly for approximation methods. We assume \( a_j(x) \) to have translational periodicity so that

\[ a_j(x) = a_0(x - x_j) \]  \hspace{1cm} (4.3)

From elementary considerations the "fitting function" \( a_0(x) \) must be symmetric. Obviously, however, these requirements do not uniquely determine its functional form, even if one specifies the "range of influence" of the grid point value, \( f(x_j) \), in the fitting formula. The explicit form of \( a_0(x) \) depends on the fitting scheme one wishes to employ, which is itself governed by the class of functions to which the wavepackets of interest belong. For example, with only very mild restrictions, an analytic wavepacket can be adequately expressed as a Fourier series (as long as the oscillations don’t become too rapid compared to the grid spacing), in which case

\[ a_j(x) = \sin(\pi(x - x_j))/\pi(x - x_j) \]  \hspace{1cm} (4.4)

However, this is an interpolation formula and as a consequence the range of influence of \( f(x_j) \) falls off very slowly (i.e., as the inverse of the separation distance from the point of interest). However, as we shall now show we can approximate wavepackets to sufficient accuracy with a much narrower "bandwidth" if we abandon the interpolation requirement. Furthermore, this allows us to use approximating functions which have very convenient dynamical properties. To illustrate this we briefly digress to discuss the dynamics of Gaussian wavepackets and Hermite functions.
4.2.2 Dynamic Behavior of Hermite Functions

Under the action of the free propagator $F(\tau) = \exp(-i\tau K/\hbar)$, a Gaussian function with standard deviation $\sigma(0)$ evolves according to the expression $\exp(-(x - x_0)^2/2\sigma^2(\tau))$ where $\sigma^2(\tau) = \sigma^2(0) + i\hbar\tau/m$ and $m$ is the particle mass. This is a special case of the well known analytic result for a freely propagated Gaussian wavepacket [42]. The parameter, $x_0$, is not a function of time because the momentum of the Gaussian function, (considered as a Gaussian wavepacket), is zero.

The Hermite polynomials, which form a complete set of orthogonal polynomials under the Gaussian weight function, are given in terms of their generating function by [15]

$$\exp(-x^2)H_n(x) = (-1)^n(d^n/dx^n)\exp(-x^2), 0 \leq n < \infty \quad (4.5)$$

We term a Hermite polynomial multiplied by its Gaussian generator a Hermite function. It follows immediately from Eq (4.5) that Hermite functions can also be propagated analytically, since the $n$th derivative with respect to $x$ commutes with $F(\tau)$. That is, if $\psi(x,\tau) = \psi(x(\sigma(2^{1/2}\tau)))$, then

$$F(\tau)\exp(-y^2(x, 0))H_n(y(x, 0)) = (-1)^n(d^n/dy(x, 0)^n)F(\tau)\exp(-y^2(x, 0))$$

$$= [\sigma(0)/\sigma(\tau)]^n\exp(-y^2(x, \tau))H_n(y(x, \tau)) \quad (4.6)$$

Thus, if we write the function $a_0(x)$ in terms of Hermite functions given by Eq 4.5, (see [22]), we find that it propagates according to

$$a_0(x, \tau) = \exp(-y^2(x, \tau))\Sigma_{n=0}b_{2n}[\sigma(0)/\sigma(\tau)]^{2n}H_{2n}(y(x, \tau)) \quad (4.7)$$
where \( a_0(x,0) = a_0(x) \) and the coefficients, \( b_{2n} \), are constants. Only even Hermite polynomials appear because \( a_0(x) \) is symmetric. Equations (4.1) and (4.7) together then give an expression for the propagated function.

### 4.2.3 Derivation of DAF Theory

We wish to derive a form for \( a_0(x) \) which is spatially confined to the greatest extent possible, so as to obtain a minimum bandwidth for the propagator. We will now show that this is accomplished if \( a_0(x) \) is expressed using Hermite functions. The \( a_0(x) \) so expressed we term Distributed Approximating Functions (DAF's). There is a whole family of DAF approximations which differ from each other in terms of the width of the Gaussian generator relative to the grid spacing. In general, one can use narrow DAF’s, with fewer Hermite functions, for wavepackets with low, in contrast to high, kinetic energy. The choice of DAF’s thus depends on the packet one wishes to propagate. We first briefly discuss the original DAF derivation which provides useful clarification and then present an alternative derivation.

If we require our fitting function to exactly fit all polynomials \( 0 \leq n \leq m \), then according to Eq 4.1 and Eq 4.3 we must have

\[
x^n = \sum_{j=-\infty}^{\infty} a_0(x-j)j^n, \quad 0 \leq n \leq m
\]

where for notational convenience we have assumed that the problem has been scaled so that the grid points fall on the integers. After some manipulation this set of equations can be reduced to

\[
\delta_{n,0} = \sum_{j=-\infty}^{\infty} a_0(j+x)(j+x)^n, \quad 0 \leq n \leq m.
\]
If we choose to write these equations in terms of Hermite polynomials rather than powers of \((j + x)\), they can be expressed in the form

\[
h_0^{(n)} = \sum_{j=-\infty}^{\infty} a_0(j + x) H_n(y(j + x, 0)), \quad 0 \leq n \leq m \tag{4.10}
\]

where

\[
h_0^{(n)} = (-1)^{n/2} \frac{n!}{(n/2)!} \quad n \text{ even}
\]

\[
= 0 \quad n \text{ odd}
\]

is the coefficient of the constant term in the nth Hermite polynomial. We now write \(a_0(x)\) in the form

\[
a_0(x) = \sum_{n=0}^{m} b_n(x) \exp(-y^2(x, 0)) H_n(y(x, 0)) \tag{4.12}
\]

where the sum is over a finite number of Hermite functions and \(b_n(x)\) is a periodic function of \(x\) which is symmetric for even \(n\) and antisymmetric for odd \(n\). Substituting Eq 4.12 into Eq 4.10 and making use of the periodicity of \(b_n'(x)\), we obtain the set of linear equations

\[
h_0^{(n)} = \sum_{n'=0}^{m} C_{n,n'}(j + x) b_{n'}'(x) \tag{4.13}
\]

where the (symmetric) matrix element, \(C_{n,n'}(j + x)\), is given by

\[
C_{n,n'}(j + x) = \sum_{j=-\infty}^{\infty} \exp(-y^2(j + x, 0)) H_n(y(j + x, 0)) H_{n'}(y(j + x, 0)) \tag{4.14}
\]

Equation (4.13) uniquely determines the function, \(b_n'(x)\). To complete the evaluation of the \(b_n\) coefficients in 4.7, we could proceed at this point by re-expanding the function \(b_n'(x) H_n(y(j - x, 0))\) in even Hermite polynomials.
However, if \( \sigma(0) \) is sufficiently large and the order of the largest Hermite polynomial required is not too high (i.e., \( m \) is small enough), the sum over \( j \) in Eq (4.14) can be replaced by an integral in which case the functions, \( b'_n \), become constants that are trivial to evaluate because of the orthogonality of the Hermite polynomials. For odd \( n \) the constants are zero and for even \( n \) they can be identified with the constants of Eq (4.11).

\[
b'_n = b_n = h_0^{(n)}/(2^n n! \sqrt{\pi}) \text{ even}
\]

From Eq (4.15) and Eq (4.7) we finally obtain the explicit result

\[
a_0(x) = \frac{\Delta x}{\sigma(0)} \exp(-y(x, 0)^2) \sum_{n=0}^{m/2} (-\frac{1}{4})^n H_{2n}(y(x, 0))/(2\pi n!)^{1/2}
\]

where \( \Delta x \) is the grid spacing (which for convenience we have taken to be unity).

The conditions for this approximation can always be met by choosing a sufficiently small grid spacing; as a practical matter they turn out not to be stringent.

A consequence of the approximation employed in going from the sum to an integral in evaluating the \( b_n \) coefficients (i.e. in going from Eq(4.14) to the DAF expression of Eq(4.16)) is that the DAF expression is asymptotic in \( m \), the number of Hermite functions employed. This is illustrated in Figure 4.1, where we examine the fit to the real part of a typical Gaussian wave at an arbitrary point not on the grid. (The qualitative behavior is independent of the choice of the point of evaluation.) On the ordinate is listed the number of Hermite functions used in the expansion and on the abscissa, the number of significant figures of the fit as measured by \( \log \left| \frac{f - f_{\text{app}}}{f} \right| \). First, it is somewhat remarkable to note that this is such a systematically varying quantity. Second, the asymptotic nature of the expansion is quite clear. By increasing the width of the Gaussian generator, it is possible to fit
Figure 4.1: Accuracy of fitting vs. number of Hermites (for different $\sigma(0)$)
an analytic function to arbitrary accuracy. The reason is that the "sum-to-integral" approximation becomes increasingly accurate as the DAF expands relative to the grid spacing. However, this high accuracy is offset by the fact that more Hermite functions are needed and an increased Gaussian envelope means a bigger bandwidth. Clearly, one wants to use the narrowest Gaussian generator that provides sufficient accuracy for the problem at hand. To illustrate this point, in Figure 4.2 we show how the quality of the fit varies as we change the momentum of the Gaussian packet. As is to be expected, for fixed width of the Gaussian generator the quality of the fit drops dramatically with increased momentum of the packet. (Although the number of significant figures for all cases given in the figure is more than ample for a scattering calculation).

In Figure 4.3 we graphically display a DAF function centered on the origin. In terms of units of the grid spacing the width of the Gaussian generator is 3.0. We later
discuss the accuracy of the fit that such a DAF provides. There are several features to be noted:

- Because of the Gaussian envelope, the DAF dies away quickly from its maximum.

- The fitting bandwidth in this case is of the order of 20 grid points. (This is typical, but the band can be narrower or broader depending on the nature of the particular packet to be propagated.)

- The DAF is approximately equal to one at the origin and zero at the other grid points, but not exactly so. That this is not rigorously the case is due to the fact that we are using an approximation rather than the interpolation method.

In Figure 4.4 we give a graphical representation of Eq 4.1.
In applying DAF theory to continuous, as opposed to discretized problems grid spacing is not an issue and so the spread of the wave packet is the major consideration in picking the width of the generating function. In discretizing problems one must also be concerned that the DAF which has minimum spreading is sufficiently wide so the DAF representation is adequate to represent the function on the grid.

Finally we reemphasize that the spreading of the propagated DAF is governed by the spread of the Gaussian.

4.3 Banded Nature of the DAF Propagator

As mentioned above, the Gaussian nature of the free propagator causes the propagator to decay as one moves away the origin, causing the operator to be \textit{banded}. This implies that the value of the wave function at time $t + \tau$ can be computed from
the neighboring values of the function at time $t$, which intuitively makes sense; there is no physical reason that in a short time-step $\tau$, the value of the wave at $t + \tau$ at a point should be influenced by the value of the wave function at points “far away” from the point at which we are propagating.

This fact causes the matrix to be a diagonal matrix with the same set of weights appearing in each row, only displaced by one column for every row (with zeros for the other entries.)
5. IMPLEMENTATION ISSUES

As mentioned in Chapter 4, the Gaussian nature of the effective free propagator causes it to decay as one moves away the origin, so the operator is \emph{banded}. This implies that the value of the wave function at time $t + \tau$ can be computed from the neighboring values of the function at time $t$, which intuitively makes sense; there is no physical reason that in a short time-step $\tau$, the value of the wave at $t + \tau$ at a point should be influenced by the value of the wave function at points "far away" from the point at which we are propagating.

The implementation of the above technique is a straightforward two-step process.

- Generation of weights.
- Propagation of the wavepacket.

5.1 Generation of Weights

The complex "weights" or the non-zero entries of each row of the matrix are pre-computed. The weights depend on two parameters:

1. A fitting parameter $\sigma(0)$, the spread of each individual DAF. The $\sigma(0)$ value to be used is determined experimentally. Figure 4.1 gives a plot of accuracy of fitting versus number of Hermites for different values of $\sigma(0)$. Naturally,
one uses the value of $\sigma(0)$ which gives maximum accuracy. The number of Hermite Polynomials to be used for a given value of $\sigma(0)$ can be determined from Figure 4.1. For a given $\sigma$ value, the number of Hermites used for maximum accuracy is *independent* of the function to be fitted.

2. A propagation parameter $\tau$, the duration of the time step. The larger the time-step, the greater the matrix bandwidth required to accurately reproduce the wavepacket at time $t + \tau$.

The DAFs are applicable for *any* function, so the weights can be generated by using the constant functions $(1.0,1.0)$; one has to specify $\sigma$, the time-step $\tau$, and the bandwidth (i.e. the number of weights that need to be generated). For a given $\sigma(0)$, and $\tau$, the bandwidth can be experimentally determined. The weights tend to zero as one approaches the edge of the band. Once the weights are computed, they are stored in a file.

### 5.2 Wave Propagation

We only consider the one-dimensional case here. The two issues to be considered (for any distributed memory machine [9] ) are

- The data decomposition.
- The algorithm for wave propagation.

For the one-dimensional case, the data decomposition is simple. Since we are operating with a one-dimensional grid, we simply look upon the array of processors as a one dimensional grid. The number of points of the grid per processor, are determined
by the size of the grid and the number of processors (or the number of processors available). The nCUBE machine at SCL is a 128 processor machine. For a typical grid size of 512, this works out to be four points per processor. Since the communication time for a message from one node to another is about 100μsec, [39] and the bandwidth of the order of 40 points on either side, it is better to have more points per processor and use fewer processors so that the time spent in communication is minimized. (Each processor gets a message from its neighbor only or from its neighbor and one neighbor across on either side.)

5.2.1 Algorithm for Wave Propagation

The algorithm for the hypercube consists of the following three steps:

**Set-up** Node 0 broadcasts (using the standard broadcast algorithm), weights to all other nodes. Each node calculates the initial wave-function for its assigned grid points and stores the values in an array.

**Wave-propagation loop** For each time-step:

- Depending upon the bandwidth, the function values from the neighboring processors are fetched and stored in a temporary array.
- The new value of the function at each grid point is calculated by multiplying the weights by the value of the function at the grid points (dot-product).
- The function values at the grid points are updated.
- The effect of the potential operator is calculated.

**Accumulation of Results** After the required number of iterations, the final values of the wave function at the grid values are gathered in node 0 (in a logarithmic
number of steps) and the results written out to the front end.

The salient features of the above algorithm are:

- One does not have to go back and forth from momentum to energy space. That saves two FFTs at each time step.

- Communication is nearest neighbor.

- The algorithm is load-balanced.

- In part because of the above two factors, the algorithm is well suited for SIMD machines.

5.3 Comparisons with FFT Methods

We compare the operation count and the communication requirements of the two methods.

5.3.1 Comparison of Operation Counts

The operation count for a complex FFT is $5n \log n$ ($n$ is the size of the input factor). Since we have to do two FFTs per time step, the total number of operations to be performed per time step is $10n \log n$. In the DAF method the number of operations per time step is $8bn$ where $b$ is the bandwidth of the DAF operator.

The operation count of the DAF is less as long as $8b$ is less than $10 \log n$. A typical $b$ value for large 1-D problems is 10–20, and $n$ ranges from 512 to 32768. Hence, the DAF method is superior for large $n$, even without grid-shortening techniques.
5.3.2 Comparison of Communications Requirements

The expression for communication time for an FFT on the hypercube is given by [13]

\[ 2(t_s + t_{comm} n/p) \log p \]  \hspace{1cm} (5.1)

where

- \( t_s \) = start-up time,
- \( t_{comm} \) = communication time for a word,
- \( n \) = length of input vector in words (size of the grid) and
- \( p \) = number of processors.

If we use the FFT method for wavepacket propagation, the above expression will give the communication requirements for each time step. The factor 2 appears because the FFT has to be performed twice for each time step. The expression for communication time for each time step in our DAF algorithm is

\[ 2(t_s + t_{comm} \times b/2) \]  \hspace{1cm} (5.2)

where \( b \) is the bandwidth (in number of grid points).

The factor 2 appears because the communication is with two neighbors. Comparing Eq(5.1) with Eq(5.2) shows that our algorithm is much more communication-efficient. Typically, \( n/p > b/2 \) inside the parentheses, and equation 5.2 has no \( \log p \) factor.

5.4 Conclusions

In this chapter we described our algorithm for a 1-dimensional problem and provided implementation details on a nCUBE. We also compared the operation counts
and the communication requirements of the two methods. In the next chapter we provide a more systematic study of the DAF method.
6. PROPERTIES OF THE DAF OPERATOR

In this chapter we derive the properties of the DAF. As we have shown in the chapter 4, any particular physical wavepacket propagation requires inclusion of energies or momenta up to a finite cut off value. Thus, for a given initial wavepacket and potential, all contributions to the propagator, associated with higher energies or momenta than the cut-off, are irrelevant for its propagation, and the result is elimination of the problematic properties of the exact, coordinate representation, free propagator matrix elements.

However, we have not considered the details of how, e.g., the optimum values of the relevant parameters of the effective are to be chosen. In addition, the roles of the three factors comprising the respective DAF effective propagators has not been fully investigated. These are extremely important issues in order to provide a general, powerful scheme for most efficiently carrying out discretized wavepacket propagations.

This chapter is organized as follows. In section 6.2 we present a new derivation of the DAF zero-time effective propagators. This enables us to analyze how the various parameters in the DAF effective propagators are inter-related, and how sufficiently accurate fits of the initial packet can be constructed. In section 6.2 we extend the derivation to treat the nonzero-time effective free propagators. Finally, conclusions
are given in section 6.3.

6.1 Derivation of the DAF function as the "Identity" for a Class of Functions

The idea is that, whereas the formal operators in the theory must apply to every function in the Hilbert space, for any physical problem conservation principles, and, in particular, conservation of energy, require that only a certain class of functions of the total space will be sampled. We set out to establish a theory which is only valid in the class of functions of the problem, a process we call "pre-filtering." By taking this approach we avoid the problematic properties associated with operators that are used in the whole Hilbert space: constant modulus of the amplitude for the \( x \to x' \) transition, and violent oscillation whose frequency increases with the distance \( |x - x'| \).

To begin we consider a continuous function, \( f(x) \), on the infinite line. It can be formally represented as

\[
f(x) = \int_{-\infty}^{\infty} \delta(x - x') f(x') dx'
\]  

(6.1)

It should be remembered, of course, that although \( \delta(x - x') \) can be formally manipulated in many applications as though it were a well-behaved function, it is not a function at all in the true sense. Because equation 6.1 must fill all functions, the \( \delta \)-function can only have a value at \( x = x' \), and must be normalized to unity.

By contrast, if we have some information about the function \( f(x) \), the \( \delta \)-function in equation 6.1 can be developed by considering it to be the limit of a sequence of analytic functions, \( \delta^{(M)}(x - x') \), which become increasingly spiked in the high \( M \)
limit. That is

\[ \delta(x-x') = \lim_{M \to \infty} \delta^{(M)}(x-x') \]  \hfill (6.2)

There are many ways to do this. For our purposes the Gauss–Hermite representation is most convenient. Thus we write

\[ \delta^{(M)}(x-x') = e^{\nabla \left[ \frac{-(x-x')^2}{2\sigma^2(0)} \right]} \sum_{n=0}^{M/2} \frac{1}{\sigma(0)} (-1)^n (n! \sqrt{2\pi})^{-1} H_{2n} \left( \frac{|x-x'|}{2\sigma^2(0)} \right)^{-1/2} \]

where \( g_M(x-x'|\sigma(0), \tau = 0) \) (defined by this equation) is a polynomial of degree \( M \), which we call the (zero-time) “shape polynomial.” We later discuss a generalized \( (\tau \neq 0) \) version of this polynomial. If the \( \delta \)-function in equation 6.1 is replaced by \( \delta^{(M)}(x-x') \), then this equation is valid for a class of functions discussed below. As \( M \) increases, the shape polynomial serves to make \( \delta^{(M)}(x-x') \) correspondingly increasingly spiked.

The function \( \delta^{(M)}(x-x') \) is dominated by the Gaussian envelope

\[ e^{\nabla \left[ \frac{-(x-x')^2}{2\sigma^2(0)} \right]} \]

which serves to define the effective extent of the function. Thus, we expect \( \delta^{(M)}(x-x') \) will be a suitable approximation for the \( \delta \)-function for any function which can be adequately expressed as a polynomial of degree \( M \) under the DAF envelope. It is obvious that for any analytic function, \( f(x) \), we can make equation 6.1 to be valid to arbitrary accuracy by choosing \( M \) to be sufficiently large. As a general rule, the oscillatory is \( f(x) \), the higher order the polynomial that will be needed to represent it. However, the oscillatory nature of a wavepacket is governed by the momentum distribution of which it is composed, and for any physical system this is controlled by
energy conversation. Thus, at the outset in considering a specific dynamics problem, we can “pre-filter” by choosing an \( M \) which is valid for a given \( \sigma(0) \).

If we substitute the value of \( \delta(M)(x - x') \) in equation 6.1, we get

\[
f(x) \approx \int_{-\infty}^{\infty} \delta(M)(x - x')f(x')dx'
\]  

(6.4)

We can view the above as an approximating equation. In more general cases, we will consider the function we want to propagate as being a function of time according to

\[
f(x, t) = \exp^{-iKt/\hbar}f(x, t = 0)
\]  

(6.5)

where \( \exp^{-iKt/\hbar} \) is the free propagator, and we will want to approximate the function at time \( t \) in terms of the values at an earlier \( t - \tau \).

We are, of course, interested in a discretized version of the problem on a grid. Obviously, it is nonsense to attempt to discretize the integration in equation 6.1. However, this can be done easily for equation 6.4, providing that \( \sigma(0) \) is sufficiently large compared to the grid spacing. Thus equation 6.4 becomes

\[
f(x) = \sum_{j=-\infty}^{\infty} (\Delta x)\delta(M)(x - x_j)f(x_j)
\]  

(6.6)

where \( \Delta x \) is the grid spacing. It is apparent that, in order to develop a satisfactory approximation, it will be necessary for the Gaussian envelope to capture a number of grid points. The degree of accuracy of the approximation depends on how smooth the function is and what is the grid spacing. These questions cannot be answered in the abstract, but rather require some knowledge of the physical problem to be solved so that the appropriate “pre-filtering” can be done. As is to be expected, it is a general rule that accuracy is improved as \( \sigma(0)/\Delta x \) is increased, but for the minimum
computational effort one, of course, wants this quantity to be as small as possible. We have found that this ratio must be approximately in the range of 1 to 5 for the problem we have examined.

The approximation is asymptotic in $M$. That is, increasing the degree of the shape polynomial increases the accuracy of the approximation for a range of $M$, but beyond a certain $M$ value the fit deteriorates. Furthermore, this happens in a very systematic way. To understand the behavior of $\delta(M)(x)$. The zeros of the shape polynomial, which for fixed $M$ are invariant when expressed in units of $\sigma(0)$ as is evident from equation 6.3. They are, of course, symmetrically distributed about the origin (because the shape polynomial is even) and are approximately evenly spaced, with the exception that there is a maximum instead of a node at the origin. This is illustrated in Figure 6.1 for $\sigma(0)/\Delta x = 0.98$ and $M = 8$. In general, there are $M$ real zeros. As $M$ is increased, the zeros, of course, increase in number on the fringe and move toward the origin so that the peak at the origin gets steeper and narrower, as is to be expected. The most accurate approximation occurs when the first zeros of the DAF (symmetrically placed about the origin) are about equal to the grid spacing, as indicated in Figure 6.2 for $\sigma(0)/\Delta x = 2.36$ and $M = 54$. This is reasonable because it implies that the functional value at the origin contributes strongly to the approximate function evaluated at the origin, but contributes very little to the approximate value of the function at other grid points. This behavior is to be compared, for example, with the Fourier interpolation function $\sin(\pi x)/\pi x)$, which is exactly equal to unity at the origin and zero at all other grid points.

As $M$ is increased beyond the optimum value, so that the first zero of the DAF appears between the first grid point and the origin, the accuracy of the approximation
gradually decreases giving rise to the asymptotic character of the approximation in
$M$. The larger the value of $\sigma(0)/\Delta x$, the larger the degree of the polynomial that is
needed to make the spacing of zeros the same as the grid spacing, as is illustrated
by comparing Figure 6.1 and Figure 6.2. In Figure 6.3 a plot of the optimal value
of $M$ as a function of $\sigma(0)/\Delta x$ is given. The accuracy of the fit with increasing
$M$ increases much less steeply for large values of $\sigma(0)/\Delta x$ is given. It is, in fact,
apparent that any analytic function can be fitted to arbitrary accuracy by increasing
the value of $\sigma(0)/\Delta x$ and choosing $M$ optimally, since in this limit the discretized
integral becomes equivalent to equation 6.4. As has been argued, this gives rise to
an arbitrarily accurate approximation for analytic functions [22].

6.2 The DAF Effective Free Propagator

The free propagator is given by

$$F(\tau) = e^{-iK\tau/\hbar}$$  \hspace{1cm} (6.7)

where

$$K = -\frac{\hbar^2}{2m} (d^2/dx^2)$$  \hspace{1cm} (6.8)

is the kinetic energy operator and $\tau$ is the time step for the propagation. Applying
$F(\tau)$ to equation 6.1 we obtain

$$F(M)(x, x' | \tau) = F(\tau)\delta(x - x')$$  \hspace{1cm} (6.9)

as the exact coordinate representation matrix element of the free propagator. (Here
$x$ is considered to be the variable, and $x'$ an index.) The analytic expression for
this exact matrix element is well known and is easily obtained by first writing the
\[ \delta(x - x') = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ik(x-x')} dk \]  

(6.10)

and then making use of the fact that the kinetic energy, and hence the free propagator, is diagonal in this representation. The result is

\[ F(M)(x, x'|\tau) = (m/2\pi i\hbar \tau)^{1/2} \exp\left[\frac{im}{2\hbar \tau}(x - x')^2\right] \]  

(6.11)

The problematic behavior of this coordinate representation matrix element has been explored in chapter 4. By applying \( F(\tau) \) to equation 6.4 we obtain the expression for the effective free propagator

\[ F(M)(x, x'|\tau) = F(\tau)\delta(M)(x - x') \]  

(6.12)

The action of the free propagator on \( \delta(M)(x - x') \) can be obtained analytically. An outline of the procedure for doing this is as follows. First, we note that

\[ H_n(y) \exp(-y^2) = (-1)^n \frac{d^n}{dy^n} \exp(-y^2) \]  

(6.13)

which is the expression for the generator of the Hermite polynomials where

\[ y = (x - x')/[\sigma(0)] \]  

(6.14)

Thus,

\[ \delta(M)(x - x') = \sum_{n=0}^{M/2} (\frac{1}{4})^n (n!\sqrt{2\pi})^{-1} \]  

\[ \frac{d^{2n}}{dy^{2n}} \{ \exp[-(x - x')^2/2\sigma^2(0)] \} \]  

(6.15)
But, from the defining relations of equations 6.7 and 6.8 it is seen that the commutator relation

\[ [F(\tau), \frac{d^n}{d\gamma^n}] = 0 \]  

holds, and thus, by making use of the well-known result,

\[ F(\tau)\{\exp[-(x - x')^2/(2\sigma^2(0))]\} = \frac{\sigma(0)}{\sigma(\tau)\{\exp[-(x - x')^2/(2\sigma^2(\tau))]\}} \]

we have that

\[ F(M)(x, x'|\tau) = \frac{1}{\sigma(\tau)} \sum_{n=0}^{M/2} \left( \frac{1}{4} \right)^n (n!\sqrt{(2\pi)})^{-1} \]

\[ \frac{d^{2n}}{d\gamma^{2n}}\{\exp[-(x - x')^2/(2\sigma^2(\tau))]\} \]

\[ = \frac{1}{\sigma(\tau)}\{\exp[-(x - x')^2/(2\sigma(0)^2)]\} \sum_{n=0}^{M/2} \left( \frac{1}{4} \right)^n (n!\sqrt{(2\pi)})^{-1} \]

\[ \left( \frac{\sigma(0)}{\sigma(\tau)} \right)^{2n} H_{2n}(\sqrt{(x - x')/2\sigma^2(\tau)})^{-1/2} \]

is the complex variance of the Gaussian.

It is important to realize that in the limit \( M \to \infty \), the sequence of effective free propagators converges to \( F(M)(x, x'|\tau) \)(the exact coordinate representation matrix element of the free propagator), regardless of the choice \( \sigma(0) \). In the limit \( \sigma(0) \to 0 \), equation 6.11 is clearly recovered, but this is also the case for any finite \( \sigma(0) \) in the \( M \to \infty \) limit. That is, there is no natural or preferred value of \( \sigma(0) \). The result is simply a manifestation of the fact that the exact expression for the free propagator contains contributions from all momenta. To pursue this point further, we note
from the theory of Fourier Transforms that if $\sigma(0)$ is the variance of the Gaussian in coordinate space, then $\hbar/\sigma(0)$ is the variance of the Gaussian in momentum space. Clearly, it is not desirable to choose a value of $M$ which is larger than that necessary to represent the wavepacket by polynomial of degree $M$ across the spread of the Gaussian in all regions and at all times. This is related to the range of momenta required to express the wavepacket at all times. Consequently, if we choose a large value of $\sigma(0)$, with a narrow momentum spread, $\hbar/\sigma(0)$, we must then correspondingly choose a large enough $M$ value to build a function that is sufficiently oscillatory to capture only the momentum components necessary to build the wavepacket. (As a special case we note that, in the $\sigma(0) \to 0$ limit, all momenta are included in the momentum distribution of the Gaussian envelope, and only the $n = 0$ term is required in the sum of equation 6.19. However, we remove (pre-filter) from the shape polynomial all momentum functions not contained in the wavepacket, since to include such components in the shape polynomial does not improve the initial fit and only serves to force $F^M(x, x'|\tau)$ towards the $M \to \infty$, high-oscillation limit.  

Making use of the fact that 

$$[\sigma^2(\tau)]^{-1} = [\sigma^2(0) + i\hbar \tau / m]^{-1} = [\sigma^2(0) - i\hbar \tau / m]/\sigma^4(0) + (\hbar \tau / m)^2]$$  \hspace{1cm} (6.21)$$

the free DAF propagator of equation 6.19 can be decomposed as a product of three factors according to 

$$F^M(x, x'|\tau) = \exp[-(x - x')^2\sigma^2(0)/(2[\sigma^4(0) + (\hbar \tau / m)^2])] *$$  \hspace{1cm} (6.22)$$

$$\exp[i(x - x')^2\hbar \tau /(2m[\sigma^4(0) + (\hbar \tau / m)^2])]g_M(x - x'|\sigma(0), \tau)$$
where

\[ g_M(x - x'|\sigma(0), \tau) = \frac{1}{\sigma(\tau)} \sum_{n=0}^{n=M/2} (-\frac{1}{4})^n (n!\sqrt{2\pi})^{-1} \cdot (\sigma(0)/\sigma(\tau))^2 n H_{2n}(|x - x'|/[2\sigma^2(\tau)]^{-1/2}) \]  

(6.23)

The polynomial is of degree \( M \) and has complex coefficients; it is called the "shape polynomial" and is the generalization of the zero-time polynomial introduced in equation 6.3. For \( \tau = 0 \), its structure has been previously discussed.

The first term in equation 6.22 provides the bare Gaussian envelope for the DAF propagator; it ultimately controls the asymptotic nature of the propagator matrix element as a function of \( x - x' \). The second factor is oscillatory in form and is very similar to the exponential term in the exact free propagator of equation 6.11. Although this term also becomes highly oscillatory as \( (x - x') \) increases, this behavior is ultimately damped out by the Gaussian. The shape polynomial has the effect of lifting the oscillatory term in the wings. The result is that the DAF propagator spreads out from the Gaussian envelope with increasing \( M \). In the limit as \( M \to \infty \), the result of equation 6.11 is obtained, although for finite \( M \) the matrix element in the wings is damped by the Gaussian. In Figure 6.4 it is shown how, for \( \tau = 0 \), the Gaussian peak is sharpened by the shape polynomial to give the zero-time DAF. This obviously occurs because all the zeros of the shape polynomial lie on the real axis for \( \tau = 0 \). For non-zero \( \tau \), the zeros of the shape polynomial move off of the real axis as a result the DAF modulus is broadened as is illustrated in Figure 6.5. It should be noted in this figure that the DAF at \( \tau = 0 \) is narrower than its Gaussian envelope, whereas the DAF propagator for a time \( \tau \) extends beyonds its envelope.

An interesting relationship exists between the Gaussian and the oscillatory phase
factor stand in fixed relationship. The first wave of the oscillation basically fills the Gaussian envelope, as is illustrated in Figure 6.6. The wave length falls off linearly as a function of \(|x - x'|\) and subsequent oscillations are of shorter wavelength but are ultimately damped by the Gaussian. The effect of the shape polynomial is to lift these outlying oscillations in the wings. Of course, as \(M\) is increased the oscillations in the wings are elevated to resemble the free propagator as illustrated in Figures 6.7 and Figures 6.8.

The accuracy of the initial fitting by DAF expansion is a critical question. This is a question of grid spacing and the choice of \(\sigma(0)\) and \(M\). To achieve satisfactory accuracy with the optimum choice of \(\sigma(0)\) given in [21], it is necessary to choose a grid spacing which is much smaller than would otherwise be necessary. In our experience with certain model problems, \(\sigma(0)\) is three or more times greater than the optimum value required, for the grid spacing we have employed. Thus, in summary, \((\hbar \tau/m)^{1/2}\) is a lower bound for the choice for \(\sigma(0)\), and the actual value is determined by the grid spacing and the accuracy required for the DAF propagator.

6.3 Conclusions

In this chapter we have given a new derivation for the DDAF–effective free propagators which greatly clarifies their properties. In contrast to previous chapters, we have developed DAF grid methods by discretizing the continuum problem. It is shown that if one defines a grid in \(x\) by a \(\Delta x\), then the first zeroes of the zero–time shape polynomial fall on the grid points. This occurs as \(M\) is increased since this systematically moves the smallest zeroes of the shape polynomial toward the origin. The \(M\) zeroes for the polynomial of degree \(M\) are interspersed between the zeroes of
the polynomial for $M + 2$ (recall that the degree of the polynomial is even because the Dirac delta function is even).

We conclude that the DAF procedures should provide rapid matrix propagations of wavepackets (by capitalizing on a highly banded structure of the DAFs).
Figure 6.1: Shape Polynomial for $\sigma(0) = 0.98$ and $M = 8$
Figure 6.2: DAF for $\sigma(0) = 2.36$ and $M = 54$
Figure 6.3: Optimum value of $\sigma(0)$ versus $M$
Figure 6.4: Gaussian envelope, the shape polynomial and their product, the DAF (for $\tau = 0$)
Figure 6.5: The Gaussian envelope and the modulus of the DAF at times $\tau = 0$ and $\tau = 0.25$. 
Figure 6.6: The Gaussian envelope and the real part of the oscillatory factor of the DAF
Figure 6.7: The real part of the exact free propagator and the real part of the DAF (M=6)
Figure 6.8: The real part of the exact free propagator and the real part of the DAF 
(M=12)
7. A COMPUTATIONAL DEMONSTRATION OF THE DAF APPROACH

7.1 Introduction

In this chapter we report computational applications of the newly developed Distributed Approximating Function (DAF) approach to real time quantal wavepacket propagation for several one dimensional model problems. The discussion is along the lines of [37]. The DAF is constructed to fit all wavepackets accurately which can be represented, to the same accuracy, by a polynomial of degree M, or less, within the envelope of the DAF. By expressing the DAF (and thus the wavepacket to be propagated) in terms of Hermite functions (each a product of a Hermite polynomial and its Gaussian generating function), the DAF approximation to the wavepacket is propagated freely and exactly for a short time $\tau$. Combining the DAF class free propagation scheme with any of several short time approximations to the full propagator enables one to propagate the wavepacket through a potential. The DAF results for the propagated wavepacket and various scattering amplitudes are shown to be in good agreement with those obtained by more standard methods.

In previous chapters it has been shown that the action of the free particle propagator on a well defined class of wavepackets can be evaluated, analytically and exactly, by expressing the wavepackets in terms of "distributed approximating functions," or
DAFs. The DAFs are constructed in such a way that

1. They accurately approximate any wavepacket that can be represented, to the same level of accuracy, as an Mth degree polynomial within the DAF envelope.

2. The action of the free particle evolution operator on the DAF can be gotten exactly and analytically.

This class of wavepackets was termed the “DAF-Class” [20]. These properties are guaranteed by expressing the DAF in terms of a new basis comprised of “Hermite functions”, each of these being the product of a Hermite polynomial and its Gaussian generator. These Hermite functions possess the fundamental significance that they are the natural functions for describing the freely evolved wavepacket of localized particles. Thus, they provide the most compact representation possible of the freely evolved DAF class of wavepackets [1,3–5]. This procedure of developing an accurate representation of a class of wavepackets in terms of exactly and analytically propagatable fitting functions, leads to expressions for freely evolving those DAF-class wavepackets. The DAF fits only a restricted class of wavepackets, and the freely evolved DAF fits the freely evolved restricted class of wavepackets. Effectively, the dynamics has been “prefiltered”, so that one is no longer depending on interference to eliminate non-physical high frequency oscillations of the exact free propagator [29, 34]. Both discretized and continuous versions of the DAF have been developed; the former yields a discrete matrix for fitting the discretized DAF-class wavepackets and for propagating them, [22, 21], while the latter is used in the real time Feynman path integral form of wavepacket propagation [22]. The Gaussian generator factor produces a highly banded matrix structure of the DAF-Class free propagator. This,
along with the “translation” property of the DAF (that enables the DAF–Class free propagator matrix to be generated from a single row), leads to a matrix propagation scheme ideally suited to massively parallel computers [22, 21]. The Gaussian generator also plays a key role in the path integral formulation, since it makes possible the evaluation of the real time path integral by Gaussian biased sampling Monte Carlo methods [30]. In a paper [29] which considered general properties of the exact free and full propagators, it was noted that the fitting of the wavepacket must be accurate enough so that repeated application of the potential region of the full propagator will not remove the wavepacket from the DAF–Class. This can be done by appropriately choosing the width of the Gaussian generator of the Hermite polynomials, and the highest degree polynomial contained in the DAF [1,5].

Our previous chapters have focussed on the formal development of the DAF approach, and the basic structure of the DAFs, but did not contain any computational demonstrations of the method. In this chapter, we provide several computational examples showing that the DAF approach can provide accurate results for real time quantum dynamics, even for systems requiring extremely long propagation times. For the examples studied, we have concentrated on the discretized or matrix version of the DAFs. We illustrate the ability of the DAF formation to accurately propagate wavepackets for long times by applying it to the following problems:

1. an electron scattering (in 1-D) off a double square barrier

2. an electron scattering (in 1-D) off a double square barrier, plus a linear potential

3. an electron scattering (in 1-D) off a double square barrier, plus a sinusoidal time dependent potential representing an external radiation field
4. an electron scattering (in 1-D) off a double square barrier, plus a term varying sinusoidally with position and a linear (with position) potential term.

In the next section, we briefly summarize the discretized DAF-expressions (DAF) for the propagated wavepacket, and in Section 7.3 we present the results for the model problems mentioned above. Finally, we present our conclusions in Section 7.4.

7.2 Wavepacket Propagation using DAFs

The propagation schemes which we use in this chapter are the kinetic referenced Modified Cayley [25, 44] (KRMC) and the kinetic referenced symmetric split operator [7, 6] (KRSSO) methods. Both involve the short time free particle evolution operator, \(exp(-iK\tau/\hbar)\), but they differ in how the potential enters. Explicitly, they yield the wavepacket at time \(t\) in terms of its values at time \(t - \tau\), according to [25, 44]

\[
|\Psi(t)\rangle = (1 + \frac{i\tau}{2\hbar} V)^{-1} e^{-iK\tau/\hbar(1 - \frac{i\tau}{2\hbar} V)}|\Psi(t - \tau)\rangle
\] (7.1)

and [7, 6]

\[
|\Psi(t)\rangle = e^{-i\tau V/2\hbar} e^{-iK\tau/\hbar} e^{-i\tau V/2\hbar}|\Psi(t - \tau)\rangle
\] (7.2)

The strategy for calculating the time evolution by either approach is that of utilizing the coordinate representation in order to take advantage of the diagonality of the potential in this basis. As has been discussed in earlier work, the DAF procedure provides a computationally tractable and accurate way of obtaining the result of the free particle evolution operator on a specific class of wavepackets (the DAF-class, consisting of all \(L^2\) packets that can be accurately represented as \(M\)th degree polynomials within the envelope of the DAF). The coordinate representation of the
DAF version of Equations 7.1–7.2 can be expressed as 7.3–7.5.

\[ \Psi(x|t) = Q(x) \int_{-\infty}^{\infty} F^M(x, x'|\tau) Q^*(x')^{-1} \Psi(x'|t - \tau) dx \quad (7.3) \]

where \( Q(x) \) is

\[ Q(x) = (1 + \frac{i\tau}{2\hbar} V(x))^{-1} \quad (7.4) \]

for Equation 7.1 and

\[ Q(x) = e^{-i\tau V(x)/2\hbar} \quad (7.5) \]

for equation 7.2, provided \( V(x) \) is real. \( F^M(x, x'|\tau) \) is the continuous DAF-Class free propagator given by equation 7.3.

\[ F^M(x, x'|\tau) = \frac{1}{\sigma(\tau)\sqrt{2\pi}} e^{-\frac{(x-x')^2}{2\sigma^2(\tau)}} \sum_{n=0}^{M/2} \left( -\frac{1}{4} \right)^n \sigma(0)/\sigma(\tau)^2 n! H_{2n} \left[ \frac{(x-x')/(\sqrt{2}\sigma(\tau))}{\sigma(\tau)} \right] \quad (7.6) \]

with

\[ \sigma^2(\tau) = \sigma^2(0) + i\hbar \tau/m \quad (7.7) \]

and where \( H_{2n} \) is a Hermite polynomial.

The width parameter, \( \sigma(0) \), is chosen so that the \( \tau = 0 \) DAF, \( F^M(x, x'|0) \), can be used to approximate any polynomial of degree \( M \), or less, accurately within the DAF envelope. Then for any wavepacket representable as such a polynomial within the DAF envelope, \( F^M(x, x'|\tau) \) provides an equally accurate time evolved wavepacket.

For an initial Gaussian wavepacket, very accurate results, with a reasonably low \( M \), can be obtained using \( \sigma(0) \), which is about 10–20 times smaller than the initial width of the Gaussian wavepacket. The discretized version of Equation 7.1 is easily obtained.
by introducing a trapezoidal quadrature for the integration over $x'$, yielding [22]

$$\Psi(x_j|t) = Q(x_j)\Delta x \sum_{j'=1}^{J} w_{j'} F^M(x_j, x_{j'}|\tau) Q^*(x_{j'})^{-1} \Psi(x_{j'}|t - \tau)$$

(7.8)

with $w_{j'}$ being the usual trapezoidal rule weight. One typically should choose $\Delta x$ so that

$$\sigma(0)/\Delta(x) \geq 3$$

(7.9)

i.e., one needs at least 3 quadrature points under each DAF.

It is important to note that, due to the Gaussian factor, $\exp[-(x_j - x_{j'})^2/2\sigma^2(\tau)]$, the DAF-Class propagator matrix, $F^M(x_j, x_{j'}|\tau)$, is very highly banded. In addition, it depends only on even powers of the difference $(x_j - x_{j'})$, so that the full matrix can be generated from one row. These are important features for the development of highly efficient wavepacket propagation schemes. (However, our purpose in the present class is to demonstrate that the DAF approach can be used to obtain accurate scattering information, so we do not concern ourselves with developing the most highly optimized codes.)

For the purposes of comparing results, we have also implemented Equations 7.1 and 7.2 by standard methods [7, 6], using the momentum representation for evaluating $\exp(-K\tau/\hbar)$, the coordinate representation for evaluating the $Q$ and $(Q^*)^{-1}$, and the Fast Fourier Transform to go between these representations [7, 26, 28]. We now turn to give the results for the four scattering systems.

### 7.3 Computational Results

Calculations have been carried out for the four model problems in order to verify that the DAF approach is capable of yielding accurate results for real time
Table 7.1: Potential parameters for short and long time propagation

<table>
<thead>
<tr>
<th>Potential Parameters</th>
<th>Short Time Propagation</th>
<th>Long Time Propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Double Barrier</strong></td>
<td>( V(x) = V_d(x) = \begin{cases} V_0 &amp; a &lt;</td>
<td>x</td>
</tr>
<tr>
<td><strong>Double Barrier + Linear</strong></td>
<td>( V(x) = V_d(x) + V_L(x) )</td>
<td>( V_L(x) = eEx )</td>
</tr>
<tr>
<td><strong>Double Barrier + Time Dependent Cosine</strong></td>
<td>( V(x, t) = -V_1 \cos(\frac{\pi(x - ct)}{a}) + V_d(x) )</td>
<td>( V_1 = 8.1985806 \times 10^{-21} \text{Joule}, \ a = 200\text{Å}, C = \text{speed of light} )</td>
</tr>
<tr>
<td><strong>Double Barrier + Linear + Cosine</strong></td>
<td>( V(x) = V_d(x) + V_L(x) - V_1 \cos(\frac{\pi x}{a}) )</td>
<td>( V_1 = 8.1985806 \times 10^{-21} \text{Joule}, \ a = 200\text{Å} )</td>
</tr>
</tbody>
</table>

Quantum dynamics requiring very long times. The potentials and their parameters are summarized in Table 7.1, and the initial packet, grid and propagation parameters are in Table 7.2.

The double square barrier can be thought of as an “anharmonic” potential, and should present a significant challenge, especially for extremely long time propagation. The first problem studied is a 1-D double barrier problem, where the parameters specifying the model are given in Table 7.1, and the propagation parameters are given in Table 7.2. Calculations were first carried out using the DAF and FFT.
Table 7.2: Initial packet and propagation parameters for short and long time propagations

<table>
<thead>
<tr>
<th>Initial Gaussian Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short time propagations</strong></td>
</tr>
<tr>
<td>Potential (a)</td>
</tr>
<tr>
<td>( X_{min} = -1500 \text{Å}, \ X_{max} = 1500 \text{Å}, \ NX = 405, )</td>
</tr>
<tr>
<td>( \Delta X = 3000 \text{Å}/404 )</td>
</tr>
<tr>
<td>Potentials (b)-(d)</td>
</tr>
<tr>
<td>( X_{min} = -2800 \text{Å}, \ X_{max} = 2000 \text{Å}, \ NX = 809, )</td>
</tr>
<tr>
<td>( \Delta X = 4800 \text{Å}/808 )</td>
</tr>
</tbody>
</table>

| **Long time propagation** |
| Potential (a) |
| \( X_{min} = -12465 \text{Å}, \ X_{max} = 12465 \text{Å}, \ NX = 2^{14}, \ \Delta X = 1.52174 \) |

<table>
<thead>
<tr>
<th>DAF and FFT Propagation Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short time propagations</strong></td>
</tr>
<tr>
<td>Potential (a)</td>
</tr>
<tr>
<td>( \tau = 1.6 \times 10^{-15} \text{sec}, \ Number of time steps = 400 )</td>
</tr>
<tr>
<td>DAF Gaussian Width ( \Sigma(0) = 3.5355 \text{Å} )</td>
</tr>
<tr>
<td>Maximum Hermite ( M = 40 )</td>
</tr>
<tr>
<td>Potentials (b)-(d)</td>
</tr>
<tr>
<td>( \tau = 4 \times 10^{-16} \text{sec}, \ Number of time steps = 1200 )</td>
</tr>
<tr>
<td>DAF Gaussian Width ( \Sigma(0) = 3.5355 \text{Å} )</td>
</tr>
<tr>
<td>Maximum Hermite ( M = 40 )</td>
</tr>
</tbody>
</table>

| **Long time propagation** |
| Potential (a) |
| \( \tau = 5 \times 10^{-16} \text{sec}, \ Number of time steps = 78,413; 100,000; 120,000 \) |
| DAF Gaussian Width \( \Sigma(0) = 3.5355 \text{Å} \) |
| DAF Gaussian Width\( \Sigma(0) = 3.5355 \text{Å} \) |
| Maximum Hermite \( M = 40 \) |
methods, combined with KRMC equations, to evaluate the action of $\exp(-iK\tau/h)$, and propagated a total of 400 time steps. A quantitative comparison of the DAF and FFT results for the real and imaginary parts of the wavepacket at randomly selected points is given in Table 7.3.

The agreement is seen to be excellent both with regard to the magnitude and phase of the wavepacket. However, it is also found that the wavepacket remains substantially trapped in the inter-barrier region. Therefore, it is important to carry out the propagation for a sufficiently long time, so that transmission and reflection probability amplitudes can be calculated. The transmission and reflection amplitudes are sensitive to the phase of the wavepacket and are typical of the type of quantity of interest in scattering problems. It is found that this double barrier problem is

Table 7.3: Comparison of the real and imaginary parts of the wavepacket after 400 time steps for the 1-D double barrier potential

<table>
<thead>
<tr>
<th>$z(\text{Å})$</th>
<th>$\text{Re}\Psi(z,t)$</th>
<th>$\text{Im}\Psi(z,t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1321.782</td>
<td>-0.204E-4</td>
<td>-0.204E-4</td>
</tr>
<tr>
<td>-1002.475</td>
<td>0.517E-4</td>
<td>0.517E-4</td>
</tr>
<tr>
<td>-750.000</td>
<td>0.35817E-2</td>
<td>0.35817E-2</td>
</tr>
<tr>
<td>-400.990</td>
<td>-0.120400E-1</td>
<td>-0.120402E-1</td>
</tr>
<tr>
<td>-200.495</td>
<td>0.414835E-1</td>
<td>0.414835E-1</td>
</tr>
<tr>
<td>0.00000</td>
<td>0.186421E-1</td>
<td>0.186421E-1</td>
</tr>
<tr>
<td>304.455</td>
<td>-0.205751E-1</td>
<td>-0.205751E-1</td>
</tr>
<tr>
<td>504.950</td>
<td>0.102290E-1</td>
<td>0.102291E-1</td>
</tr>
<tr>
<td>750.000</td>
<td>-0.303000E-2</td>
<td>-0.303000E-2</td>
</tr>
<tr>
<td>980.198</td>
<td>-0.910E-4</td>
<td>-0.910E-4</td>
</tr>
<tr>
<td>1240.01</td>
<td>0.510E-4</td>
<td>0.510E-4</td>
</tr>
</tbody>
</table>

$^a$E-4 denotes $10^{-4}$. 
Table 7.4: Converged reflection and transmission probabilities for electron scattering off the double barrier potential

<table>
<thead>
<tr>
<th>Reflection</th>
<th>78413 time steps</th>
<th>100000 time steps</th>
<th>120000 time steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAF</td>
<td>0.510</td>
<td>0.506</td>
<td>0.505</td>
</tr>
<tr>
<td>FFT</td>
<td>0.510</td>
<td>0.505</td>
<td>0.505</td>
</tr>
<tr>
<td>Analytical(^a)</td>
<td>0.509</td>
<td>0.509</td>
<td>0.509</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmission</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DAF</td>
<td>0.479</td>
<td>0.486</td>
<td>0.487</td>
</tr>
<tr>
<td>FFT</td>
<td>0.490</td>
<td></td>
<td>0.485</td>
</tr>
<tr>
<td>Analytical(^a)</td>
<td>0.491</td>
<td>0.491</td>
<td>0.491</td>
</tr>
</tbody>
</table>

\(^a\)Results obtained by standard, \textit{time independent} boundary matching of the wavefunction and derivative at potential discontinuities.

extremely challenging because it takes a very long time for the wavepacket (a total of 39 pico-seconds) to completely decay out of the double barrier region! It is very encouraging that the DAF approach is able to evolve the wavepacket correctly, including phase information, for such a long time. The resulting transmission probabilities obtained by the DAF and FFT methods, are given in Table 7.4, and agree with time independent results to within a percent.

The second model problem considered simply has a linear potential (constant classical force) added to the previous double barrier potential. The potential and propagation parameters are given in Tables 7.1 and 7.2. This system was propagated by the DAF and FFT methods for a total of 1200 time steps, but with a time step 1/4 that used in the study of the double barrier problem. In Figures 7.1 and 7.2, we present the comparison of the DAF and FFT results for the modulus and real parts of the wavepacket. It is seen that they agree very well. The constant acceleration due to the linear potential leads to ever increasing oscillations in the
Figure 7.1: Comparison of FFT and DAF methods for a double barrier + linear potential (real part)
Figure 7.2: Comparison of FFT and DAF methods for a double barrier + linear potential (absolute value)
real and imaginary parts of the wavepacket, and these high frequency oscillations are accurately reproduced by the DAF-Class free propagator. However, continued propagation by either the same DAF-Class free propagator or FFT grid will eventually be unable to accurately follow the wavepacket because, from the DAF point of view, the wavepacket will become so oscillatory as to leave the DAF class, while the FFT spatial grid implies a maximum momentum which can be correctly propagated, so that it also be unable to accurately propagate the packet.

The third model problem which we have solved takes the double barrier given in Table 7.1, and adds to it a time dependent, oscillating potential of the form given in Table 7.1. The moduls and real part of the wavepacket for this model are shown in Figures 7.3 and 7.4, where we again see excellent agreement between the DAF and FFT results. The time dependence of the potential makes this an interesting problem for the KRMC propagation equations because it is valid both for time independent and time dependent potentials [31, 47]. The oscillating field is seen to have a substantial effect on the overall probability distribution of where the electron is likely to be observed. A portion of the electron's wavepacket moves a significant distance, and a portion of the wavepacket is significantly delayed by the influence of the oscillating field.

Finally, the most challenging model incorporates the double barrier plus linear potential, and adds a spatially sinusoidally varying term; see Table 7.1. In order to accurately follow the wavepacket, the time steps are reduced to $4.0 \times 10^{-16}$ sec, and the real part and absolute value of the wavepacket are shown in Figures 7.5 and 7.6. The dynamics of the wavepacket are seen to be quite complicated with the linear term producing ever increasing oscillations, and the sinusoidal potential causing a
Figure 7.3: Comparison of FFT and DAF methods for a double barrier + cosine time-dependent potential (real part)
Figure 7.4: Comparison of FFT and DAF methods for a double barrier + cosine time-dependent potential (absolute value)
Figure 7.5: Comparison of FFT and DAF methods for a double barrier + linear + cosine time-independent potential (real part)

delay in part of the wavepacket (compared to Figures 7.1 and 7.2), and increased acceleration in part of the packet. The DAF procedure again is seen to yield results in quantitative agreement with those obtained using FFTs.

7.4 Conclusions

The objective of this paper has been to provide computational demonstrations of the ability of the DAF approach to treat correctly real time quantum dynamics. The model potentials chosen for this purpose were taken from a group of double barrier
Figure 7.6: Comparison of FFT and DAF methods for a double barrier + linear + cosine time-independent potential (absolute value)
systems which are useful in studying electron dynamics in quantum heterostructures [33]. The double square barrier by itself actually poses a substantial challenge because

1. it involves infinite forces (but finite impulses)

2. it can cause considerable distortion of the wavepacket, and

3. it can drastically delay the wavepacket in the region between the barriers, thereby requiring very long propagation times before the collision of the electron with the potential is over.

Additional difficulties are created by adding to the double barrier potential a linear potential, a time varying sinusoidal potential, or a linear plus spatially oscillating potential. The constant force due to the linear term creates ever increasing oscillations in the wavepacket, while the additional oscillating field compounds the high frequency oscillations with considerable distortion in the spatial distribution of the electron’s probability distribution. We consider the fact that the DAF—effective free propagator, used both in the KRMC and the KSSMO methods, was able to produce highly accurate results for all four of these models convincing evidence of the validity of the DAF formalism.

On the basis of these results, we are extremely optimistic that other versions of the DAF—formalism will also be successful and provide very powerful new tools for real time quantum dynamics [1–6]. We are carrying out many such computational studies now, with particular emphasis on the Gaussian biased sampling— Monte Carlo evaluation of the DAF—path integral scattering amplitude [20], and the quadrature (DDAF) and Monte Carlo (CDAF) evaluation of real time dynamics using the interacting—DAF formalism presented in [20].
8. CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

At the outset, we wanted to design parallel algorithms for studying the phenomenon of time-dependent quantum scattering. First we reduced a 3-dimensional problem to a 1-dimensional problem taking advantage of the symmetry of the problem and using the concept of center of mass. We implemented the algorithms on an nCUBE and a MasPar and compared the results of our implementations.

In our literature survey, we determined that the state of the art algorithms for the simulation of time-dependent quantum scattering used FFT methods. FFT methods, developed for uni-processor machines, involve applying a repeated sequence of very short time-step propagations.

At each time step, free propagation of the wavepacket is most easily handled in the momentum representation where as the effect of the potential is most easily obtained in the coordinate representation. These representations are Fourier Transforms of each other, and thus wave propagation requires two FFTs at each time step. These methods were shown to be expensive to use on parallel, distributed-memory machines, because the FFT makes the propagation algorithms communication bound. For example, on a hypercube, each neighbor has to communicate with each of its hypercube neighbors. This motivated us to design a formalism that allowed us to
propagate wavepackets in the coordinate space representation, eliminating the need for FFTs.

We introduced the notion of DAFs, where by we were able to handle the free propagation of a wavepacket in the coordinate representation. The DAF formalism comes about from the fact that the generating function of Hermite polynomials is the derivative of an exponential function. We showed that if we know the analytical propagation of an exponential function, we can derive a propagation scheme for any wavepacket. The resulting algorithm is an approximation algorithm, with different DAFs being required for wavepackets with different initial momentums.

The approximation algorithm we have developed is shown to be communication efficient and simple to implement. It requires the pre–computing of “weights” that can used to propagate the wavepacket on a grid. The wave function at time $t + \tau$ is computed from the wave function at time $t$ by an operation equivalent to a sparse–matrix vector multiply. We have presented the implementation details for 1–D wavepackets. We show that the algorithm is

- load balanced and
- employs nearest–neighbor communications.

We next attacked the development of the DAF formalism from a different viewpoint, so that we can specify the parameters of a DAF operator in a systematic manner. The DAF operator is shown to be a product of three terms. We examined the behavior of each of the terms to explain the behavior of the DAFs. We showed that the parameters of the DAF depend on the initial wavepacket, the size of the grid, the grid–spacing, and the size of the time–step. We gave a relationship which
guides the choice of the DAF parameters.

We showed the viability of our methods by using DAFs on challenging 1-D problems and comparing results against FFT methods. We demonstrated that our methods provide accurate simulations and are competitive with FFT methods even on uni-processor machines. When compared to analytical results, transmission and reflection coefficients calculated for the tunneling problem are, in fact, better than those calculated by FFT methods.

8.2 Future and Related Work

We have to demonstrate that the DAF formalism works for two dimensional and three dimensional scattering problems. These problems are particularly challenging, because multi-dimensional grids require large amounts of memory. There are two ways of attacking this problem:

- Use parallel disk I/O, so that secondary storage can be used to store data, without disk I/O becoming a bottle-neck.

- Explore methods for shortening grid size.

Fellow researchers have presented the idea of using traveling DAFs [19] (an extension of the DAF formalism) to effectively slow down the wavepacket, enabling one to use bigger grid spacing; and hence use fewer grid points. In some problems of interest, we are interested in "accurate" results only on a particular part of the grid. We should be able to use a fine grid in the area of interest and use a coarse grid elsewhere. This will also enable us to use smaller grids. Ideas for extending DAF methods for use on multi-grids are presented in [19].
Once the ideas for shortening the grids have been worked out, we can concentrate our efforts on two and three dimensional problems.
BIBLIOGRAPHY


[34] N. Makri. *Chem. Phy. Lett*, 159:489, 1989. The prefiltering technique developed in the paper is based on introducing a sharp cutoff, $P_{max}$, into the Fourier integral evaluation of the free particle propagator. The resulting effective free propagator possesses a banded structure, but decays with the distance propagated, $(x - x')$, as $\sin[P_{max}(x - x')]/(x - x')$, which is much weaker than the Gaussian decay of the DAF-effective free propagator.


APPENDIX A: CODE FOR GENERATING WEIGHTS

/* Program to generate weights */

#include <stdio.h>
#include <math.h>
#include "scat.h"
#define PI M_PI
#define HMAX 200
#define NMAX 6  /* no of hemites used */
#define M1 6

double
    norm[M1], h0[M1], b[M1];
double calapprox(), filter(), fnc();
FILE *fp1,*fopen();

main(){
    int i,j;
    double temp,y,a1;
    struct cmplx temp1,temp2,p,a,atau;
    int m,n;

    fp1 = fopen("weights","w");

    /*calculate b[i]s*/
    h0[0] = 1.0;
    norm[0] = sqrt(PI);
    b[0] = h0[0]/norm[0];
    for( i = 1; i < M1; i++){
        h0[i] = - h0[i - 1]/(double)i;
        norm[i] = 4.0 * norm[i -1];
        b[i] = h0[i]/norm[i];
    }

    /*printf("enter values for y,a,m,n\n");
    scanf("%lf %lf %d %d",&y,&a,&m,&n);
    printf("%g,%g,%d,%d\n",y,a,m,n);
    */

    y = 0.0;
printf("enter values for a\n");
scanf("%lf",&a1);
printf("%g\n",a1);
/*read bandwidth*/
printf("enter values for m\n");
scanf("%d",&m);
printf("%d\n",m);
fprintf(fp1,"%d\n",2*m+1);
a.re = a1;
a.im = 0.0;

temp1.re = 1.0;
temp1.im = 2 * UTAU * a1 * a1;
convctop(&p,&temp1);
powc(&p,&p,-0.5);
convptoc(&atau,&p);
printcomp(&atau);

/*
for(j = 1; j < NMAX; j++){
*/
    calapprox(&temp1,y,m,NMAX - 1,&a);
fnc(&temp2,y,0.0);
printf("function to be fitted is\n");
printcomp(&temp2);
printf("approx function is\n");
printcomp(&temp1);
proptau(&temp1,y,m,NMAX - 1,&a,&atau);
printf("propagated value is\n");
printcomp(&temp1);

/*
compmult(&temp1,&temp1,&atau);
printcomp(&temp1);
*/
fnc(&temp2,y,UTAU);
printf("exact propagated value is\n");
printcomp(&temp2);

/
printf("%19.17f, %19.17f\n",fnc(y),temp);
/
}

/*Just for fitting, no propagation*/
/*m = bandwidth*/
/*n = degree of polynomial*/
/*a = i/(sqrt(2)*sigma(0))*/
/*d = complex result*/
double

calapprox(d,y,m,n,a)
double y;
int m,n;
struct cmplx *d,*a;
{
    double x;
    struct cmplx temp,sum,val,temp1,temp2,temp3;
    struct cmplx p[HMAX];
    int i,j;
    temp.re = 0.0;
    temp.im = 0.0;
    temp2.im = 0.0;
    for(j = -m; j <= m; j++){
        fnc(&val,(double)j,0.0);
        temp2.re = y - (double)j;
        sum.re = 0.0;
        sum.im = 0.0;
        for( i = 0; i <= n; i++){
            herm(p,&temp2,i,a);
            compmult(&temp1,&p[2*i],a);
            multcbyr(&temp1,&temp1,b[i]);
            sum.re += temp1.re;
            sum.im += temp1.im;
        }
        printf("%d, %19.17f %19.17f\n",j,sum.re,sum.im);
        compmult(&sum,&sum,&val);
        temp.re += sum.re;
        temp.im += sum.im;
    }
    printf("%d,%19.17f, %19.17f\n",j,temp,sum);
*/
    d->re = temp.re;
    d->im = temp.im;
}

/*specify the function to be fitted or propagated*/
double fnc(d,x,t)
double x,t;
struct cmplx *d;
{
    evalint(d,1.9,x,t,0.0);
    d->re = 1.0;
    d->im = 0.0;
}

/*used in the calculation of the hermites*/
double filter(d,a,x)
struct cmplx *x,*a,*d;
{  
struct cmplx temp,temp2;
  double temp1;
  
  compmult(&temp,a,x);
  compmult(&temp2,&temp,&temp);
  temp1 = exp(-temp2.re);
  d->re = temp1*cos(temp2.im);
  d->im = temp1*sin(-temp2.im);
}

/*calculate the coefficients of hermite polynomials;
Each coefficient premultiplied by the filter*/
herm(p,x,n,a)
struct cmplx p[];
struct cmplx *x,*a;
int n;
{
  int i;
  struct cmplx temp,temp1,temp2;
   
  filter(&p[0],a,x);
  compmult(&temp,a,x);
  compmult(&p[1],&temp,&p[0]);
  multcbyr(&p[1],&p[1],2.0);
  for(i = 2; i <= 2*n; i++){
    multcbyr(&temp1,&p[i-2],2.0*(i -1));
    compmult(&temp2,&p[i-1],&temp);
    multcbyr(&temp2,&temp2,2.0);
    p[i].re = temp2.re - temp1.re;
    p[i].im = temp2.im - temp1.im;
  }
}

/* propagate function by timestep atau */
/*m = bandwidth*/
/*n = degree of polynomial*/
/*a = 1/(sqrt(2)*sigma(0))*/
/*d = complex result*/
proptau(d,y,m,n,a,atau)
double y;
int m,n;
struct cmplx *d,*a,*atau;
{
  double x;
  struct cmplx temp,sum,val,temp1,temp2,temp3,temp5;
  struct cmplx p[HMAX];
  int i,j;
  
  temp.re = 0.0;
  temp.im = 0.0;
  temp5.re = 0.0;
temp5.im = 0.0;

for(j = -m; j <= m; j++){  
    fnc(&val,(double)j,0.0);
    temp2.re = y - (double)j;
    temp2.im = 0.0;
    compmult(&temp2,atau,&temp2);
    sum.re = 0.0;
    sum.im = 0.0;
    temp3.re = 1.0;
    temp3.im = 0.0;
    for( i = 0; i <= n; i++){  
        herm(p,&temp2,i,a) ;
        /*
         * compmult(&temp1,&p[2*i],&val);
         */
        compmult(&temp1,&p[2*i],a);
        multcbyr(&temp1,&temp1,b[i]);
        compmult(&temp1,&temp1,&temp3);
        compmult(&temp3,&temp3,atau);
        compmult(&temp3,&temp3,atau);
        /*
        * compmult(&temp1,&temp1,&val);
        */
        sum.re += temp1.re;
        sum.im += temp1.im;
    }
    compmult(&sum,&sum,atau);
    /*
    printf("%d, %19.17f %19.17f\n",j,sum.re,sum.im);
    */
    fprintf(fp1,"%19.17e %19.17e\n",sum.re,sum.im);
    temp5.re += sum.re;
    temp5.im += sum.im;
    compmult(&sum,&sum,&val);
    temp.re += sum.re;
    temp.im += sum.im;
    }
    /*
    printf("%d,%19.17f, %19.17f\n",j,temp,sum);
    */
    d->re = temp.re;
    d->im = temp.im;
}
APPENDIX B: CODE FOR PROPAGATION

/*Program for 1-Dimensional quantum scattering with a specified potential field*/

#include "Pcmplx/broad.h"
#include "Filter/scat.h"

/*macros to decide whether to send points to neighboring processors. Values are sent to left(right) if oksl(oksr) is true;
The corresponding okrr(okrl) will be true in the neighboring processor*/
#define oksl(i) ((iproc > 0) && ((iproc + (i)) < nproc))
#define okrr(i) ((iproc >= 0) && ((iproc + (i)) < (nproc - 1)))
#define oksr(i) ((iproc < (nproc - 1)) && ((iproc - (i)) >= 0))
#define okrl(i) ((iproc < nproc) && ((iproc - (i)) > 0))
#define MAXNO 1 /*Maximum number of neighboring processors*/
#define NOOFWT 81 /*Maximum number of weights*/
#define GD 0x7fff /*Message type*/

#define ALPHA (1.0e+18*ME) /*Potential Parameter*/
#define BETA (8.1985806e-21) /*Potential Parameter*/
#define GAMMA (200.0e-10) /*Potential Parameter*/
#define IPOS (-2000.0e-10) /*Initial Position of the wave*/
#define V0 (6.1476414e-22) /*Height of potential parameters*/
#define XB0 (-105.0e-10) /*Four points of the barrier*/
#define XB1 (-35.0e-10)
#define XF0 (35.0e-10)
#define XF1 (105.0e-10)
#define C (2.99792458e8) /*Velocity of light*/

char s[20] = "lg.db"; /*Used for naming files*/
struct cmplx b[NOOFWT]; /*Array for holding weights*/
struct cmplx grid[PP]; /*Grid array*/
struct cmplx pot1[PP]; /*Potential array*/
struct cmplx pot2[PP]; /*Another potential array*/
struct cmplx wgrid[(2*MAXNO + 1)*PP]; /*Augmented grid (with data from neighboring processors*/
int noofwts,nop;
int i1,i2,i3,i4,i2_,i3_; /*points on the grid identified for
calculation of transmission and
reflection coefficients*/
int i3proc,i2proc,i3_proc,i2_proc; /* processors corresponding
to these points*/
struct cmplx t1sum,t2sum,t3sum,t4sum; /*Running sums needed
for calculation of coefficients*/

struct cmplx   fac,tfact;
double dsumnO;

main(argc,argv)  
int argc;
char *argv[] ;
{  
   struct cmplx suml,sum2;
   int i,j,k,l,n,lindex,rindex;
   FILE *fp1,*fopen();

   init();
   setup();
   n = atoi(argv[argc -1]);
   l = strlen(s);
   calsqmod(&sum1);  /*Calculate the initial normalization*/
   calpot(0);  /*Calculate potential factors*/
   k = 1;
   for( i = 0, j = atoi(argv[k]) ; i < n; i++){/*for all time steps
specified on the command line */
      if( i == j){
         calsqmod(&sum2);
         if(iproc == 0){
            mystrcat(s,argv[k],1);
            fp1 = fopen(s,"w");
            fprintf(fp1,"Normalization = \%%10.8f\n",sum1.re);
            fprintf(fp1,"Normalization = \%%10.8f\n",sum2.re);
            outp(j,fp1);
            fclose(fp1);
         }
         else{
            outp(j,fp1);
         }
         j = atoi(argv[++k]);
      }
      multbypot(pot1);  /* Multiply by the potential factor*/
      gatherpoints(nop,&lindex,&rindex); /*get points from neighbors*/
      calnval(i,lindex,rindex,nop);  /*new values of the
wavefunction*/
   } /*for all time steps*/
   calpot(i + 1);
} /*for all time steps*/
multbypot(pot1);  /*Multiply by potential factor*/
calsqmod(&sum2); /*Calculate normalization*/

if(iproc == 0){
    mystrcat(s,argv[k],1);
    fpl = fopen(s,"w");
    fprintf(fp1,"Normalization = \%10.8f\n",sum1.re);
    fprintf(fp1,"Normalization = \%10.8f\n",sum2.re);
    outp(i,fp1); /*output function*/
    fclose(fp1);
}
else{
    outp(i,fp1);
}

}

init(){ /*initialize iproc etc.*/
    whoami(&iproc,&pid,&host,&lnproc);
    nproc = ( 1 << lnproc );
}

setup(){
    FILE *fpl,*fopen();
    int i,j,nbytes,nityp;
    fp1 = fopen("Filter/weights","r");
    nbytes = sizeof(int);
    dest = 0xffff;
    source = -1;
    typ = 0;
    nityp = 1;
    if(iproc){ /*Broadcast weights to all processors*/
        fscanf(fp1,"\%d",&noofwts);
        nwrite(&noofwts,nbytes,dest,nityp,NULL);
        for(j=0; j < noofwts; j++){
            fscanf(fp1,"\%lf \%lf",&b[j].re,&b[j].im);
        }
        nbytes = sizeof(struct cmplx)*(noofwts);
        nwrite(b,nbytes,dest,typ,NULL);
    }
    nread(&noofwts,nbytes,&source,&nityp,NULL);
    nbytes = sizeof(struct cmplx)*(noofwts);
    nread(b,nbytes,&source,&typ,NULL);
    calnop(&nop);
    calfunc(grid,0.0,IPOS/DELX); /*Calculate the initial wavefunction*/
calcfunc(grid, time, intpos) /*Calculate the analytic wavefunction at time and initialpos intpos*/
struct cmplx grid[];
double time, intpos;
{
    int i, igt;

    igt = -(PP * nproc)/2 + 1 + iproc * PP;
    for(i = 0; i < PP; i++){
        evalint(&grid[i], USIGMA, (1.0*igt-intpos), time, UP);
        igt++;
    }
    /* Identify points to the left and right of barrier */
    igt = -(PP * nproc)/2 + 1 + iproc * PP;
    i3proc = -1;
    i3 = -1;
    for(i = 0; i < PP; i++){
        if((igt * DELX > XF1) && ((igt - 1)* DELX <= XF1)){
            i3 = i;
            i4 = i + 1;
            i3proc = iproc;
            printf("igt= %d, i3 =%d, i3proc=%d\n", igt, i3, i3proc);
            printf("z3 = %19.17e",igt*DELX);
            if(i == PP - 1){
                printf("i3 and i4 in differnt procs\n");
            }
            t3sum.re = grid[i3].re;
            t3sum.im = grid[i3].im;
            t4sum.re = grid[i4].re;
            t4sum.im = grid[i4].im;
            fac.re = cos(EO/HB*TAU);
            fac.im = sin(EO/HB*TAU);
            printf("fact ");
            printcomp(&fac);
            tfact.re = 1.0;
            tfact.im = 0.0;
            break;
        }
    }
    igt++;
}
igt = -(PP * nproc)/2 + 1 + iproc * PP;
i2_proc = -1;
i2_ = -1;
for(i = 0; i < PP; i++){
    if(fabs(igt * DELX - XB1) < 0.1*DELX ){
        i2_ = i;
    }
i2_proc = iproc;
printf("igt= %d,i2_ =%d,i2_proc=%d\n",igt,i2_,i2_proc);
break;
}
igt++;
}

/*identify points to the left of the barrier*/
igt = -(PP * nproc)/2 + 1 + iproc *PP;
i2proc = -1;
i2 = -1;
for(i = 0; i < PP; i++){
    if((igt * DELX < XBO) && ((igt +1)* DELX >= XBO)){
        i2 = i;
i1 = i - 1;
i2proc = iproc;
printf("igt= %d,i2 =%d,i2proc=%d\n",igt,i2,i2proc);
if(i == 0){
    printf("i3 and i4 in different procs\n");
}
t1sum.re = grid[i1].re;
t1sum.im = grid[i1].im;
t2sum.re = grid[i2].re;
t2sum.im = grid[i2].im;
fac.re = cos(EO/HB*TAU);
fac.im = sin(EO/HB*TAU);
printf("fact ");
printcomp(&fac);
tfact.re = 1.0;
tfact.im = 0.0;
break;
}
igt++;
}

igt = -(PP * nproc)/2 + 1 + iproc *PP;
i3_proc = -1;
i3_ = -1;
for(i = 0; i < PP; i++){    
    if(fabs(igt * DELX - XF0) < 0.1*DELX ){    
        i3_ = i;
i3_proc = iproc;
printf("igt= %d,i3_ =%d,i3_proc=%d\n",igt,i1_,i3_proc);
break;
}
igt++;
}

/*Calculate the unit potential*/
double
uv(x,t)
double x;
int t;
{
  double v();
  return(ME * (DELX * DELX) * v(x,t)/(HB * HB));
}

/*Calculate the actual potential*/
double
v(x,t)
double x;
int t;
{
  double rtime = t * TAU;
  /*
   return((ALPHA * DELX *x) - BETA*cos((M_PI * DELX * x)/GAMMA));
   */
  /*
   return(M_PI/100.0);
   */
  /*
   return(ALPHA * DELX * x);
   */
  /*Simple Double Barrier*/
    if(DELX * X < (XBO - 0.1*DELX)){
      return(0.0 );
    }else{
      if(DELX * x < (XB1 + 0.1*DELX)){
        /*
           printf("barrier point %f, %f\n",x,x^DELX);
        */
        return(V0 );
      }else{
        if(DELX *x < (XF0 - 0.1*DELX)){
          return(0.0 );
        }else{
          if(DELX * x < (XF1 + 0.1*DELX)){
            /*
               printf("barr point %f, %f\n",x,x*DELX);
            */
            return(V0 );
          }else{
            /*
               printf("bar point %f, %f\n",x,x*DELX);
            */
            return(V0 );
          }
        }
      }
    }
return(0.0);
}
}

/*Linear + Double Barrier Potential*/
/*
if(DELX * x <= XBO){
return(0.0 +
ALPHA*x*DELX - BETA*cos(M_PI * ( DELX * x )/GAMMA ));
}
else{
if(DELX * x <= XB1){
return(V0 +
ALPHA*x*DELX - BETA*cos(M_PI * ( DELX * x )/GAMMA ));
}
else{
if(DELX *x <= XFO){
return(0.0 +
ALPHA*x*DELX - BETA*cos(M_PI * ( DELX * x )/GAMMA ));
}
else{
if(DELX * x <= XF1){
return(V0 +
ALPHA*x*DELX - BETA*cos(M_PI * ( DELX * x )/GAMMA ));
}
else{
return(0.0 +
ALPHA*x*DELX - BETA*cos(M_PI * ( DELX * x )/GAMMA ));
}
}
}
*/
/*Cosine time-dependent + double barrier potential*/
/*
if(DELX * x <= XBO){
return(0.0 +
-BETA*cos(M_PI * ( DELX * x -rtime*C )/GAMMA ));
}
else{
if(DELX * x <= XB1){
return(V0 +
-BETA*cos(M_PI * ( DELX * x -rtime*C )/GAMMA ));
}
else{
if(DELX *x <= XFO){
return(0.0 +
-BETA*cos(M_PI * ( DELX * x -rtime*C )/GAMMA ));
}
else{

}
if(DELX * x <= XF1){
    return(V0 +
    - BETA*cos(M_PI * ( DELX * x -rtime*C )/GAMMA) );
} else{
    return(0.0 +
    - BETA*cos(M_PI * ( DELX * x -rtime*C )/GAMMA) );
}
} /*
}

hfactor(d1,d2,tau,v) /*Calculation of potentaial factors*/
struct cmplx *d1,*d2;
double tau,v;
{
    double temp1,temp2;

    temp1 = tau * v/2.0;
    expl(d1,-temp1);
    /*for modified cayley */
    /*
    d1->re = 1.0;
d1->im = -temp1;
temp2 = 1+temp1*temp1;
d2->re = 1.0/temp2;
d2->im = -temp1/temp2;
    */
}

calpot(t) /*Calculate the hfactors at time t*/
int t;
{
    int i,igt;

    igt = -(PP * nproc)/2 + 1 + iproc *PP;
    for(i = 0; i < PP; i++){
        hfactor(&potl[i],&potl[i],UTAU,uv((double)igt,t));
        igt++;
    }
}

multbypot(hpot) /*Multiply the wavefunction by the potential factors*/
struct cmplx hpot[];
{
    int i,igt;
    struct cmplx temp;
if(iproc == i2proc || iproc == i3proc || iproc == i2_proc ||
iproc == i3_proc){
    for(i = 0; i < PP; i++){
        if((i == i2 + 1) || i == i2_)
            (i == i3 -1) || i == i3_){
            temp.re = grid[i].re;
            temp.im = grid[i].im;
            compmult(&grid[i],&grid[i],&hpot[i]);
            grid[i].re += temp.re;
            grid[i].im += temp.im;
            multcbyr(&grid[i],&grid[i],(double)0.5);
        }
    }
} else{
    for(i = 0; i < PP; i++){
        compmult(&grid[i],&grid[i],&hpot[i]);
    }
}
calsqmod(sum) /*Calculate the normalization of the wavefunction*/
struct cmplx *sum;
{
    int i;
    double temp,temp1;
    temp = 0.0;
    for(i = 0; i < PP ; i++){
        temp += grid[i].re * grid[i].re;
        temp += grid[i].im * grid[i].im;
    }
    sum->re = dsumn(temp);
    sum->im = 0.0;
}
gatherpoints(noofprocs,lin,rin) /*gather wavefunction from the neighboring processors*/
/*lin and rin are indexes to the extreme left and right of array (returned values)*/
int noofprocs;
int *lin,*rin;
{ int i, j,lindex,rindex,source,dest,nbytes,typ;
nbytes = sizeof(struct cmplx) * PP;

j = lindex = rindex = MAXNO * PP;
for(i = 0; i < PP; i++){
    wgrid[j++] = grid[i];
}
rindex += PP - 1;

/* receive from right */
source = iproc + 1;
/*
source = -1;
*/
dest = iproc - 1;

/* send grid values to left neighbors (receive from right neighbors)*/
for(i = 0; i < noofprocs; i++){
    if(oksl(i)){
        nwrite(&wgrid[rindex - PP + 1], nbytes, dest, i, NULL);
    }
    if(okrr(i)){
        nread(&wgrid[rindex + 1], nbytes, &source, &i, NULL);
        rindex += PP;
    }
}

source = iproc - 1;
dest = iproc + 1;

/* send grid values to right neighbors (receive from left neighbors)*/
for(i = 0; i < noofprocs; i++){
    if(oks1(i)){
        nwrite(&wgrid[lindex], nbytes, dest, i, NULL);
    }
    if(oks2(i)){
        nread(&wgrid[lindex - PP], nbytes, &source, &i, NULL);
        lindex -= PP;
    }
}

*rin = rindex;
*lin = lindex;


calnval(n, lindex, rindex, noofproc)
int lindex, rindex, noofproc, n;
{
    void updcoeff();

    int i, j, k, mindex, sindex;
    struct cmplx sum, temp, temp2;
for(i = 0; i < lindex; i++){
    wgrid[i].re = 0.0;
    wgrid[i].im = 0.0;
}

/*For Plane Wave Only*/
/*

if(iproc == 0){
    double f1,f2;
    int igt;

    f1 = P * P/(2 * ME);
    f2 = TAU * n;
    temp.re = cos(f1 * f2/HB);
    temp.im = -sin(f1*f2/HB);
    igt = -(PP * nproc)/2 + 1 - noofwts/2;
    j = lindex -= noofwts/2;
    if(lindex < 0){
        printf("error\n");
    }
    for(i = 0; i < noofwts/2; i++){
        evalint(&temp2,USIGMA,igt*l.0-IP0S,0.0,UP);
        compmult(&wgrid[j],&temp,&temp2);
        igt++,j++;
    }
}

for(i = rindex + 1; i < (noofproc*2 + 1)*PP; i++){
    wgrid[i].re = 0.0;
    wgrid[i].im = 0.0;
}

mindex = PP * MAXNO;
if( mindex < lindex || mindex > rindex){
    printf("error in %d\n",iproc);
}

/*Calculate new values*/
for(i = 0; i < PP; i++){
    sindex = mindex + i - noofwts/2;
    sum.re = 0.0;
    sum.im = 0.0;
    for(j = sindex, k = 0; k < noofwts; k++,j++){
        compmult(&temp,&bCk[j],&wgrid[j]);
        sum.re += temp.re;
        sum.im += temp.im;
    }
    grid[i].re = sum.re;
    grid[i].im = sum.im;
/*calculations for reflection and transmission coefficients*/
updcoeff();

void updcoeff(){ /*update runnings sums used for calculation of reflection and transmission coefficients*/
struct cmplx temp;

compmult(&tfact,&tfact,&fac);
if(i2proc >= 0){
    compmult(&temp,&grid[i1],&tfact);
    t1sum.re += temp.re;
    t1sum.im += temp.im;
    compmult(&temp,&grid[i2],&tfact);
    t2sum.re += temp.re;
    t2sum.im += temp.im;
}
if(i3proc >= 0){
    compmult(&temp,&grid[i3],&tfact);
    t3sum.re += temp.re;
    t3sum.im += temp.im;
    compmult(&temp,&grid[i4],&tfact);
    t4sum.re += temp.re;
    t4sum.im += temp.im;
}
}

calnop(nop) /*How many neighboring processors used ?*/
    /*value returned in nop*/
int *nop;
{
    int i,j;
    j = 1;
    i = PP;
    while(i < noofwts/2){
        j++;
        i += PP;
    }
    if( j > MAXNO){
        printf("error in calculation of nproca in %d\n",iproc);
    }
    *nop = j;
}

outp(n,fp1) /*output at iteration number n using file pointer fp1*/
    int n;
    FILE *fp1;
{
    struct cmplx temp,temp1,temp6;
struct cmplx sum1;
int i,j,k,l,lindex,rindex,noofpts,noofpts2;
double x,y,width,t,temp3,temp4,temp5;
char buf1;
int source,typ,nbytes;
/*Yin Ma fudge factor*/
double con = sqrt((double)(DELX * 1.0e+10));

if(iproc == 0){
    struct cmplx buf[PP];
    j = 0;
    temp5 = 0.0;
    fprintf(fp1,"Parameters of the problem are\n");
    fprintf(fp1,
        "Sigma = %19.17e, tau = %19.17e \nDelx = %19.17f P = %19.17f",
        SIGMA,TAU,DELX,P);
    fprintf(fp1,
        "USigma = %19.17e, Utau = %19.17e \nUDElx = %19.17f UP = %19.17f",
        USIGMA,UTAU,UDELX,UP);
    fprintf(fp1,\"No. of time steps = %d\n",n);
    fprintf(fp1,\"No. of grid points = %d\n",PP * nproc);
    noofpts = PP * nproc;
    noofpts2 = noofpts/2;
    k = -noofpts2 + 1;

    /*node 0 write its portion of the grid */
    for(l = 0; l < PP; l++){
        fprintf(fp1,\"%d %19.17f\n",k,DELX*1.0e10);
        temp3 =
            sqrt(grid[l].re*grid[l].re + grid[l].im*grid[l].im);
        fprintf(fp1,\"re=%19.17f , im=%19.17f , abs=%19.17f\n",
            grid[l].re/con,grid[l].im/con,temp3/con);
        evalint(&temp,USIGMA,k-IPOS/DELX,UTAU+n,UP);
        fprintf(fp1,\"%19.17f , %19.17f\n",temp.re,temp.im);
        k++;
    }
    nbytes = sizeof(struct cmplx)*PP;
    typ = GD;
    for(i = 1; i < nproc; i++){
        nwrite(&buf1,0,i,GD,0);
        nread(buf,nbytes,&i,&typ,0);
        for(l = 0; l < PP; l++){
            fprintf(fp1,\"%d %19.17f\n",k,DELX*1.0e10);
            temp3 = sqrt(buf[l].re*buf[l].re + buf[l].im*buf[l].im);
            fprintf(fp1,\"re=%19.17f , im=%19.17f , abs=%19.17f\n",
                buf[l].re/con,buf[l].im/con,temp3/con);
/*
   evalint(&temp,USIGMA,k-IPOS/DELX,UTAU*n,UP);
   fprintf(fp1,"%19.17f ,%19.17f\n",temp.re,temp.im);
*/
   k++;
}
} else{
    source = 0;
typ = GD;
nread(&buf1,0,&source,&typ,0);
nbytes = PP * sizeof(struct cmplx);
nwrite(&grid[0],nbytes,0,GD,NULL);
}
if(i2proc >= 0){
   struct cmplx temp1;
double temp2 = sqrt((double)(DELX*1.0e10));
printf("no of iterations = %d\n",n);
divcbyr(&temp1,&tisum,temp2);
printf("tisum ");
printcomp(&temp1);
divcbyr(&temp1,&t2sum,temp2);
printf("t2sum ");
printcomp(&temp1);
fflush(stdout);
}
if(i3proc >= 0){
   struct cmplx temp1;
double temp2 = sqrt((double)(DELX*1.0e10));
printf("no of iterations = %d\n",n);
divcbyr(&temp1,&t3sum,temp2);
printf("t3sum ");
printcomp(&temp1);
divcbyr(&temp1,&t4sum,temp2);
printf("t4sum ");
printcomp(&temp1);
printf("tfact ");
printcomp(&tfact);
fflush(stdout);
}
}
mystrcat(s,t,n) /*string concatenation*/
char *s, *t;
int n;
{
   int i,j;
i = n;
j = 0;
while((s[i++] = t[j++]) != '\0');