The Parallel Glasgow Shell Model Code And Applications.

Brian Ewins.

June 5, 1995

This thesis is dedicated to the memory of Donald Reid.

Potius sero quam nunquam.

(Better late than never.)

Livy

Abstract

This thesis describes the further development of the Glasgow Shell Model code, following on from the thesis of Dr. Mohammed Riaz. In that work, a possible parallel development of the Glasgow code was discussed, and a simplified version of the code constructed which could only run on three processors. Rather than immediately continue in this direction, we felt it would be worthwhile to investigate all of the possible ways that the code could be implemented in parallel, or that the current parallel version of the code could be made faster. Various models of the code were used to arrive at an implementation which is best able to satisfy our expectations for the parallelized version of the program.

The development of this code is then described, showing how the code was written to be at once as optimal and as portable as possible, to take account of future architectures that may become available to us, and discussing problems that arise in doing large shell model calculations (in parallel or not).

We go on to describe some applications of the new code, specifically to the termination of rotational bands in light *sd*-shell nuclei, and some original calculations in the cranked shell-model description of the nucleus, with a deformed basis being used.

Finally the alternatives to the present work are described, showing where the present shell model code fits into the panoply of nuclear models for large-basis calculations.

Declaration

Except where specific reference is made to the work of others, this thesis has been composed by the author. It has not been accepted in any previous application for a degree. I further state that no part of this thesis has already been or currently is being submitted for any degree or qualification at any other university.

Brian Ewins.

Acknowledgments

> Really it is not I who am writing this crazy book. It is you, and you, and you, and that man over there, and that girl at the next table.
>
> *James Joyce, Finnegan's Wake.*

I'd like to thank a whole bunch of people without whom I could not have completed this thesis. The work was funded by the SERC. For help during the work involved, and the writing up, I'd firstly like to thank my supervisor, Rex Whitehead, and the other members of the Nuclear Structure Group: Sandy Watt, Mourad Abdelaziz, and Leila Ayat. I single out Luke Taylor for special thanks for putting up with my bad jokes, and always being ready for more beer through three years sharing our office and carry-outs.

My erstwhile collaborators, including Neil Rowley, Stefan Frauendorf, Charles Blythe, and Ian Wright, I'd like to thank for taking the time to explain their problems to me, even though I couldn't be as helpful as I'd have liked. I'd like to thank Derek Higgins, Anne McKinnon, Sam Kilgour, and Gary Keiliff for their patient help in the computing aspects of this project.

Next come my family: Mum, Dad, Ciaran, Anna-Maria, Patrick and Liam, Michael, Mary Jane, Claire, Colette, and Paul – for food, cash, and not getting on at me too much for being so darned slow to finish.

To flatmates Gillian, Rebecca, Mary, and Pamela, finally you can get someone who'll wash the dishes!

And finally all the little people, the back-room boys (and girls), without whom I couldn't have gone to the pub when I was skint: Alan, Amarjit man, Andrew, Betty, Carolyn, Chrispfffft! , David, David, David, Douglas, Wing Cmdr. Gary, Gezza, Gordon, Helen, El Presidente Jack, Saad, Terry, and any David I missed out.

Contents

List of Figures

Chapter 1

Introduction

> Forsan et haec olim meminisse iuvabit.
> (One day we'll look back on this and laugh.)
> *Virgil, Aeneid.*

1.1 Beginnings.

In the original article on the Glasgow Shell Model Code [1], the authors made a statement which applies, to a large extent, to the contents of this thesis:

> The unconventional techniques embodied in the Glasgow shell–model program have aroused fairly widespread interest among ... nuclear physicists. While the usefulness of these methods lies in their simplicity, and hence their suitability for computation, they are ... far removed from usual shell–model ideas ... Our exposition will be a little unusual in that the implementation of our method on computers is inextricably linked to the choice of the methods themselves and in many cases separate discussion is neither desirable nor possible.

While this is still pertinent, the computational difficulties have, over the intervening years, become less intertwined with the theoretical issues of nuclear physics, and so I have attempted to write this thesis in such a way that the chapters on the two topics can be read independently. However, the development of the present code, and its subsequent applications, both depend on the limitations and advantages of the original code, so this first chapter will be devoted to placing the work of the thesis in its historical context.

1.2 Motivation

The process of parallelizing the Glasgow code was begun in the thesis prior to this work, of Dr. Mohammed Riaz[2]. The approach of that work was somewhat more ad-hoc than the present discussion, being intended to show that a parallel version of the code may be possible, and went some way to implementing such a program.

In this thesis, the over-riding consideration is how fast we can expect the parallel code to run, and to this end, every aspect of parallelizing the Glasgow code, from the topology of the network used, to the order that bits should be in in a Slater determinant, to variations of the Lanczos algorithm, are discussed. We take the point of view of computer science and consider the time complexity of the program with respect to the various parameters of its operation, so as to predict accurately how large are the calculations that we will be able to run, on the present or any future architecture.

Another facet of this project that has become apparent as work went on is that there is a questionmark over whether or not the code would find any applications. It has been said that there is very little to gain in standard shell model calculations until the size of the basis can be increased by more than 2 orders of magnitude (the present code is designed to enable us to approach that figure, if it works as well as could possibly be hoped, but even that would be

stretching its capabilities). We demonstrate that there *are* indeed calculations which can usefully be done by the present program which were not possible previously, and don't require such massive basis sizes. We also answer the criticism that large basis calculations are better done by different methods (specifically, F.D.S.M. and Monte-Carlo models).

1.3 An Overview of Nuclear Theory.

1.3.1 Historical Models.

In the 1930's and 40's, the prevailing models of the nucleus were of the 'liquid drop' type; these predicted the macroscopic properties of the nucleus, such as its ground state energy, and, indeed, the nucleus was thought to be very like classical liquids — i.e. unstructured. However, one of the results of the liquid drop model was the Bethe-Weisacker Semi-Empirical mass formula, and it was found in experiments that there were systematic deviations from this formula at particular mass numbers, and that some 'mirror' nuclei were more stable than their isobars[3]. This was very suggestive of a shell model of the nucleus, in analogy with the electron shell model.

One strong objector to this emerging theory was Niels Bohr. The implication of a shell model is that filled shells are inactive; i.e. there are a small number of nucleons, perhaps only one, outside the filled shells which determines some of the properties of the nucleus, such as, for instance, the angular momentum of the ground state. Bohr's objection was basically that the nucleus was so dense that each nucleon would undergo many collisions in the time that it would take the nucleon to make a single orbit; this would scatter it into many different orbits, thus making us unable to determine its orbit in macroscopic time. The consequence is that no single particle orbit can determine macroscopic effects, such as the angular momentum of the ground

state. The objection was later shown to be false; the Pauli principle excludes nucleons from identical orbits, preventing the frequency of collision that Bohr imagined[4].

1.3.2 The Shell Model.

Nuclear shell model theory that we are familiar with today really began with the work of Mayer and Jensen in 1949 [5]. (One notes that both the shell model and programming, the subjects of this thesis, were started by women (Mayer, and Ada Lovelace[6]), an unusual circumstance in the sciences). In a series of papers, a single particle model was used to predict properties of the nucleus. By this point in time, some excited state properties were known, and in a few nuclei, this model could predict energies and transition rates for these states. The central part of this model was the assumption of a central potential formed by the core, which had a strong spin-orbit component, which gave the 'correct' ordering of levels.

This model, where there were exact integer numbers of particles in each shell, proved inadequate to the task of explaining the vast bulk of nuclear data, however. There is also the dissatisfying element of the ad-hoc addition of a core potential. The first improvement on this was the introduction of a two-body interaction [7] between particles in the valence shell. This was still seen as a perturbation on the single-particle Hamiltonian, which was designed to make the calculation fit the data. The calculations now allowed the number of particles in each shell to be non-integral. Physically, this could be seen as measuring the degree to which Bohr was correct, the single particle orbits are subsumed into collective motion.

1.3.3 Modern Shell Theory.

It seems to be a small step from here to removing the core term; we simply increase the size of the calculation until the entire core, with its two-body interaction with the valence particles is included. This has not been the case, for a variety of reasons:

- The problem just gets too big to handle. Indeed, this is the reason for the existence of this thesis.
- The two-body interaction does not seem sufficient to explain what happens in the core (i.e. the saturated part of the nucleus).

There are a variety of reasons for the second problem; two-body interactions are generally constructed in the simplest possible form that explains the phase shifts observed in scattering experiments [8], preferably with some underlying 'meson exchange' justification. However, many potentials are phase-shift equivalent, so a large part of the choice is up to the theorist, and popularly the interaction is fitted to excited states in small calculations [9], which do not have the core effects that we want to examine. Also, the interaction should have a 'hard core' — i.e. nucleons act like billiard balls, not points. This effect is impossible to include exactly in the matrix elements that result. Finally, we may have to go further than the two-body interaction, specifically, to include density-dependent effects, it is simpler (and is equivalent in some cases [10]) in the shell model to have a three body interaction. These effects seem to improve Hartree-Fock calculations of the ground-state properties of the nucleus, and fortunately, there is a relatively simple way of constructing such matrix elements from two-body matrix elements, described in the original paper on the Glasgow code [11].

It must be emphasized at this point that the shell model of the 1940's was a single-particle shell model, whereas nowadays practitioners use a more realistic

two-body interaction, and allow full configuration mixing, the only limit to the calculation being the size of computer used. However the advantage of having a shell structure in the nucleus was as great then as it is now. It allows us to divide up the nucleus into three parts: the core, the valence shells, and the outer shells. The main tenet of shell model theory is that both the core and the outer shells can be pretty much ignored in calculating many properties of the nucleus. This reduction is what allows nuclear theory calculations to be done — since the number of states in a problem depends combinatorially on the number of shells that particles are allowed to fill, anything that reduces this number makes the problem more tractable.

The proton-neutron 2-body shell model that is used in the Glasgow code is not the only descendant of the original shell model work, however. Since the next stage of the work on nuclei took place in the age before computers, algebraic methods had to be found for easier solution of the huge problems that arise as the number of particles and shells considered grows. The obvious way to do this is to exploit conserved or nearly conserved symmetries of the Hamiltonian to reduce the original problem to a series of smaller problems. The first candidate for such reduction is also obvious, since the Hamiltonian must be rotationally invariant. An approach of this sort reduces the number of states involved in any shell model calculation by 10-fold or more. This kind of model, in which each nuclear state has a well defined angular momentum, is called the J-scheme. The Glasgow code emerged in the 1970's as a direct rival to codes based around this idea. Note, however, that since no 'nearly conserved' symmetry has been assumed to be true, all states in the energy range probed by a calculation using this method should appear.

1.3.4 Truncation Schemes.

There are several ways of reducing the size of shell-model calculations:

- By using exact symmetries. As mentioned previously, the Hamiltonian preserves J^2 and J_z; hence subspaces with different values for these quantum numbers can be considered separately. An additional symmetry imporant in the present, large basis, calculations, is parity. Only considering states of one parity reduces the size of multi-shell calculations by half.

- By using nearly-exact symmetries. Some states in the nucleus are well described by applying conservation of some group operation. The Fermion Dynamical Symmetry Model (F.D.S.M.) uses this technique [12], as do, to some extent, the I.B.M./I.B.A. models[13].

- Using the shell structure of the nucleus. As well as considering shells as closed, we can also note that transitions between major shells are supressed by a factor $1/2\hbar\omega$ (the 2 here is because adjacent major shells have opposite parities), since $2\hbar\omega$ is the energy required to create a state with 2 particles excited into other shells. We then use this to limit excitations, not at the $0\hbar\omega$ level (considering a single closed shell) but at $2\hbar\omega$, $4\hbar\omega$, ..., levels. Or, a nearly equivalent (but easier to implement) scheme is to place bounds on the numbers of particles that can appear in each major or minor shell.

The point is, that apart from using exact symmetries, truncation schemes *exclude* states from calculations. Restricting the basis via an energy denominator has most justification for us, since the nature of the shell model is to examine low-lying excited states. This is not to say that imposing symmetries is not a reasonable thing to do, but since states with good symmetry are well described by other models, and the states that are left out as a consequence may well be near the ground state, we should really be concentrating on precisely the states that these schemes miss out!

One possible avenue for using the nearly conserved symmetry truncations that may be more fruitful is to investigate the connection between the 'true' Hamiltonian – the interaction used in the shell model – and the parameters of these models.

1.3.5 Modern Collective Methods.

When greater restrictions are placed on the nuclear Hamiltonian, it is no longer true that all states in the probed energy range will be seen. However, symmetry-based truncation schemes and models are both resonable and useful in many situations. The Interacting Boson Approximation (I.B.A.) is a model that grew out of Wigner supermultiplet theory of the 1950's [14]. In it, the fermions in the nucleus are assumed to 'pair up' to form bosons, which then undergo a variety of reactions which sometimes seem chosen as much for their ease of calculation, e.g. separability of the interaction as any resemblance to an underlying nucleon-nucleon interaction. Particular choices produce the vibrational and rotational bands seen in many nuclei. This model is closely allied to the Interacting Boson Model[13], where there is no longer a close tie to the 'real' fermions. Handwavingly, the approximation works since fermions do tend to pair up in the nucleus, somewhat like Cooper pairs in superfluids[15]. The effect is obvious in the difference in binding energy of neighbouring even-even and odd-even nuclei, and indeed the looseness of the 'extra' particle was one of the reasons for the success of Mayer's original single particle model.

It is also striking in some nuclei that they appear to be solely composed of α-particles, for instance, ^{16}O. This led to the construction of models based on α-particles and the forces between them, derived or contrived at one step removed from the fundamental quark interactions [16]. These are able to predict some surprisingly stable chain states of the $4n$ nuclei, which are more deformed (from a collective viewpoint) than anything previously discovered.

Moving up from pairs and αs we come to cluster models, where the stable 'lumps' that form are magic nuclei assumed to orbit each other [17]. One of the major interesting features of this branch of the subject is that by reducing appropriate nuclei to three-body problems one can write down the Fadeev version of Schroedinger's equation for the system, and thereby solve it exactly, at least in principle.

Another possibility is to look at further approximate symmetries of the Hamiltonian. An extension which examines the groups Sp(6) and SO(8) is the Fermion Dynamical Symmetry Model (F.D.S.M.). In this model, fermions are again combined into pairs as bosons, and the model space is truncated where these bosons have pseudo-angular momentum less than or equal to some value (typically 2). This is very like an extension of I.B.A., and the correspondence of bosons to fermions in both these models allows mappings to realistic Hamiltonians to be made. Indeed, the subgroup chains belonging to the pseudo-SU(3) and SU(4) models of Arima and Hecht are contained in those of the above groups [12, 13].

1.3.6 Advantages of Shell-Model Method.

However, all of the models except the J-scheme mentioned above, and the m-scheme used in the Glasgow code, represent truncations of the model space. While this does not mean that the results that they obtain are wrong, it does mean that a typical full shell model calculation will find more states in the same energy region for the same nucleus. What is more, these states are by their very existence physically interesting; they do not form part of some simple group structure or other, but may form a bridge between two such schemes, and could not be found by either. Meanwhile, the shell model will, in principle, find all of the states missed by these various truncation schemes.

Another aspect of models of the nucleus is how fundamental they are.

In other words, how much information is taken from knowledge of the 'true' nuclear reaction (of which, more later) and how much is 'empirical', designed to achieve the correct results by changing parameters. A typical example of the latter is the I.B.M., which has its parameters tuned for whichever nucleus is under study, and does not give any information about nuclei of neighbouring masses (with the exception of mirror nuclei). It is not true to say that this model just returns to you the information you put in to it; more information comes out of the model than is needed to tweak its parameters. But it is in stark contrast to the shell model, where many levels are obtained in nuclei throughout the sd-shell with a single set of parameters derived from the bare nucleon-nucleon interaction.

This is not to say that the shell model is our most fundamental theory of the nucleus, but it forms a necessary bridge between Q.C.D. at the deepest level, and collective models such as those described above, which are necessary to make calculating nuclear properties tractable.

1.4 The Glasgow code

We have so far described a range of models which describe truncations of the usual shell-model space. What, then of models which are equivalent ? In this section, I describe the J-scheme and m-scheme, and the reasons for the inception of the original glasgow code.

While the m-scheme, which I shall describe shortly, is simpler to use for simple problems, the great bulk of shell model theory until the mid 1970's, and much of it since, concerned the J-scheme (also called jj-coupling). In this scheme, each many-body basis state is a linear combination of states so formed as to have good angular momentum quantum numbers (usually, isospin

quantum numbers are included here too.) Thus, we have

$$\text{Basis} = \{\phi_{J,M}\} \tag{1.4.1}$$

This is useful since the Hamiltonian also has rotational symmetry, so that

$$\text{for arbitrary i, } \mathrm{H}\phi^i_{J,M} = \sum_n a_n \phi^n_{J,M} \tag{1.4.2}$$

Hence, we need only consider states with one value of J at a time. This allows us to reduce the number of basis states by a factor of 10 or so (relative to the m-scheme). This Hamiltonian is usually expressed as two-body matrix elements between two-body states coupled to different values of J. At this point, we run into the problems with the J-scheme :

- The basis states must be decomposed into coupled pairs of particles coupled to the remainder of the state before we can use 2-body operators. This is not simple.
- New states constructed after operation with the Hamiltonian must be explicitly made to be antisymmetric.

These two problems are taken into account by introducing new coefficients for antisymmetric decomposition and reconstruction of the state being operated on, called coefficients of fractional parentage. Forming these turns out to be the half the battle in a typical J-scheme calculation, and unfortunately the recursive algorithm that is used to generate them is numerically unstable for large numbers of particles. The problem could be summed up by saying that Racah algebra (the algebra of angular momenta) is an 'unnatural' thing to ask a computer to do. This begs the question: what, then, is a 'natural' thing for a computer to do? One answer, which I have been leading to, is the m-scheme as used in the Glasgow Code.

1.4.1 The m-Scheme

In the so-called 'm-scheme', every many body state of the nucleus is represented by a Slater determinant. In second quantized notation, it is sufficient to write , e.g.

$$\phi_{1,3,4} = a_1^\dagger a_3^\dagger a_4^\dagger \mid 0\rangle \tag{1.4.3}$$

(here $a_n^\dagger$ denotes a creation operator, while $\mid 0\rangle$ denotes the vacuum state), where the creation operators (of filled single particle orbitals) in the state are in some canonical order.

Since the Slater determinant is the sum of all permutations of the products of the single particle wavefunctions involved – with appropriate phases on each term – it is uniquely identified by its first term, up to a phase. It is this that we write as the second quantized notation for the state, with the canonical ordering defining the phase. Now, each filled orbit has associated with it some angular momentum and its projection on the z-axis, ($\mathbf{j}m$). The Slater determinant formed above does not necessarily have a good angular momentum quantum number, **J** but it does have a good projection of angular momentum, namely

$$\mathrm{M} = \sum_i m_i \tag{1.4.4}$$

Where i runs over all occupied orbits. Since the Hamiltonian, as before, is rotationally invariant, it preserves this quantum number, and we need only consider a basis constructed of Slater determinants of one value of M. Further, since the Hamiltonian has no preferred direction, there are no matrix elements which depend on M. Hence, a calculation done with any particular value of M, say, M_1, which is less than M_2, will contain all of the states in the calculation using M_2, up to a rotation. Expressed using the ladder operator for total angular momentum:

Quantity	2^{nd} quantization	Binary							
$\phi_{1,3,4,6,7}$	$a_1^\dagger a_3^\dagger a_4^\dagger a_6^\dagger a_7^\dagger \mid 0\rangle$	1	0	1	1	0	1	1	0
destroy 1,7	$a_1 a_7$	1	0	0	0	0	0	1	0
create 2,5	$a_2^\dagger a_5^\dagger$	0	1	0	0	1	0	0	0
$\phi_{2,3,4,5,6}$	$a_2^\dagger a_3^\dagger a_4^\dagger a_5^\dagger a_6^\dagger \mid 0\rangle$	0	1	1	1	1	1	0	0
$row1 \wedge row2 \wedge row3$		0	1	1	1	1	1	0	0

Table 1.1: An Operator in Binary

$$J_-\{\phi_{J,M+1}\} \subseteq \{\phi_{J,M}\} \tag{1.4.5}$$

Why is this representation any easier for a computer to understand? Well, in the above example, an equivalent operation to writing down a list of filled orbits would be to write down a list of all orbits in some model space and then specify which ones are filled.

$$rep(\phi_1) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a_1^\dagger & & a_3^\dagger & a_4^\dagger & & a_6^\dagger & a_7^\dagger & \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \tag{1.4.6}$$

In equation 1.4.6, filled orbits are identified as 1's, empty orbits as zeros. The connection with computers is now somewhat clearer, since every number is represented internally as a string of 1's and 0's, representing Slater determinants this way is completely natural. Of course, it is only useful if operations on such basis states can be done in an efficient manner. This is indeed so. consider the action of a matrix element

$$\langle \phi_1 \mid H \mid \phi_2 \rangle = \sum_{i,j,k,l} \langle \phi_1 \mid a_i^\dagger a_j^\dagger a_k a_l \mid \phi_2 \rangle \tag{1.4.7}$$

Where i, j, k, l run over all possible combinations that give the result. In binary, the representation of the element of the sum with $i = 2, j = 5, k = 1, l = 7$, looks like table 1.1.

In the last row of 1.1, I have written the result of
(row 1)$\wedge$(row 2)$\wedge$(row 3), meaning the binary bitwise XOR operation on the binary column of each of these rows. It is obvious that this, and the result we desired, in row 4, are identical. This is true in general; In other words, a two-body operator can be easily implemented using logic operations that are extremely fast on a computer.

1.4.2 Parity Representation.

This is not the only computer-friendly representation of an m-scheme state. There have already been described in the literature [18] methods of packing the m-scheme representation above into less bits by using properties of the binomial coefficients; more important for the present application is another representation which allows the efficient computation of the phase of matrix elements.

Because of the size of the matrix involved in the Glasgow code, we cannot store it; and, for the size of problem we envisage tackling (see section 2.2.3) it would not even be possible to use the usual sparse representation of element and coordinate. We are left to effectively construct the many-body matrix elements from the two-body matrix elements as we go. Consider such a matrix element:

$$\langle \phi \mid a_i^\dagger a_j^\dagger a_k a_l \mid \phi' \rangle \tag{1.4.8}$$

There is no guarantee here that the creation and destruction operators are in the canonical order required for our list of Slater determinants. The swapping of creation and destruction operators around necessary to produce simply a set of creation operators in such a normal order introduces a phase which we must know.

Fortunately, there is a representation of the m-scheme equivalent to that

Quantity	Occupancy Rep.								Parity Rep.							
ϕ_5	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
ϕ_3	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1
ϕ_1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
$\phi_{1,5}$	1	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0
$\phi_{1,3,5}$	1	0	1	0	1	0	0	0	0	1	1	0	0	1	1	1

Table 1.2: Different representations.

Quantity	Parity Rep.							
$\phi_{1,3,4,6,7}$	0	1	1	0	1	1	0	1
destroy 1,7	0	1	1	1	1	1	1	0
create 2,5	0	0	1	1	1	0	0	0
$\phi_{2,3,4,5,6}$	0	0	1	0	1	0	1	1
Result of XOR	0	0	1	0	1	0	1	1

Table 1.3: Parity Representation

given in section 1.4.1 which enables quick calculation of the phase. It is easiest to explain the 'phase' representation by example, such as that in table 1.2.

In this example notice how the string of 1's in the parity representation changes to a string of 0's, and back again, immediately after each occurrence of a 1 in the occupancy representation, as we move from left to right. Notice also, that the parity representation of $\phi_{1,5}$ is in fact the XOR of the parity representations of ϕ_1 and ϕ_5. This is again true of parity representations in general. Now let us consider the action of a destruction operator on the parity representation of the state. We use the example of the last section: see table 1.3.

The result of repeatedly XORing the parity representation of the initial state with that of the operators is seen to be identical to the parity representation of the final state, as required.

Of what use is this? It turns out that calculating the phase we require is equivalent to counting the number of bits set between those changed by the operator, and deciding whether this number is odd or even. It is obvious on inspection that we can tell this parity of the number of bits set by inspecting the first and last of the bits in our range from the parity representation.

If these two bits are the same, then the parity is even; if they differ, then it is odd. It turns out that we have to loop over the bits of the occupancy representation anyway in the program, when we first come across a state to be operated on; the parity representation can be economically constructed at this point, while the parity representation of the operators requires a trivial amount of storage. With both pieces of information, the phase calculations are simple.

1.4.3 Advantages of the J-Scheme.

If the m-scheme method is so much better than J-scheme codes, then we must ask why such codes persist. The answer, of course is that every scheme has its drawbacks, and the price we pay for simplicity of computation in the m-scheme is greatly increased basis sizes. Consider the *sd*-shell. In this model space, the largest J-scheme basis that must be considered is around 20,000 states, while the largest m-scheme basis that must be considered is 93,000 states. Thus, storage on a computer quickly becomes a problem, and in particular, the Hamiltonian must be stored in a compact fashion. In the Glasgow code, it has in the past been stored in the form of all non-zero matrix elements (as is common in sparse matrix codes). In the present code it is stored as two-body uncoupled matrix elements, since even a sparse version of the many-body matrix soon becomes too large to handle.

The size of storage involved and its attendant difficulties do wipe out some of the gains of using a 'computer-friendly' representation. However it has been

found that compared to the best J-scheme code, (OXBASH [19], although this is in fact something of a hybrid) the J-scheme codes perform best on smaller problems, while on larger problems the Glasgow code is the best one to use [20].

1.5 Reasons for Parallelization

There are two pressing reasons why the Glasgow code should be ported to a parallel machine. They are:

- There is not enough room on any economically viable serial machine to store the vectors required;
- Given *any* serial chip, we can take N of them and make a machine that is in principle N times faster. This is the simplest route to speeding up any computer.

Parallel machines offer the double bonus that every time you add a chip, the machine runs faster, but also they have increased memory capacity. In the Glasgow code, these are not just desirable, but prerequisites of performing calculations beyond the *sd* shell.

There is also a good reason why the Glasgow code is considered first for parallelization and not the J-scheme codes. This is, simply, that the c.f.p.'s are computed in an iterative manner. In other words, at each step, the result of the previous step must be known, so the procedure is inherently serial. It could be said that the same Lanczos algorithm is used in both codes for diagonalizing matrices, and the argument could be put that the c.f.p. problem can be recast as a matrix problem, so that standard parallelizing solutions could apply. The problems with these arguments are connected: the Lanczos algorithm does not need to run to completion to give us the results we need, whereas the c.f.p.

matrix must be completely diagonalized to give the information necessary for the next part of the calculation. This, combined with the numerical instability of the c.f.p. recurrence, renders it unsuitable for parallel architectures.

It is worth mentioning that OXBASH is being used for multi-shell calculations, in which it is in effect run in parallel: however, the way in which this is done is by diagonalizing the Hamiltonian for different J values, and different restrictions on the basis, on different computers – an essentially coarse grain parallelism which will not be suitable for very large calculations, as the problem cannot be subdivided many times this way.

Chapter 2

Parallelism.

> The purpose of models is not to fit the data but to sharpen the questions.
>
> *Samuel Karlin.*

Before we go on to describe how the Glasgow Code is parallelised, we discuss some general aspects of parallel programming which have some bearing on the work being done.

2.1 Common paradigms for Concurrency.

The applications normally converted for concurrent use are usually those for which the programming cost is small, because of some inherent parallelism in either the data or the algorithm. The most common of these have been wired into the hardware of some machines.

From our point of view, we have a number of processors (N) which must be connected together to form some network topology. The choice of paradigm for the concurrent code is the major determining factor in the choice of topology used. The choice of topology can be crucial, since in moving to a parallel architecture, we add a new operating overhead, the communication time. This

is very likely to increase as we add more processors to the network, and could swamp the gain in speed that we would hope for.

The architecture which we are programming for is a network of transputers, which are RISC-chips with built in floating point processors, and usually around 4MB of RAM, or roughly equivalent to a single 486 PC. They each have 4 1-bit wide two-way connections which can be used to build up a network of processors acting together. Our target is to write a program which will be able to run efficiently on a 1000 T-9000 transputer network. The actual hardware that we wrote the program for, and the machine on which we were actually able to run it, which turned out to have a much lower specification than we had hoped, are described in more detail in section 2.2.2.

We now go on to describe the different ways a parallel program can be constructed.

2.1.1 Task Farming.

The simplest programs to parallelize are those where large numbers of identical small tasks must be performed on many independent packets of data. This is quite a common situation, for instance, deciding whether a point lies in the Mandelbrot set[21] only depends on the co-ordinates of the point, and so completing this task for all points in some region, is one such operation.

Here, the division of labour is into a co-ordinating master task, and many copies of the small processing task (the 'slave'). Normally, there are many more data packets than processors, so a *load balancer* is required to keep the network as busy as possible. Since processing tasks only have to communicate with the master task, the topology used must minimize the distance between an arbitrary node and this master task. Binary and ternary trees are the usual such topologies which can be implemented on a transputer network.

2.1.2 Geometric Parallelism

This kind of parallelism often arises in physical problems. The idea is that if for, say, a problem where the value of some function at a point depends only on the data associated with the point itself and, the data associated with the nearest neighbouring points, then the space of points can be divided into regions, where the only dependence of each region on the others is at the points where it comes into contact with the neighbouring regions, e.g. boundary conditions must be met at the surface of volume elements or the perimeters of areas. The satisfaction of boundary conditions only requires that processors acting on neighbouring volume elements communicate the values at their surfaces to each other.

Here, obviously we have only nearest neighbour communication. This is commonly translated into a grid topology, where the volume elements are rectangular columns, or to a hypercube for problems of higher dimensionality (for example, Lattice Q.C.D.[22]). Since the perimeter of (say) a square area element is proportional to the square root of the area, as we divide up the area into smaller and smaller squares (so that we can use more and more processors), the communication time, which is proportional to the perimeter, increases much more slowly. Hence the communication overhead only increases slowly with increased numbers of processors.

To an extent (as we shall see) our program uses a geometric method of dividing up data. However, the function acting on the data is non-local (in the sense that many non-neighbouring points affect the value at each point) and this makes our task much more difficult than usual.

2.1.3 Algorithmic Parallelism

This is a catch-all phrase for algorithms which include some element which can be parallelized. The most common is when a serial process which has

several stages must be repeated again and again. Then, instead of having a 'jack-of-all-trades' process which performs all of the stages, before moving on, each stage is given a specialist process which can act at the same time as the others. The end result is much like a car production line. This method is called 'pipelining' and as the name suggests, the usual topology is a line of processors. This is used in almost all computer chips to speed up execution at machine instruction level.

The difficulty with algorithmic parallelism is that the more you want to divide the program up, the more programming work has to be done. In the previous two methods, the 'grain' of the parallelism (how far it can be divided up) lay in the *data*, and since we are usually looking at very large data sets when using highly parallel machines, it does not tend to be difficult to divide it up many times. In algorithmic parallelism, there are three distinct levels of grain: coarse, in which whole programs or large tasks are run concurrently, which is the way our program works; medium, in which the bodies of most loops are not iterated but performed simultaneously, with the task size being of the order of a few tens of statements, and fine grain, where small groups of machine instructions are performed concurrently. (Graining can also occur in data-parallel programs – for example, OXBASH is run this way, and the UNIX batch(1)[23] command is often implemented this way, but this is a sign that parallelism is really not appropriate for the program in question).

Some progress has been made over recent years in constructing compilers which automatically parallelize code, generally at the loop level, but problem-specific knowledge can almost always lead to a better algorithm. However, the problem remains that, at best, the program will be divided up into a few tens of tasks.

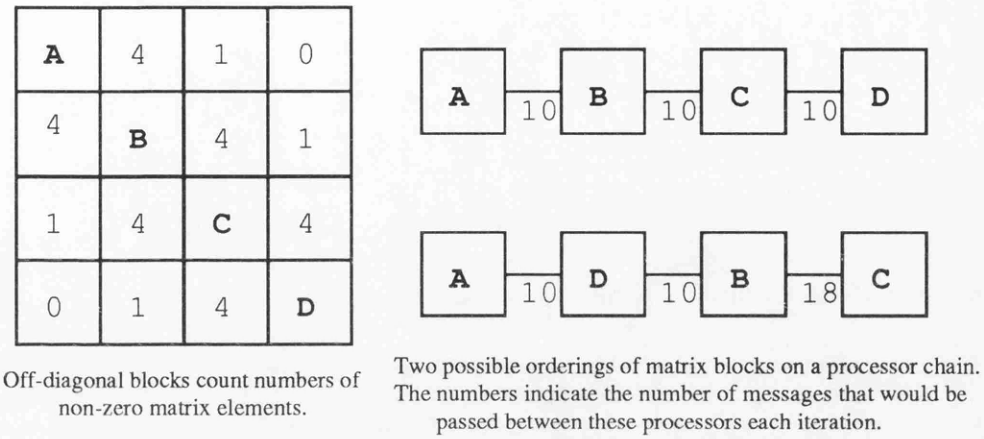
With the geometric and farming methods, you can in general increase the number of processors tackling the problem without any extra programming effort whatsoever. However, algorithmic parallelism does tend to produce the

greatest increase in speed for a program (since it makes use of knowledge specific to the problem, so-called 'superlinear' speedups can be achieved[24]). Because of this tradeoff, in general programs are divided up algorithmically first, then further subdivided along geometric or task farm lines. The Glasgow code has in fact been divided up algorithmically in the thesis by Riaz [25], into `operate` , `locate` and `master` tasks, as described in more detail in section 3.2.3. The present work is mainly looking at dividing the vectors operated on into blocks (which could be seen as geometric parallelism), described in section 3.2.4.

2.1.4 Topology issues

Since our division of the vectors requires that each `locate` processor can communicate with every other, the topology of the network must be designed to minimize the distance between an arbitrary pair of processors. It can be shown in fact that it is important given a `locate` task which operates on one part of the basis table, to have it physically close in the network to tasks operating on numerically close parts of the basis table (figure 2.1) since there are more matrix elements between them. We must also place the `operate` tasks in the network, and the obvious place to do this is adjacent to its corresponding `locate` task.

The topology proposed previously[26] was a chain of pairs surmounted by the `master` process. A major objection to this is that messages going 'up' the chain are always delayed by messages coming down, there is no alternative, less congested route to take. A consequence of this topology would be the last processor pair always lagging behind the first one, and the chances of deadlock occurring (see section 2.1.6) are high. The next more complex topology, a ring of processors, has almost identical properties. To provide better routing in the network, we would like to connect the opposite sides of the ring in such a



Off-diagonal blocks count numbers of non-zero matrix elements.

Two possible orderings of matrix blocks on a processor chain. The numbers indicate the number of messages that would be passed between these processors each iteration.

Figure 2.1: Effect of processor order in the network.

way as to minimise the distance between every pair of processors, and doing so leads to the optimal regular topology, the chordal ring. This is actually equivalent to a torus of processors (in the sense that the diameter and mean internode distance of each have the same N-dependence.) Hypercubes are not relevant in the present context as transputers do not have enough connections to construct one, and the structures where the so called 'Moore bound' for the minimum mean internode distance is achieved (such as the Petersen graph) do not exist for most values of N (the topologies decribed here are pictured in figure 2.2).

However, research has shown that networks whose sizes approach the Moore bound, for large N can be achieved by just connecting the processors randomly together (and possibly adjusting the resulting network slightly) [27]. Unfortunately, it would be difficult in such a random structure to preserve the information that we have that numerically close `locate` tasks should be as nearly adjacent as possible. Also, in some of the variants of C used in this project, a unique number has to be allocated for message transports between each and every processor,the table for which increases in size proportional to

N^2, becoming unwieldy for large values of N (remember, our target size is $N = 1000$). Regular topologies have the advantage that entries in this lookup table can be generated, rather than having to be stored.

A further problem arises when attempting to implement such point to point communication on anything other than the simplest topology. At the start of this project, no system was available that could transparently send messages from one process to another without a direct physical link. This is necessary, because otherwise the *source* of the task for each processor has to be altered to take into account the position it has in the network. This would in turn mean having 1000 slightly different compiled versions of the Glasgow code for our target machine, which is clearly unacceptable (this was, in fact, the way the code worked before the work of this thesis began).

A router task has to be placed on each processor in such a way that direct and indirect communication is not distinguished in the program. This is in itself an active research area, and at first we attempted to use TINY [28], which turned out to be prohibitively complex and unreliable, and a system (DynaLoader [29]) whose source code had to be altered, as it had been developed for a specific application very different from our own. While this work progressed, a new version of MEIKO C appeared which incorporated TINY in a simpler fashion. With this upgrade, our own attempts at creating a router were abandoned (indeed, versions of the code written up to this time were no longer able to run under the upgraded system).

With the difficulties in choosing a topology, and the additional overheads from point-to-point routing tasks, it was felt necessary to model the program on the network. This would tell us how the communication demands of the program would affect it as the number of transputers increased, and the size of the problem changed, and hopefully we could decide whether or not it would be worth the extra effort of programming, and the overhead at runtime it would entail.

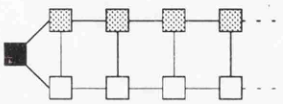
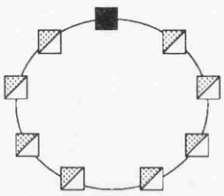
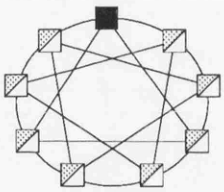
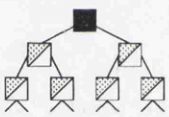
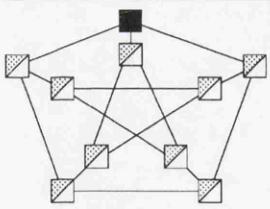
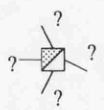
Network Topology	Diagram of Network.	Size-dependence of: Mean internode distance, d Diameter, D
Chain.		$d \propto N$ $D \propto N$
Ring.		$d \propto N$ $D \propto N$
Chordal Ring. (equivalent to torus)		$d \propto \sqrt{N}$ $D \propto \sqrt{N}$
Binary Tree.		$d \propto N$ $D \propto \log_2 N$
A `Moore Bound' topology. ( the `Petersen Graph')		$d \propto \log_{v-1} N$ $D \propto \log_{v-1} N$
Randomly connected.		$d \propto \log_{v-1} N$ $D \propto \log_{v-1} N$ (see text)
KEY :	■ Master. □ Locate. ▨ Operate. ▨ Operate/Locate pair (shown this way for clarity).	

Figure 2.2: N dependence of different networks

2.1.5 A Model of the Code

In this model of the code, we begin as is usual by dividing the time taken to run the program once into its 'serial' and 'parallel' portions.

$$t_T = t_S + t_P \text{ , on a single node.} \tag{2.1.1}$$

Where t_T is total running time, t_S is the time spent in the serial part of the code, and t_P is the time spent in the parallel part of the code. By serial, we mean every portion of the serial code that is still only run serially in the parallelized version of the code. In our case, this amounts to just the disk access time, and the time taken at the start of the program to set up the basis and Hamiltonian. Since the basis and Hamiltonian could be re-used, and the disk access time cut to virtually nothing by storing all vectors for reorthogonalization dynamically (see section 3.2.2), we can assume that:

$$t_S = 0. \tag{2.1.2}$$

On N nodes (and here we mean nodes in the general sense of an `operate -locate` pair forming a single node), we have :

$$t_T = \frac{t_P}{N} + t_C \tag{2.1.3}$$

the communication time, t_C is an added overhead of parallel systems. Taking the simple model of the parallel part of the code used in section 3.2.4, we have that:

$$t_T = \frac{pv^{k+1}}{N} + t_C, \text{ where } p, k \text{ are constants.} \tag{2.1.4}$$

Here, v is the dimension of the basis. We will now justify the assumption of this power law dependence. This was originally used to indicate, for a given basis state, the number of basis states with which it has a non-zero matrix element. This number must be less than v, since the matrix is sparse, and it seemed unlikely that it would remain constant as the basis size increased.

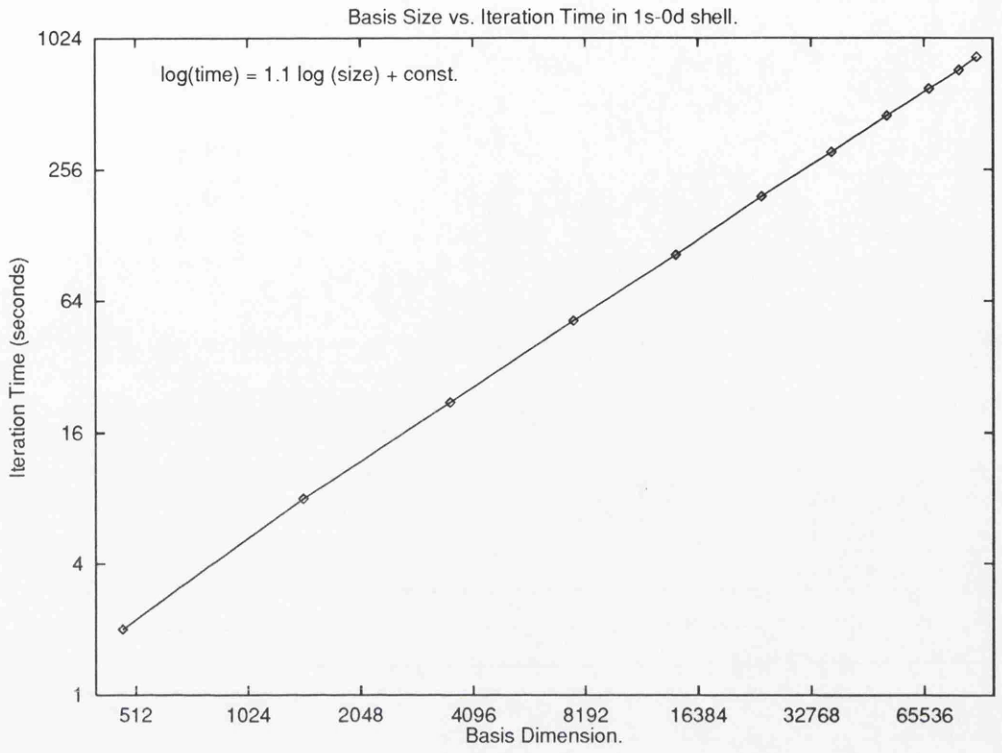


Figure 2.3: Dependence of Iteration Time on Basis Size

Consequently, the number of non-zero matrix elements in the entire matrix lies between v and v^2. The form v^{k+1} was postulated for some $0 < k < 1$. The speed of the code was then measured against basis size over the entire *sd-* shell. It was found that the time taken for each run satisfied very closely this power law, with $k = 0.1$. In fact, the relationship is surprisingly accurate over this range of three orders of magnitude, with a correlation coefficient of $r = 0.99$ for the regression. This is depicted in figure 2.3.

The communication time again depends on this measure of the sparsity of the Hamiltonian matrix, since every off-diagonal matrix element to a basis state, except those to the same vector block as the one from which the matrix element originated, require some communication to be done. An assumption will be made here that the number of matrix elements which map between each pair of processors is identical. There is some justification for this. In

figure 2.4 we see that outside the central peak, the number of matrix elements at each Hamming distance varies about a reasonably constant value over most of the matrix (the poor agreement at small distances can be ignored because no communication is done there, and at large distances the disagreement is because this plot is for a very small matrix. The graph is labelled 'fraction of matrix filled' as the scale is normalised so that the values would be 1 for every distance if the matrix were completely full). It could also be argued that this particular choice of model is a worst case for our code; it will tend to overestimate the communication time by expecting more communication to be done over longer distances. This gives the amount of time that a node spends on messages that originate or terminate at that node is:

$$t_{node} = c\frac{(N-1)}{N}\frac{v^{k+1}}{N} \tag{2.1.5}$$

But point-to point communication requires work to be done by routing tasks, which will increase the time taken up by the communication. It is often stated that communication time depends directly on the mean internode distance $\overline{d}$ in the network (measured in hops. see figure 2.5), but I have not seen this result *proved* for any case.

It may seem an intuitively obvious result, but the usual explanation, that the receiving node must wait for messages being passed on through other processors, is just as obviously false. The reason is that the receiving task can perform other tasks while waiting for its communication channels to become active; indeed, good programming would demand that this be the case. The delay is not, in fact, at the originator of the message, or at the receiving end, but on all of the processors in between. That this produces the observed effects in our case is shown next.

We make one final assumption: that no message is produced in the network which is not received. This is entirely reasonable, since otherwise we would be doing the work of producing these dummy messages for no reason. Combining

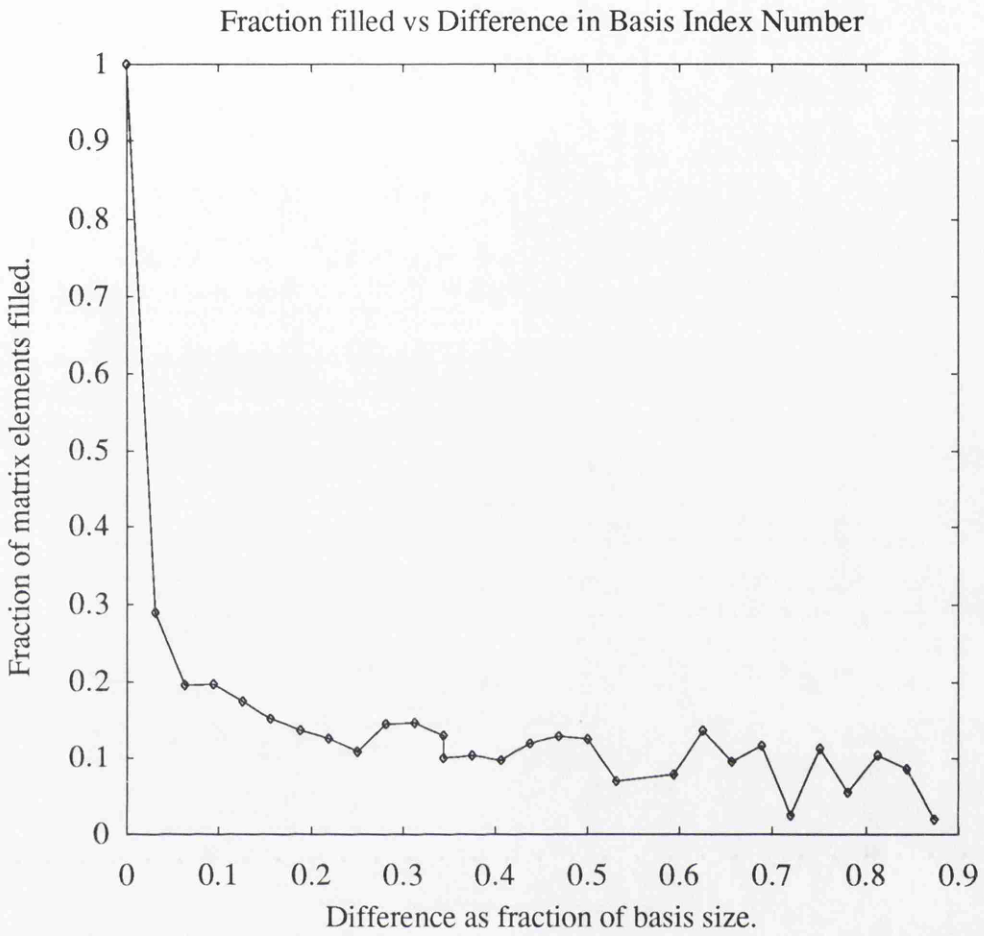


Figure 2.4: Number of Messages vs. Distance travelled, for 640 states

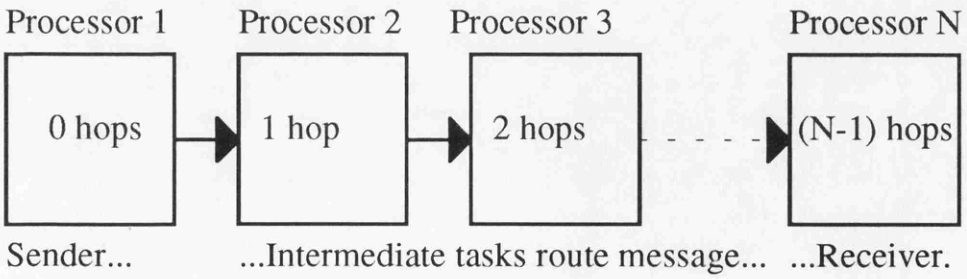


Figure 2.5: Measuring distance in a network

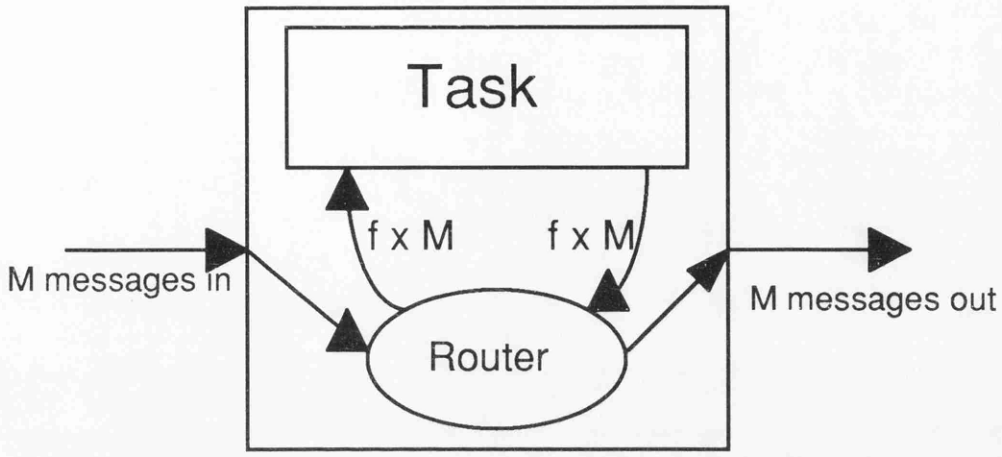


Figure 2.6: A Typical Processor.

this and our assumption of homogeneity across the network tells us that each processor receives as many messages as it sends (and that this number is the same on every processor).

Consider a single processor in this network (figure 2.6). It has at least two tasks active on it; a routing task, to redirect messages intended for other processors, and its 'useful' task. Some fraction f of the messages that arrive at this processor terminate there, and we have shown that an identical fraction must be sent out also. The amount of communication time in total on this processor is then:

$$t_C = c(\frac{1-f}{f} + 1) \tag{2.1.6}$$

Where c is the time spent on communication which originates or terminates at this processor. There is an intimate relation between f and the mean internode distance, given the current set of assumptions. We now look at our processor in the context of the network:

As the messages from the 'central' processor in our homogeneous network propagate outward, they act in the manner described above, passing through

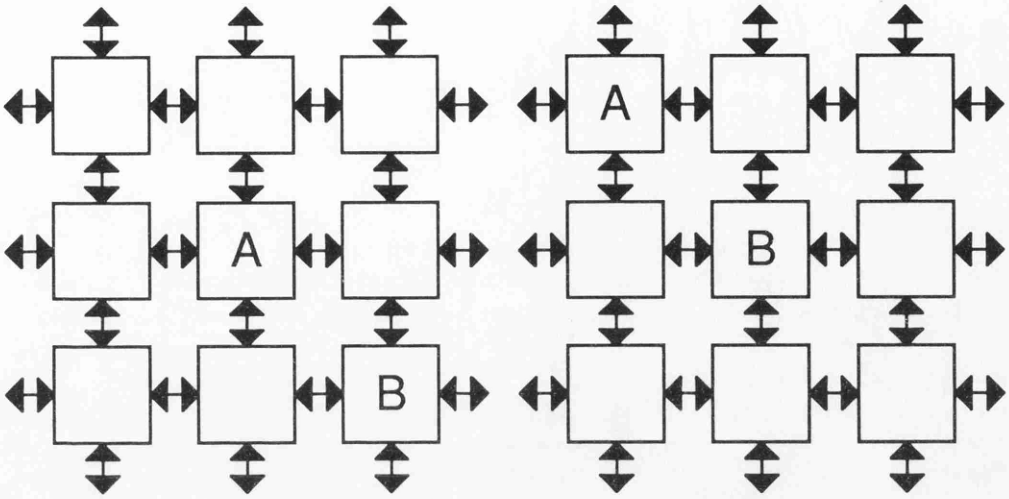


Figure 2.7: The Network is the same, viewed from any processor

each processor's routing task on the way. Since our central processor communicates equally often with every other processor, the number of processors affected on average by a message is precisely the mean internode distance. This means that each processor, on average, sees $\overline{d}$ messages that were not intended for it for every message that was intended for it. And since our central processor is 'the same' as all the rest in precisely this sense, we obtain the result:

$$f = \frac{1}{\overline{d}} \tag{2.1.7}$$

This is immensely useful, since although f is the factor that is the *real* measure of how much point-to-point communication slows down a program, the value of $\overline{d}$ is known for arbitrary N for a wide variety of network types.

Incorporating this result into our formula, we finally get the result:

$$t_T(N) = \frac{t_P}{N} + c\overline{d}(N) \tag{2.1.8}$$

and by assumptions about the form of the matrix that we have made:

$$t_{total}(N) = \frac{v^{k+1}}{N}(p + \frac{c\bar{d}(N-1)}{N}) \tag{2.1.9}$$

This formula is more useful if we use the quantity known as the 'Speedup', which is simply:

$$Speedup = \frac{t_{total}(1)}{t_{total}(N)} \tag{2.1.10}$$

If we write down the reciprocal of this quantity, and make some suitable simplifications, we get:

$$\frac{1}{Speedup} = \frac{1}{N}(\frac{c(N-1)}{pN})\bar{d} \tag{2.1.11}$$

But since p and c are constants, this value can be easily manipulated to give us the topology-dependence of a network. This provides a simple check on the model. We use this in the next chapter to examine the topology-dependence in our program.

2.1.6 Communication Breakdown.

There are two more effects that occur in parallel processes which deserve a little attention before we move on, and which are left out of the above model. The first is message contention, or 'blocking', in which a process tries to send a message, but the router is actively sending one at the time. This increases the communication time, and what is more, simulation shows that the effect increases with time. There are natural 'breaks' in our program at the end of each iteration which allow the network to catch up with itself, but this might become worrisome in the long iterations required by the large calculations planned.

This effect was investigated primarily to check the assertion that there is an 'optimum' message size. It is obvious that since each message carries a small

header, then if we decrease the message size the amount of useful information passed in a given amount of communication time is reduced. It was suggested that large message sizes increased contention, and thus there was some middle ground where communication became most efficient. The simulation of this process in a simple model, and test programs on the Parsytec Multicluster, seem to show that this fear was unfounded, and that there is in fact no limit to the size of message that should be used.

The final problem that can occur is deadlock. This is the phenomenon that occurs when two processors are trying to send or receive from each other simultaneously. Both processors end up waiting forever to hear from the other. This example is fairly easy to circumvent, but it is fairly easy to construct more complicated problems, where a large number of processors are waiting. For example: at a round dinner table, everyone turns to their left to talk to their neighbour, and finds themselves looking at the back of someone's head. If they all have the same 'recovery' algorithm, after a few seconds they will all turn to their right, and discover themselves in the same predicament, and so on.

It is possible to construct nearly deadlock-free routing systems, and this is a continuing area of research in Computer Science. However, these systems are much slower than those such as TINY or PARIX used in the present work. Indeed this was a design choice in the writing of PARIX: if you suffer from deadlock, it's probably due to bad coding, and it's up to you to fix it. If you don't then your program will run much faster than if it isn't trying to prevent deadlock.

Our code suffered from apparent cases of deadlock for particular input values and not for others. This was apparently alleviated by increasing the message size (and therefore reducing or simply changing the number of messages in the network.) Until some better way is found to circumvent this problem, the program will leave the choice of message block size in the hands of the user for this reason.

2.2 A Tractable Problem?

As suggested already, before we decide to make any attempt to parallelize the Glasgow Code, we must first discover if it is worth doing so. By worth it, I mean in particular, will the program run in a reasonable amount of time, using a reasonable amount of memory, on a reasonable number of chips? We consider each point separately.

2.2.1 A 'Reasonable' Time

Before we can decide the question of whether the program can run in a reasonable amount of time, we must ask what that time is. We choose, fairly arbitrarily, two days. There are a number of good reasons for this, viz.:

- Computer stability time. Computers, especially high end ones, sometimes resemble the British weather. "If it's raining today, it'll be raining tomorrow" applies equally truly to whether or not we can expect a system to be active tomorrow. However, we can rarely say with any confidence that we can guarantee the computer will not break down any further into the future. As I write, a straw poll of Unix workstations (a relatively stable system on this campus) indicated uptimes of seven to twelve days. Thus, a two-day limit gives us a reasonable margin of safety.

- Urgency of getting results. The shell model codes are not just intended as theoretical niceties, but as tools for confirming or directing experiments. The results are often asked for with a deadline a few weeks, or at most a month, away, to satisfy the pressing need to publish. A breakdown in the program over the course of two days, or badly formed starting data, for instance, can be corrected; but if the error occurred sometime during a week-long run it would become more difficult to trace, and a large fraction of the time remaining to complete the research would have

been lost.

- Finally, it is simply the length of a weekend. It is still true that most people run their programs interactively during the week, and the weekend is the only time that the computer system is free enough to support the intensive calculations we require. While the computer itself may not be in use during the week, large hard disks tend to be a shared resource over several computers, and it has been found by experience that users doing normal work on other machines attatched to the disk can as much as double the computing time.

A second aspect of 'reasonable time' is how the program scales with the size of the problem, i.e. it may run in less than two days for an s-d shell calculation with $< 100{,}000$ states, but will it do so for an (as yet untried) calculation with a million or more states? The usual criterion for judging intractability in this sense is the description of problems as 'P' or 'NP' [30]. The letters here stand for 'running to completion in (Non-) Polynomial time, in terms of some measure of the size of the problem'. NP-problems are considered insoluble, since the time taken by the calculation grows so much faster than the size of the problem.

So, is our problem NP? The answer to this really depends on how you measure the size of the problem. If you measure it in terms of the number of active shells in the calculation, the answer is — almost certainly — yes. The number of states in the calculation grows combinatorially with the number of shells (for instance, the number of states when half of the orbits are filled roughly doubles with the addition of each *orbit*), and the time depends in turn on a low-order polynomial of the number of states (see section 2.1.5). This realization has led to probabilistic approaches to the nuclear shell model coming into vogue again, particularly the Monte-Carlo method [31], which I will discuss further later on, but also various methods inherited from chaos

theory and statistical thermodynamics for determining level densities [32].

There are, however, very good reasons why these methods are not always applicable; the particular example of transition rates will be discussed in my conclusions with reference to the work of Koonin et al [31]. The next question we ask is, is the model tractable even if we measure problem size by number of states? Since we have already lowered our sights somewhat, we also note that even a quadratic time dependence is probably too much for us; but this is a matter of economics discussed in section 2.2.3. The actual calculation of the time dependence that occurs is the subject of the following chapter.

2.2.2 'Reasonable' Amount of Memory.

In their book, *Computers and Intractability*[30], Michael Garey and David Johnson comment:

> It is useful to begin by distinguishing between two different causes of intractability allowed by our definition. The first, which is the one we usually have in mind, is that the problem is so difficult that an exponential amount of time is needed to discover a solution. The second is that the solution *itself* is required to be so extensive that it cannot be described with an expression having length bounded by a polynomial function of the input length.

The problem of memory used, as here described, is closely tied to the NP-ness of problems. In particular, again, if we regard the input data as being the list of shells and the Hamiltonian for the nucleus in question, the output, i.e. the eigenvectors in the m-scheme, are almost obviously going to be intractably big. Again, however, we must look at how the problem scales with the number of states involved instead. Here, one of the reasons for choosing the Lanczos algorithm is its requirement of only $O(2n)$ units of storage, where n is the dimension of the basis space. However, the requirement

of reorthogonalizing the Lanczos vectors, as we shall see in the next chapter, forces us to set aside $O(jn)$ units of storage, where j is the number of Lanczos iterations. Thus, a limit is put on both the basis size *and* the number of iterations, an uncomfortable situation.

The actual amount of memory available to us is again a matter of economics and chip design; this is discussed in the next subsection.

2.2.3 'Reasonable' Number of Processors.

What we consider 'reasonable' in this context runs along the lines of the biggest machine that we could get to use in the next few years – the largest machines available in the world today, will, in the near future, be the processing power available to academia and the public economically.

A pertinent example of this is the supercomputer constructed at Southampton University consisting of 1000 T-200 transputers. These processors, the first transputers, were once considered state-of-the-art. Now it is possible to buy them in bulk for around $50 U.S., and thus the particle physics group were able to buy the workings of an extremely powerful machine for the price of around 10 workstations!

Other large machines which are, or will soon be, available, are Shell's Parsytec Supercluster machine, based around 1000 T-800's [33]; the Connection Machine in Edinburgh, with around 65,000 smaller chips; the Meiko surface in Edinburgh, with 512 T-800's [34] and the planned GC-5 Parsytec machine with 65,000 T-9000's (which is reportedly now to be constructed from a hybrid of T-800's and Motorola-supplied chips.)

To make the comparison fair here it should be said that the Connection Machine's chips are not specifically designed for parallelism, although the surrounding hardware is; but T-800's are slower processors than those used in that machine.

Other parallel machines have been constructed recently using the more recent DEC-Alpha and TI-C40 chips which outstrip all of the T-800 machines used above. However, keeping our feet on the ground, it seems that a figure of 1000 transputers is not unreasonable to ask for. Shell's Parsytec supercomputing centre has, in fact, expressed an interest in the development of the Glasgow Code.

Most T-800's now used have a 4 Megabytes RAM, although 16M chips are available. This gives us a possible memory of 4 Gigabytes on our 'reasonable' machine. Experience of other such computers has shown that a roughly equivalent amount of disk space is also the most that we can expect.

When this project was begun, we believed that an extremely fast transputer based machine was just around the corner. The T800 transputer was already dated when we began this project, and Intel claimed to be ready to release the T9000 replacement - supposedly at least 10 times faster, with a hard wired virtual topology (i.e. messages are routed in hardware) and 16M memory as standard. The deadlines for Intel's release of this chip came and went, partly because of a worldwide shortage of silicon, which delayed the development of the Pentium and ARM3 chips around the same time. The T9000 now looks to have been all but abandoned after years of delay.

Chapter 3

Implementation.

> But forms of thought move in another plane,
> Whose matrices no natural forms afford,
> Unless subjected to prodigious strain;
> Say, light proceeding edgewise, like a sword.
>
> *Donald Davie, 'Gardens No Emblems'.*

3.1 Introduction

In this chapter I will describe the workings of the program in the aspects in which they are different from the previous serial versions of the code. There are two aspects to this: firstly, we must deal with the fact that the calculations we plan to do are bigger than any previously attempted, which creates problems of itself, and secondly, we must parallelize the program.

3.2 Concurrency in the Glasgow Code.

3.2.1 The Lanczos algorithm.

Since we are going to divide the program up first along algorithmic lines, it is essential to understand what the algorithm does. The Lanczos algorithm proceeds by operating repeatedly on a vector with a real symmetric matrix, removing at each step the parts of the vector parallel to vectors already obtained in the process, like so:

$$\begin{aligned}
\mathbf{H}.\mathbf{v}_1 &= \alpha_1\mathbf{v}_1 + \beta_1\mathbf{v}_2 \\
\mathbf{H}.\mathbf{v}_2 &= \beta_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \beta_2\mathbf{v}_3 \\
\mathbf{H}.\mathbf{v}_3 &= \beta_2\mathbf{v}_2 + \alpha_3\mathbf{v}_3 + \beta_3\mathbf{v}_4 \qquad (3.2.1) \\
&\vdots \\
\mathbf{H}.\mathbf{v}_n &= \beta_n\mathbf{v}_n + \alpha_{n+1}\mathbf{v}_{n+1} + \beta_{n+1}\mathbf{v}_{n+2}
\end{aligned}$$

$$\text{where } \mathbf{H}^* = \mathbf{H}, \text{ and } \forall n : \mathbf{v}_n^T\mathbf{v}_m = \delta_{m,n}. \qquad (3.2.2)$$

Effectively, we are constructing the matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ to perform a similarity transform on H (which is fully constructed when $n = \dim(\mathbf{H})$). The resulting matrix is the tridiagonal matrix formed by the αs and βs in the above equations. As this is a similarity transformation, the eigenvalues are preserved.

However, the Lanczos algorithm has the unique property that as the number of iterations increase, the extreme eigenvalues of the tridiagonal matrix formed at each stage converge quickly on the extreme eigenvalues of the full matrix. Typically, you need only 100 iterations to obtain good approximations to the lowest ten or so eigenvalues of a matrix of *any* size. This property of the algorithm makes it ideal for the nuclear structure problem, where only the lowest states in the energy spectrum are clearly not collective in nature, and hence are of interest to shell-model theory.

Another aspect of the algorithm is the relatively small amount of space it needs to run. At each iteration, only two vectors and the matrix need be stored. If the matrix itself is very sparse, or comes from some generating function, the storage required is much reduced, to the level where very large matrix problems can be tackled.

The other algorithms commonly used to diagonalize matrices are the Givens, and Householder methods[35], both of which require the full matrix to be stored. They can be adapted for sparse matrices, to reduce the effect on the storage required, but neither method has the convergence property of the Lanczos algorithm.

3.2.2 Reorthogonalisation.

Unfortunately, the Lanczos algorithm is not so ideal in real life. Small roundoff errors at each stage become magnified by the iteration, and eventually the vector being operated on is no longer orthogonal to vectors that were generated near the start of the process. One solution, that used in the current code, is to re-orthogonalize the current vector to all previous vectors at the start of each step. Since the vectors in our problem are extremely large, we at first stored them on disk. However, because hardware limits us to having only one node with disk access (the master) this leads to a bottleneck at the end of each iteration. The alternative is to store all the vectors generated dynamically, which uses huge amounts of memory. Typically, we want 2×10^7 floating point numbers in each vector and 200 vectors, or 16×10^9 bytes. This represents the *entire* storage capacity of 400 transputers leaving no space for the program or other data.

In fact, full orthogonality is not necessary. Eigenvectors of identical numerical accuracy can be obtained by a variety of methods, some of which work with the Lanczos vectors (the iterated vectors) and some using only the con-

verged eigenvectors. We have investigated the possibility of implementing such a scheme, and the best that we can hope for by using these schemes (which require a small amount of extra storage, and a relatively small calculational overhead) is a reduction in the storage by 4/5. This is a lot - a saving of the equivalent of 320 transputers in the above example - and will require to be implemented if the program is to achieve its full potential. However, in the interests of completing this project on time, it is noted that just switching to full dynamic storage has a similar effect on the *speed* of the program, while the storage used by the program does not affect the timing at all, in our test calculations. Hence, what follows will remain true when the program eventually has a full semi-orthogonalization scheme.

3.2.3 The `operate` /`locate` division

The program is first divided up into three tasks: a `master` task which coordinates all operations, and handles disk access, and two other tasks, which we call `operate` and `locate` , which perform the matrix multiplication.

The `operate` algorithm is, essentially:

get vector v from nearest `locate` task.
for each basis state i in vector v_n
  for each pair p of particles in i
    destroy p in i to get s
    for each pair h of holes in s
      create particles to replace holes h. Label new state j.
      figure out H_{ij}, the matrix element
      send $H_{ij} \times v_{n,i}$ to `locate` block dealing with j.
tell `locate` that we have finished.

Note that at this stage there is only one 'block', which stores each complete vector. $v_{n,i}$ is the value assosciated with state i in the nth Lanczos vector.

The `locate` task, on the other hand, does several different things. Depending on the stage of the iteration we are at, it jumps to a different command as follows:

get command from `master` . Do command:
locate:
    send last vector to local `operate` task.
    while more packets are arriving:
        read state j and contribution $H_{ij} \times v_{n,i}$
        find state j
        if it exists, add $H_{ij} \times v_{n,i}$ to $v_{n+1,j}$
        else discard this contribution
        tell `master` that we have finished.
scalar product:
    get vector numbers m, n from `master` .
    return the contribution to $v_m^T v_n$ from this block.
add vectors:
    get vector numbers m, n and scale factor r from `master` .
    add $r \times v_n$ to v_m.
create vector:
    allocate space for a new vector.

The operations on the `locate` task, when used in the correct sequence, can normalize, or reorthogonalize, the vectors stored. Note that the implementation differs somewhat from the simple algorithms described above, but the general plan of the program is as described. Most of the time on a `locate` processor is spent in the locate task, and in the main we will refer to this task as if that were all that it did.

It has been found empirically that the `operate` and `locate` operations

take similar amounts of time, so that maximum efficiency, i.e. the minimum of C.P.U. idle time, can be achieved by placing these tasks on separate processors. Although this was achieved in test situations in the previous thesis, it has not been pointed out that these two processes should scale identically with basis size if we are to maintain this efficiency with larger problems. This might at first be thought to be the case, but in fact the `locate` task becomes more efficient as the basis size is increased. This allows us to put any extra small tasks which are likely to increase in load as the problem size increases – in particular communication, on to the `locate` task. The reasoning behind this statement is made clear in the next section.

3.2.4 Division of Vectors

It is seen from the description of the `operate` task in the previous section that the operation on any single basis state is independent of all the others. Thus, we can group the basis states into blocks, and operate on the blocks independently. This becomes an imperative anyway as the basis size increases, since there is a limit to the size of vector a processor can hold. This memory imperative also demands that we split up the basis table held by the `locate` tasks.

The results of these operations are not so simple to process, however. Each matrix element can map one state to any other, so if the basis table is stored in blocks as well, we have to first search to find out which block the resulting state belongs to, and since each block is stored in a `locate` task on a separate processor, we must be able to communicate with an arbitrary processor. If too much communication must be done, the topology of the network will probably be important in the runtime of the code. Hence we must devote some attention to finding out if this will be the case, and secondly finding the best topology for our purpose.

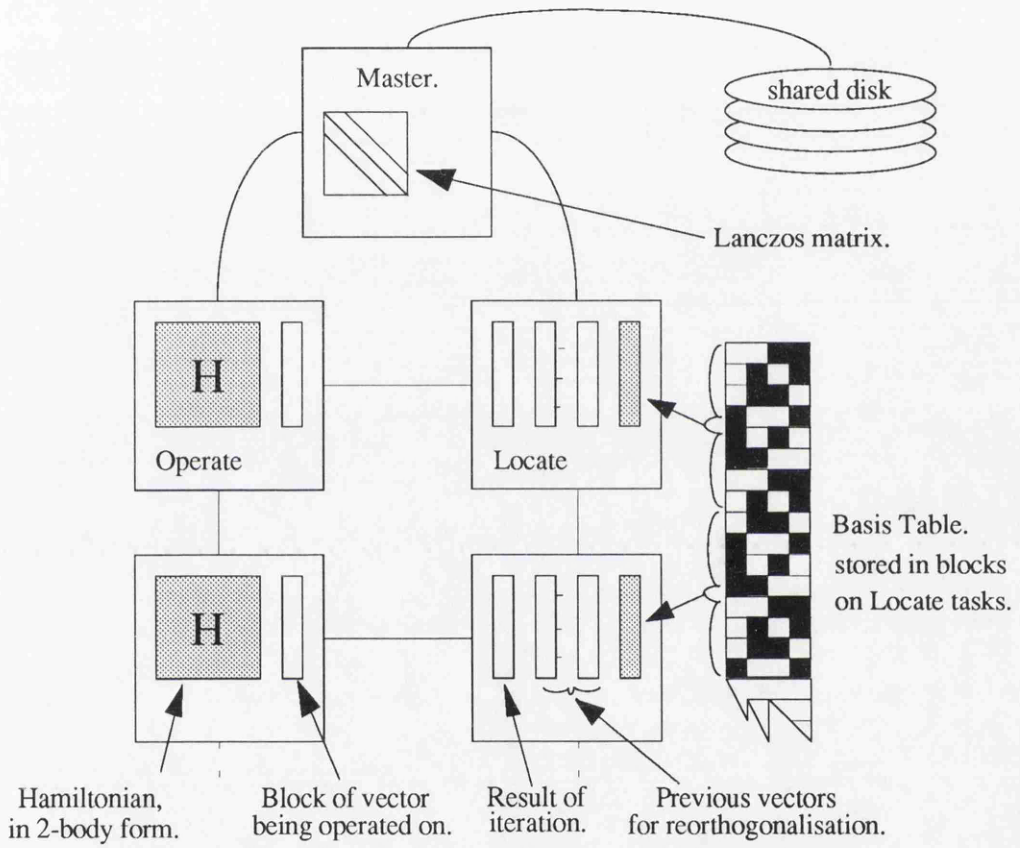


Figure 3.1: Schematic representation of the Parallel Code

Before I leave this section, this is the best place to comment on the scaling of `operate` and `locate` tasks with basis size. Now that we have divided the vector up, it is clear that each `operate` task represents a mapping from a vector block to some semi-random section of the full vector:

$$\frac{v}{N} \mapsto \frac{v^k}{N} \text{ where } N \text{ is the number of } \texttt{operate -locate} \text{ pairs.} \qquad (3.2.3)$$

(The reason for the choice of a power law for the second term was explained in the previous chapter) . Since no contribution is 'lost' by the network, the average number of contributions received by any `locate` task is v^k/N. These, in turn, are mapped back on to the block of basis table that the `locate` task holds, which is of dimension v/N. At first sight, as I said, this might seem as if the two processes scale identically with v, being proportional to v^{k+1}/N^2. However, only the `operate` task must loop over all elements of its basis table. The `locate` task can use an efficient binary search algorithm to place contributions in its vector block, giving

$$t_{loc} \propto \frac{(v^k \log(v/N))}{N} \text{ , and } t_{op} \propto \frac{v^{k+1}}{N^2}. \qquad (3.2.4)$$

This is our justification for putting any extra tasks on to the `locate` processor.

3.3 Multi-Shell Calculations.

3.3.1 Spurious States.

Up until now, the shell model code has mainly been used for performing calculations in one major shell (the *sd*-shell or the *p*-shell). The point of increasing the size of calculations that it is possible to do is to increase the number of shells which can be included. However, once elements of a second or third major shell are introduced, the shell model as it stands begins to produce spurious states.

In the shell model, we usually have a core, which gives us a single-particle potential. Obviously, this core is assumed to be fixed in space, but it is a property of the nuclear hamiltonian that it must be invariant with respect to Galilean transformations. As a result of breaking this symmetry of the Hamiltonian, some eigenvalues in the shell model calculation may have spurious components. These components are due to centre of mass motion.

This centre of mass motion is unphysical, so states which exhibit this motion are spurious, and must be removed from our results. How this is done follows from a more mathematical description of the problem, which will also elucidate why it only becomes a problem in large calculations.

Consider the single particle Hamiltonian for the Harmonic Oscillator potential:

$$\mathbf{H} = \frac{1}{2M}\sum_{i=1}^{A}\mathbf{p}^2(i) + \frac{1}{2}M\omega^2\sum_{i=1}^{A}\mathbf{r}^2(i) \tag{3.3.1}$$

$$\begin{aligned}\mathbf{H} &= \frac{1}{2M}\sum_{i=1}^{A}(\mathbf{p}(i) - \frac{\mathbf{P}}{A})^2 \\ &+ \frac{\mathbf{P}^2}{2AM} + \frac{1}{2}M\omega^2\sum_{i=1}^{A}(\mathbf{r}(i) - \frac{\mathbf{R}}{A})^2 \\ &+ \frac{1}{2}MA\omega^2\mathbf{R}^2\end{aligned} \tag{3.3.2}$$

where M and A indicate the mass of a nucleon and the number of nucleons in the nucleus, respectively. The vectors

$$\mathbf{R} = \frac{1}{A}\sum_{i=1}^{A}\mathbf{r}(i) \tag{3.3.3}$$

and

$$\mathbf{P} = \sum_{i=1}^{A}\mathbf{p}(i) \tag{3.3.4}$$

represent the centre of mass spatial co-ordinates, and momentum, respectively. The shell model hamiltonian is seen to consist of two parts, namely a Galilean invariant part, and a purely centre of mass part:

$$\mathbf{H} = \frac{\mathbf{P}^2}{2AM} + \frac{1}{2}M\omega^2 + \frac{1}{2}MA\omega^2\mathbf{R}^2 \tag{3.3.5}$$

An eigenfunction of the single-particle Hamiltonian can be expanded in terms of the (commuting) eigenfunctions of the two separate parts. It is only the Gallilean invariant part that we wish to retain. Now consider the number of oscillator quanta in such an eigenstate (of the single particle Hamiltonian).

$$\mathbf{H}\psi_N = E_N\psi_N = (N + \frac{3}{2}A)\hbar\omega\psi_N \tag{3.3.6}$$

Expanding this in terms of the eigenfunctions of the centre of mass and invariant Hamiltonians, we see that any combination is allowed of the form:

$$\psi_N = \phi_{N_{C.M.}}\psi_{N_{rel.}}, N_{C.M.} + N_{rel.} = N \tag{3.3.7}$$

One conclusion that can be drawn from this, is that if N is zero, then only one contribution is allowed. This is effectively the case when we only include one major shell above the core. Another consequence of this observation is, incidentally, that we do not need to include the kinetic energy operator in the Hamiltonian when we only have one major shell if we only want to know the *differences* in the energies of the eigenstates. In the shell model, the binding energy cannot usually be calculated (with realistic interactions) to any great degree of accuracy, so this is not a great loss.

However, as soon as we start to include more shells, we get the spurious states mentioned before (with non-zero $N_{C.M.}$). So, how do we get rid of these states? Well, each state that we want rid of has got a non-zero number of centre of mass oscillator quanta. We could simply add a term to our Hamiltonian whose value (when operating on such a state) is simply a multiple of its C.M.

quanta. An obvious candidate is the harmonic oscillator potential in C.M. co-ordinates, with some suitably large oscillator parameter. This places the spurious states high enough in energy that they do not appear in the generated spectra.

A second method is to attempt to identify all states with a high degree of spuriosity. I say degree, because all of the above algebra was done using a *harmonic oscillator* central potential. For the more realistic potentials used in actual calculations, we could expand in terms of oscillator functions and would find that usually states have some component which is spurious (in that sense). However, even in terms of the eigenfunctions of the true relative and C.M. hamiltonians, the states can be of mixed spuriosity, since the basis is likely to be a truncation of any full $N\hbar\omega$ space. Still, we can attempt to identify states as being *nearly* non-spurious, by applying appropriate operators. In the current code we use the centre of mass kinetic energy operator, and have had reasonable success in identifying states correctly.

3.3.2 Basis Generation

A second problem which occurs in multi-shell calculations is generation of the basis. In small calculations, this takes a minimal amount of time, but in multi-shell calculations, it could possibly take on the order of *hours* if steps are not taken to optimize how the states are formed. Additionally, we do not expect to be able to perform unrestricted calculations immediately; ad-hoc truncations are performed on the basis by restricting the number of particles which may exist in a particular j-shell, or the excitation from the 'pure' shell model position (particles fixed outside the valence major shell). This requires us to make the extra effort of calculating the number of particles in each j-shell, and major shell, for each state. A final consideration is that the constructed states *must* be in numerical order for the algorithms which search the basis

table to work.

The alternative methods that can be used to perform this task are:

- Forming tables of possible states for each j-shell,m value and number of particles considered. These are then combined to form the basis. This may at first sight seem wasteful; but, each table is re-used many times generating a single basis, and what is more, these tables can be stored in files and used for *every* ensuing calculation. Thus this task could be done by a separate program once only. A second program is required to combine the tables, and it is here that we run into problems; it is not particularly easy to generate states from these tables directly in numerical order.

- Proceed as in the first method, except that at the combination stage, we create many files containing partial lists which are ordered. These files are then merged (using any off-the-shelf algorithm [36]). This is really easy to do; the resulting method is, however, fairly slow. The number of tables that are used by these two methods can also be huge.

- Generate proton half-states and neutron half-states; work out the relevant quantum numbers for each and discard the unwanted states. This is essentially the algorithm used in the original program. This has two real problems with it; firstly, the numbers of half states can end up greater than the number of states in the biggest *sd*-shell calculation for problems of the size we propose to tackle; it takes very little advantage of the truncations we impose. Also, the method takes an inordinately long amount of time, since typically, when truncations are imposed, more than 99% of the states generated are found to be unwanted.

This problem is to an extent unresolved. We continue to use the third algorithm, but now it is done in a separate program, whose output is sent to

a file read by the main program. The basis generation stage is an obvious candidate for parallelization, since for each half-state (proton or neutron) the formation of complete states is independent.

This would also solve another problem, since it is found that the serial host machine (a Sun workstation), which handles access to the parallel computing surface, would not have enough memory to hold all the states generated. At the moment in the parallel code this problem is circumvented by generating the basis in blocks, and when each block is passed to a processor pair (as described in a later section) the memory used is freed up. To do this efficiently, though, requires that the program knows *in advance* how many states it is going to generate. This is simple if we do not truncate the basis in any way, but normally, there is no simple formula for this. The program currently does this for *sd*-shell calculations by generating the states without storing them once, purely to count them, and then generating them again to pass on. This obviously is not practical if the basis is large. With the newer method, we can measure the size of the basis efficiently simply by asking the operating system the size of the basis file. A final benefit is that calculations to compare model Hamiltonians do not require the basis to be recalculated.

3.3.3 Word Size.

A third consideration when creating multi-shell bases is that the number of orbits will probably exceed the number of bits in a computer word. When this work was begun, it was considered that 64 bits (and therefore orbits) would probably suffice, and it was believed that the transputer was capable of 64-bit calculations. As it turns out, the transputer can do 64-bit double precision floating point calculations, but that it cannot do any more than 32-bit integer arithmetic, which we require in the program for manipulating the bits of the state representations. As an example of where this might be required, if all of

the orbits in the usual ordering up to the $f\frac{7}{2}$ orbit are included, we require to be able to represent 56 orbits.

To overcome this problem, in the past a compressed representation has been suggested based on the properties of binomial coefficients [18]. This is actually of no use in the present circumstance, since to calculate the matrix elements we must unpack the states into both normal occupancy and the parity representations, both of which are 64-bit. The compressed representation would be useful, for instance, if the entire matrix were stored in sparse form instead of two-body form. Two other drawbacks of this representation are that it requires extra effort to unpack the representations, and that it still provides us with a limit of about 40 orbits compared with 32.

The solution to this problem — at least for less than 32 orbits for protons and neutrons separately — is to store the proton orbits and neutron orbits in separate words. This requires a radical rewrite of the central loop of the program; no longer are operators assumed to act on both halves of a state separately. In effect, the central loop is split into three parts; one where proton-only operators are considered, one where neutron only operators appear, and one which connects the two. The effects are more far-reaching than this; it is no longer possible to check the phase in the last loop by considering only two bits. If we order the bits with proton orbits highest then four bit values must be known: the bit where the proton operator acts, the lowest proton bit, the highest neutron bit, and the bit where the neutron operator acts.

This two-word representation becomes clumsy and slow, however, when it is used to search the basis table. A straight translation means that we must make two comparisons each time where we made only one before, but in at least the binary search of a vector block (see the section on the locate task later on) we can divide the search up into an initial search of proton half-states followed by a search of neutron half-states.

As a result of all of this, we manage to avoid most of the problems of using

a two-word representation at the expense of having increased the complexity and decreased the speed of the program. Incorporating the changes naively into the program made it *seven* times slower. A more careful rewrite, which involved reorganising the internal storage of the Hamiltonian, was only about half the speed of the single-word code; better, but still disappointing.

3.4 The Systems Used

Two parallel systems were used in the implementation of the code, both of which are based on the Inmos T-800 transputer.

The first system used was the MEIKO computing surface. At the start of the project this could be described as very much in a state of flux; the user logged into a special environment to run his programs on the transputer boards, using a mixture of occam and the programmers 'native' language in every application: the occam providing almost all the communications services. It was at this point that we considered the TINY routing program, since, even though it was unsupported, it would be easier to maintain and understand than a bulk of occam code. Programs were cross-compiled on a Sun Workstation before loading on to the transputer network, and programs had to be manually debugged. Separate script files had to be written to describe the mapping of the program on to the transputer network.

The MEIKO machine was at a later stage upgraded - so many people were now using the TINY system on MEIKO's machines that it became incorporated into the C libraries supported on the machine, with a few minor changes. Around this time, a new Sun motherboard was incorporated into the machine, providing a better integrated UNIX environment for running programs. However, there was still a need to write separate script files to describe how the network was to be constructed.

The PARIX machine was provided to the University of Glasgow's Elec-

tronic Engineering Department in the final year of the project. It, too began with a different operating system, Helios, but this was one which had significant advantages over MeikOS in how programs were run: the program could be written in such a way that the size of the network could be supplied as a runtime argument, essentially hiding the scaling of the program to different network sizes from the user, which is seen as a necessary property for the success of our code. The operating system was replaced half way through the year with PARIX, a small operating system kernel which resided on each transputer and provided both message routing and a dedicated error reporting facility, while making the Helios-style communications extensions more closely resemble those of Meiko C.

This operating system is where development ended up for a variety of reasons:

- PARIX was designed to be the O.S. of choice for T-9000 systems when they arrive; thus the program will be ready immediately when new architectures become available.

- Porting the working program from Meiko C to PARIX was made simple, by writing 'shell' functions in a program `meiko.c` which hid all machine specifics from the rest of the program. When moving to PARIX, this file was simply replaced by an equivalent file `parix.c` which performed the equivalent functions on the new machine.

- Availability of the machine: a University review has led to a decision to shut down the MEIKO facility from July 1994, while there is a possibility that Parsytec will allow us to run the ported program on a 1000 transputer machine in Holland or a 500 transputer machine in Greece.

3.5 The Code in Action

The code described in the previous sections was constructed first on the MEIKO computing surface. This did not turn out to be a simple development from the code of the previous thesis. It was found that the code of that thesis had been written in such a way as to intimately involve both the number and positions of all processes in the code that was to be compiled. Further to this, it was discovered that the program discarded parts of the last vector block generated.

The program was rewritten to make it independent of position and network size, thus removing the need to recompile the program before each run. Also, the mistake in processing vector blocks was corrected. The program had previously divided the basis into blocks of fixed size, several of which might be allocated to each processor. This was seen to be inefficient, since it required the user to know in advance the correct size of block for the basis he was going to use, and the program was simplified somewhat by making it allocate only one block to each processor. Finally, the program was run on varying numbers of transputers for various sizes of problem. The results are presented in raw form in table 3.1, and are manipulated to show the effect of the topology in diagram 3.2

It is seen from the graph of the results 3.2 that the speed of the program falls off markedly from what we would consider the ideal speedup. This particular set of results was obtained with the MEIKO's autoconnect facility, which connects processors together in a regular fashion. The underlying topology — essentially a random net — is hidden from the programmer, as the commands treat point-to-point and nearest neighbour communication identically. Using the model of the program developed in the previous section, we can hopefully determine the effect of the topology on the program.

The results are startlingly different from what we might expect. Instead of

No. of Transputers	Iteration time (seconds)
3	11119
5	5911
7	4199
9	3269
11	2710
13	2316
15	2042
17	1838
19	1648
21	1545
23	1404
25	1337
27	1282
29	1216
31	1177
33	1147

Table 3.1: ~24,000 state calculation

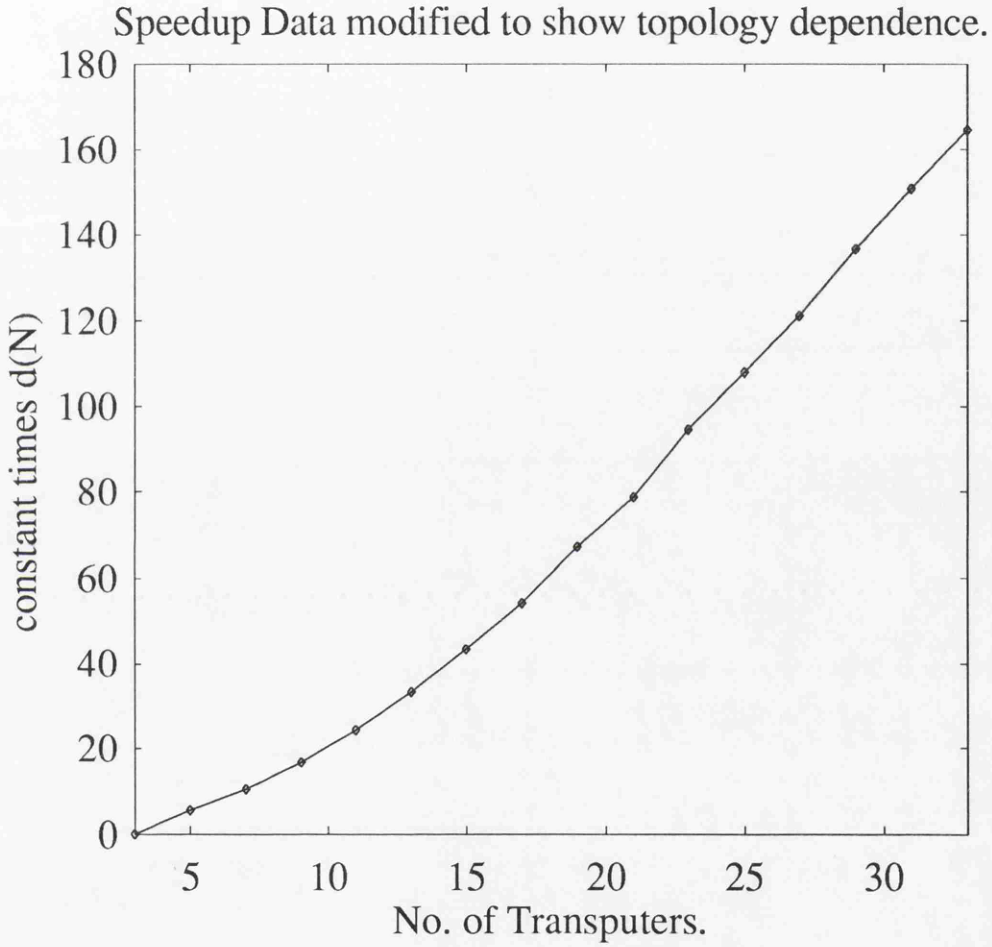


Figure 3.2: Iteration time vs. Number of Transputers

a logarithmic dependence on network size, the program instead scales slightly worse than linearly with the size of the network (note that a program which did no communication would appear on this graph as a line running nearly parallel to the x-axis at nearly zero). What has gone wrong?

A likely answer to what has happened here is that there are only 8 transputers per motherboard in the MEIKO computing surface. Once this number is exceeded, we should find that the time taken to send messages is greater, since we have to communicate between more than one board on the machine, a task which is performed by dedicated routing switches. It is highly probable that a given run of the program will be allocated processors that are split over several motherboards even if the number of transputers required is less than 8, since other programs may already be running on the machine.

The experiment above was repeated for other topologies, but the results were essentially the same. This is particularly disappointing given the effort that was taken to make the program topology independent! However, in the more recent versions of the languages available for transputers it is not possible to take advantage of nearest neighbour communications *mixed* with point-to-point communication without risk of deadlock, since the systems were not designed to perform both types of routing simultaneously.

A second possible explanation is that which is explicitly stated for the Parsytec system: in this machine, the transputers are *always* wired into a grid topology, the intention being that, since all communication looks the same, the user will not consider the underlying topology. This may seem shortsighted, given that the topology can affect the performance of a program quite dramatically, but PARIX is designed for T-9000 systems which have on-board facilities for the virtual topologies that we have in software (e.g. TINY). Since the main processor no longer does any communication routing, we have every right to expect (using the model of the last section) that the communication time will drop to virtually nil in many applications. It may seem odd to the

reader that so much effort has gone into understanding network topologies when in the end they are hardwired as grids, but it must be remembered that up until around the time of the start of this project it was common for a user to have to take the machine apart and physically wire in the topology required. Software routing has removed this extra complexity from the program, while, it seems, reducing the efficiency of the resulting networks.

3.5.1 Scott's algorithm

These results are not as convincingly good as we would like to achieve. Obviously, the communication time we have achieved is unreasonably large. The root cause of this is the non-locality we have introduced by dividing the data up along the vectors. Another method of parallelizing the Lanczos algorithm has been researched[37], wherein as well as dividing up the vector into blocks, the matrix itself is divided up. Each processor gets two vector blocks, and computes both the forward an reverse matrix contributions between these. In diagram 3.3 the contributions v'_{ij} are added:

$$v'_j = \sum_i v'_{ij} = \sum_i H_{ij} v_i \tag{3.5.1}$$

The v'_j are the blocks of the resulting vector, v', which will then be orthogonalised for use as the next Lanczos vector. This addition and orthogonalisation is done at the end of each iteration, on the processors which handle the blocks along the diagonal (in an implementation which takes no account for the symmetry of the matrix). Thus, all processors act as `operate` tasks, with the diagonal processors acting as `locate` tasks between iterations.

Obviously, this means that we no longer need to communicate to processors distant to each other in the network. In fact, the only communication that is done is passing round copies of the vector at the start of each iteration, and passing back the new vector for reorthogonalization at the end. With this

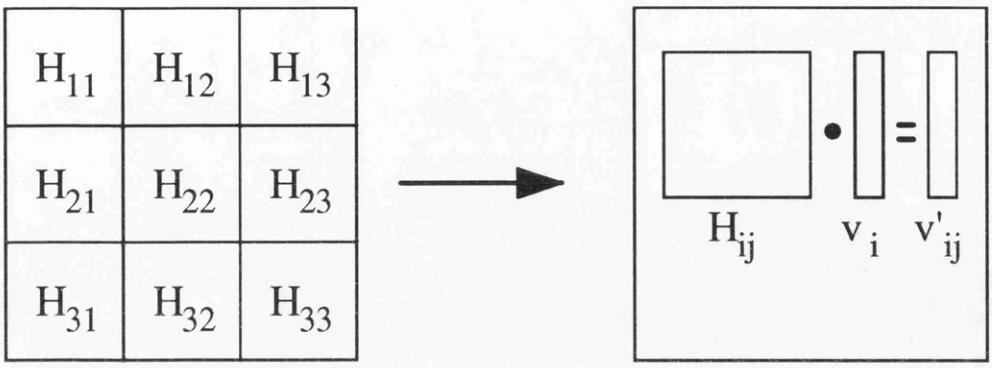


Figure 3.3: Division of Labour in Scott's Algorithm

in mind, the performance of this code should not degrade appreciably with network size at all.

However, there is a new problem, in that instead of needing $2N$ processors to divide a vector up N times, we now need $\frac{1}{2}N(N-1)$ processors. This means for instance, that 1000 processors with this algorithm leave the same amount of space for vector blocks as only about 90 with the code with the previous algorithm. However, it will run more than ten times as fast on the problem. This is a serious blow to our plans for doing a 20 million state calculation - remember, this required memory worth around 400 processors to do the job. Two million or so states looks much more realistic if we use this algorithm, and this with great difficulty.

It should be emphasised that in an implementation of the Glasgow code using Scott's algorithm, we do not actually divide up the matrix, since it is never stored explicitly, but instead, only use some of the generated matrix elements. This becomes more and more inefficient as we divide the matrix up further, so the running time does degrade slightly as the problem is scaled up. This can be got around by looping over both the block of states associated with the v_i and that assosciated with the v_j, and generating the pairs to create

and destroy by looking at the overlap of these two states. This is somewhat slower in the usual implementation of the program, but may be useful when applying Scott's method. In a chapter 5, I discuss a method of reorganizing the matrix which would also alleviate this problem.

Chapter 4

Applications.

The illustration
is nothing to you without the application.
You lack half wit. You crush all the particles down
into close conformity, and then walk back and
forth on them.

Marianne Moore, 'To A Steamroller'.

4.1 Rotational Bands.

Work that it was proposed to do with Dr. Ian Wright of Manchester University will be described. It was aimed at determining where proposed rotational bands should terminate, by using a much larger model space than has previously been possible. As well as this, the effects of inclusion of different parts of the model space were to be investigated, as were the accuracy of several model interactions[9, 38, 8].

The first problem that is encountered in any large-basis shell model calculation is that of choosing an appropriate Hamiltonian. This will be discussed first.

4.1.1 The Nuclear Hamiltonian.

As previously mentioned, the conventional approach to the shell model is to use a Hamiltonian where some of the matrix elements are fitted to low-lying states of well known nuclei [9]. This approach runs into immediate problems in larger calculations. Specifically, we no longer have enough states whose character is well known enough to do a good fitting. Indeed, the character of states which span several major shells, and are thus most affected by cross shell matrix elements are precisely those we wish to investigate, i.e. those for which the interaction is least known.

Worse than this, the calculations that would be required to fit the interaction become next to impossible. In the m-scheme shell model, we cannot use the truncations that allowed Wildenthal to fit interactions in the sd and pf shells. The calculations we end up doing, to perform the fit, are those that were deemed so difficult that the present program became necessary to attack them.

Consequently, other means of fixing the Hamiltonian must be sought. There are several approaches that have been tried with varying degrees of success when this problem has occurred in the past.

- Combining several fitted interactions. For instance, the Preedom – Wildenthal sd interaction and the Millner – Kurath $0p$ shell interaction, in combination with some cross-shell interaction, could be used for the mass region near the closure of the $0p$ shell.

  There are some obvious objections to this. Firstly, how we pick our cross shell interaction is crucial, since it is, in essence, its action that we wish to investigate. However, there is no obvious candidate for this interaction. Secondly, the fitting of these interactions takes into account in some way the fact that shells were missing from the model space in the fit; but we restore some of the most important of these shells in our calculation.

We'd like our calculations to be free of this kind of double-accounting.

- Using a schematic interaction. The main candidate here, is the modified surface delta interaction (MSDI). It is really too simple to model any effects that we may be interested in realistically, but this very simplicity lends it the advantage that its results are easily interpretable. There is a problem with this interaction [39] when used over three major shells: it has far too large a matrix element between major shells of the same parity. This would necessitate the insertion of another 'fudge factor'. However, since this is just one number we have to vary, the interpretation of results should still be fairly simple.

- Using a Hamiltonian derived from a potential. The Kuo-Brown matrix elements[40] are readily available, and would be our first choice in this category. Their use may seem a little odd, since many better nuclear interactions have been devised since this was introduced, but since the calculations in the *sd* shell used the Kuo matrix elements, with only the least well determined being fitted, they are probably the best way to obtain direct comparison with earlier results.

- Using a Hamiltonian derived directly from experiment, i.e. without an intervening potential model. There is (as far as I know) only one attempt at doing this; the Sussex [8] matrix elements are supposed to be a highly model-independent set of matrix elements derived from the phase shifts in nuclear scattering experiments. Unfortunately, their solution is just one of a large class of phase-shift equivalent interactions, being only one of the simplest of these. The most recent of these sets of matrix elements is also relatively old in terms of the experimental data it was derived from. However, this model independent approach is very attractive, and perhaps it is time to use the latest phase-shift data,[41] which is now

much more complete, and revive this body of work.

For reasons of simplicity, and the ability to quickly compare results, we choose to use the MSDI and Kuo-Brown sets of matrix elements, and to eventually move to using the Sussex set.

4.1.2 Band Terminations.

The particular property that we wish to investigate is the termination of rotational bands in light *sd*-shell nuclei. It is well known that some *sd* shell nuclei display rotational bands, which are easily explained in the context of a collective model where the nucleus is deformed. States in the same band have similar structure - classically we would say that the nucleus is shaped identically in each state in the band and so has a definite moment of inertia. Different bands are labeled by the angular momentum of the lowest state of the band (the 'bandhead'), since in the classical model we are rotating the rotating nucleus additionally to reach higher spins, so the lowest spin a nucleus can have in a band is this value, K

A simple collective model of the nucleus would naively expect the bands so formed - series of states with angular momenta that differ by two - not to terminate at all, but to have more and more levels as the nucleus spins faster and faster. However, in practice, the energy levels of a band terminate at relatively low angular momenta. This would suggest that a more detailed model of the nucleus is required.

Microscopically, the phenomenon of band terminations has been quite successfully described in the *sd*-shell by looking at theoretical calculations of the occupancy of the highest-spin orbits. With n particles, obviously the highest spin that can be achieved by placing these particles in the orbits of a shell of angular momentum j is $(2j+1)n$.

This is a very simplistic model though, and it could prove worthwhile to

check to see if there are any states with more complicated structure in the band. This is looked at in the next two subsections.

4.1.3 ^{19}Fluorine.

In ^{19}F, the states of interest are two $\frac{11^-}{2}$ states which may or may not be attached to bands in this nucleus.[42] There is some as yet not fully explained data on the α- transition rates of some of these states. (from ^{19}F to ^{22}Na). Note that strong transition rates indicate likely common structure, it was hoped that we could look at the structure of the states in the shell model, or better still, do the calculation for ^{23}Na as well and get the α decay rates directly for easier comparison to experiment.

Our Hamiltonian has been chosen, it remains to choose a model space that will cover any interesting structure, while producing a basis that is still small enough to handle. Ideally, for this particular calculation, we'd like to include the p-shell below the sd-shell, as well as the $f\frac{7}{2}$ shell above. It may seem that the $f\frac{7}{2}$ level is unnecessary, but it is known that the sd-shell lies too close to the $f\frac{7}{2}$ in very light sd-shell nuclei for the sd-shell to be considered closed at the top. However, this would give a basis size of at least a few million states. A better idea is to try the calculation at different levels of accuracy, only increasing the number of orbits considered if the previous ones added made a difference.

4.1.4 ^{24}Magnesium.

In ^{24}Mg there is a very similar problem, in that there are several states which appear to belong to rotational bands but have as yet not been positively identified as such[43]. These states are denoted with question marks in figure 4.1. This calculation is of especial interest to us as it is in the middle of a shell: a full $1\hbar\omega$ calculation is almost certainly required in this problem and that will

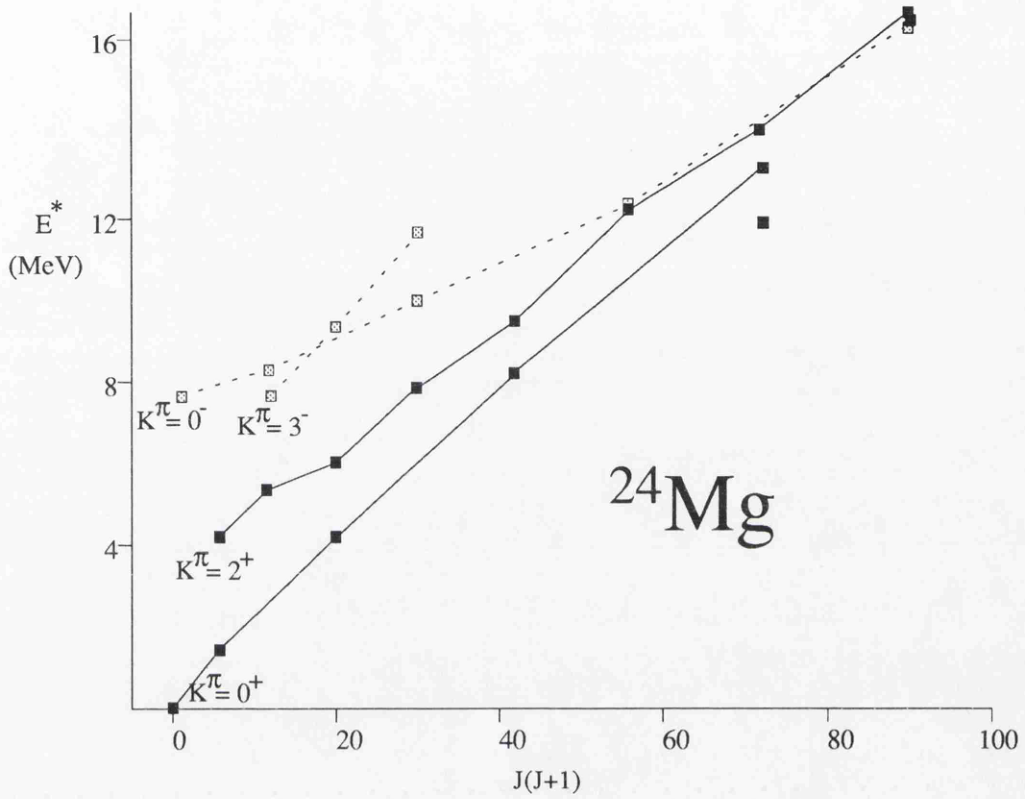


Figure 4.1: Proposed Rotational Bands in ^{24}Mg.

have a basis of tens of millions of states. This is obviously well beyond the capabilities of any program with the exception of the parallel Glasgow code, involving as it does excited states which we want to find the symmetries of, and so cannot use either Monte-Carlo or collective methods.

4.2 The Cranked Shell Model.

While the shell model as it stands picks up all of the states in a given energy region, it is accepted that the model space quickly becomes too big for this to be practical. The usual approach in the model is then to truncate the

calculation by restricting the occupancy of various orbits. Another approach, used in collective models, is to assume that the nucleus is deformed into a prolate or oblate shape from the start, and to look for a stable shape for the nucleus.

The idea in using a deformed basis is that the basis states 'effectively' include components of shells not in the calculation if we use the same 'shells' as used in the spherical basis. There is obviously a (possibly infinite) expansion of non-spherical oscillator states in terms of spherical oscillator states, with the leading term being a component of the spherical state (at small deformations). By including the terms after this, we obtain a leg-up into higher energy regions than those covered by the shell model. Our justification for doing this is that states with a large overlap with the deformed states so formed do exist, forming rotational bands in the spectra of many nuclei. Obtaining more than the lowest few states in these rotational bands in a spherical model becomes extremely difficult.

This approach generally uses variations on the *single-particle* shell model, where the particles move in an assumed potential. Two-body interactions only affect the results once the energy minima for the ground state have already been found. As mentioned in chapter one of this thesis, this is considered unsatisfactory; can we justify the single-particle potential, and what effect does the two body interaction have on the position of the ground state energy minimum? At this point, the approach of the collective model practitioner is to apply perturbation theory.

I have collaborated with Dr. Neil Rowley of Daresbury and Dr. Stefan Frauendorf of the NBI to investigate an alternative to this approach, in which we set up a shell model calculation in a *deformed* basis. Here we still assume some one-body potential, but within it, we allow full two-body interactions. The choice of interaction used here is unlikely to be one used for complete spherical shell model interactions. To begin with, we would like to compare

the results of these calculations to those with the single-particle type calculations. Secondly, the two-body interaction must take account of the subtraction of the one-body interaction from the usual (spherical model) interaction. The way to do this is not at all obvious without actually repeating the calculation in both models, missing the point of the exercise. As a consequence, schematic interactions are used. A second reason for using simple schematic interactions is that the properties of the spectra can be related directly to the (few) parameters of the potentials

4.2.1 The Choice of Calculation.

In the current study, we have chosen to look at the $g\frac{9}{2}$ j-shell, and also the $h\frac{11}{2}$ shell. The reason for doing so lies in the Nilsson level scheme for zero deformation.

In this diagram, it is seen that the $g\frac{9}{2}$ shell lies in a shell with no other levels of the same parity. Thus we expect that most of the states of this (positive) parity in the mass region where we might expect the orbit to be half-filled should in fact consist entirely of a $g\frac{9}{2}$ component, up to the threshold for two-particle excitation. A similar story can be told for the $h\frac{11}{2}$ shell.

Since these shells appear to fairly independent of interactions with other shells, they are ideal shell model candidates, and the model spaces involved are almost trivial. We do not, however, see these calculations as the limit of using the shell model in deformed calculations, the driving force behind all of this is the new parallel code which can handle extremely large calculations. This should allow us to do similar calculations which allow several shells to mix, and also, as we shall see, to perform cranked calculations.

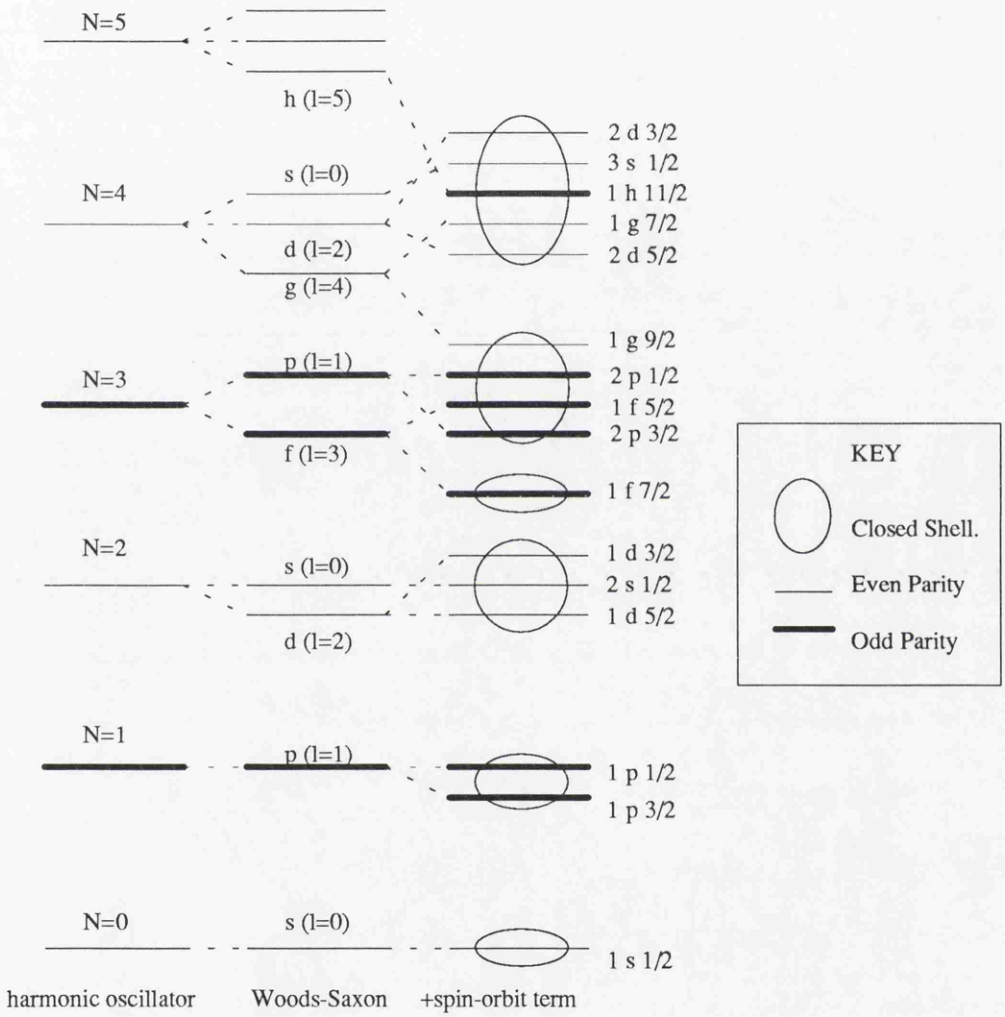


Figure 4.2: Level Scheme in Single-Particle Model

4.2.2 The Hamiltonian.

As mentioned above, the Hamiltonian used is to be schematic. Firstly, there is a quadrupole deformation term:

$$\mathbf{H} = \mathbf{H}_{def} + \mathbf{H}_{2body} \tag{4.2.1}$$

where:

$$\langle \phi_{j,m} \mid \mathbf{H}_{def} \mid \phi j\prime, m\prime \rangle = \delta_{(j,m),(j\prime,m\prime)} \kappa \frac{3m^2 - j(j+1)}{j(j+1)} + \mathbf{H}_{core} \tag{4.2.2}$$

and the deformation parameter κ is allowed to vary. A value of 3 for κ corresponds approximately to a 2:1 deformation. The 'true' liquid drop deformation into an ellipsoid is not, in fact, quadrupole, but for small deformation, this is a very good approximation.

The two body term is chosen to be the Surface Delta Interaction, since it consists of Clebsch-Gordan Coefficients which can already be generated within the program, and it has a simple enough form for the results to be more readily understood. In particular, it has very few parameters, the form that we use being [44]:

$$\begin{aligned}
&\langle j_a j_b \mid V^{SDI}(1,2) \mid j_c j_d \rangle_{JT} = \\
&\quad (-1)^{n_a+n_b+n_c+n_d} \frac{A_T}{2(2J+1)} \sqrt{\frac{(2j_a+1)(2j_b+1)(2j_c+1)(2j_d+1)}{(1+\delta_{ab})(1+\delta_{cd})}} \\
&\quad \times \{(-1)^{j_b+j_d+l_b+l_d} \langle j_b \frac{-1}{2}; j_a \frac{1}{2} \mid J0 \rangle \langle j_d \frac{-1}{2}; j_c \frac{1}{2} \mid J0 \rangle [1 - (-1)^{l_a+l_b+J+T}] \\
&\quad - \langle j_b \frac{1}{2}; j_a \frac{1}{2} \mid J1 \rangle \langle j_d \frac{1}{2}; j_c \frac{1}{2} \mid J1 \rangle [1 + (-1)^T] \}
\end{aligned} \tag{4.2.3}$$

The two parameters, A_0 (the coefficient of the isoscalar term) and A_1 (the coefficient of the isovector term) are allowed to be different.

4.2.3 Zero Deformation.

In this case, we just have a normal (spherical) shell model calculation with the Hamiltonian described in the previous section. The first calculation we attempted of this type, using the $g\frac{9}{2}$ shell, successfully reproduced the results of a previously published paper of my collaborators. [45]. This was done purely to check that the adapted code was working properly, before moving on to a larger calculation.

The zero-deformation calculation was then carried out for the $h\frac{11}{2}$ case, since my collaborators had already performed this calculation, but had been unable, using their method, to determine the isospin of the states which they were interested in. Two calculations were in fact carried out; one for $h\frac{11}{2}$ with two protons and two neutrons, and one with two protons and ten neutrons. The answers are expected to be similar, since in the second calculation we are really using two neutron 'holes'. There is no particular reason for choosing neutron holes; protons and neutrons are equivalent in this calculation. The results of the calculation are in figure 4.3.

The states of interest in this calculation are the two lowest 10^+ states. It was found, using the method employed by Rowley and Frauendorf, that when deformation increased, there was a lot of mixing between these two states in the n=2 case, whereas there was very little mixing between these states in the n=10 case. There is also a marked difference in the energy gaps between these two pairs of states. The proposed explanation for this was that the isospins of the two 10^+ states differed in the n=2 case, but were identical in the n=10 case. This was, indeed found to be correct.

A second calculation, which also could not be done previously, was completed. In this calculation, the isoscalar and isovector parameters used were different; the rationale for this was that having $A_1 \simeq 1.3A_0$ would be more realistic in the mass ranges where this particular shell would be the valence

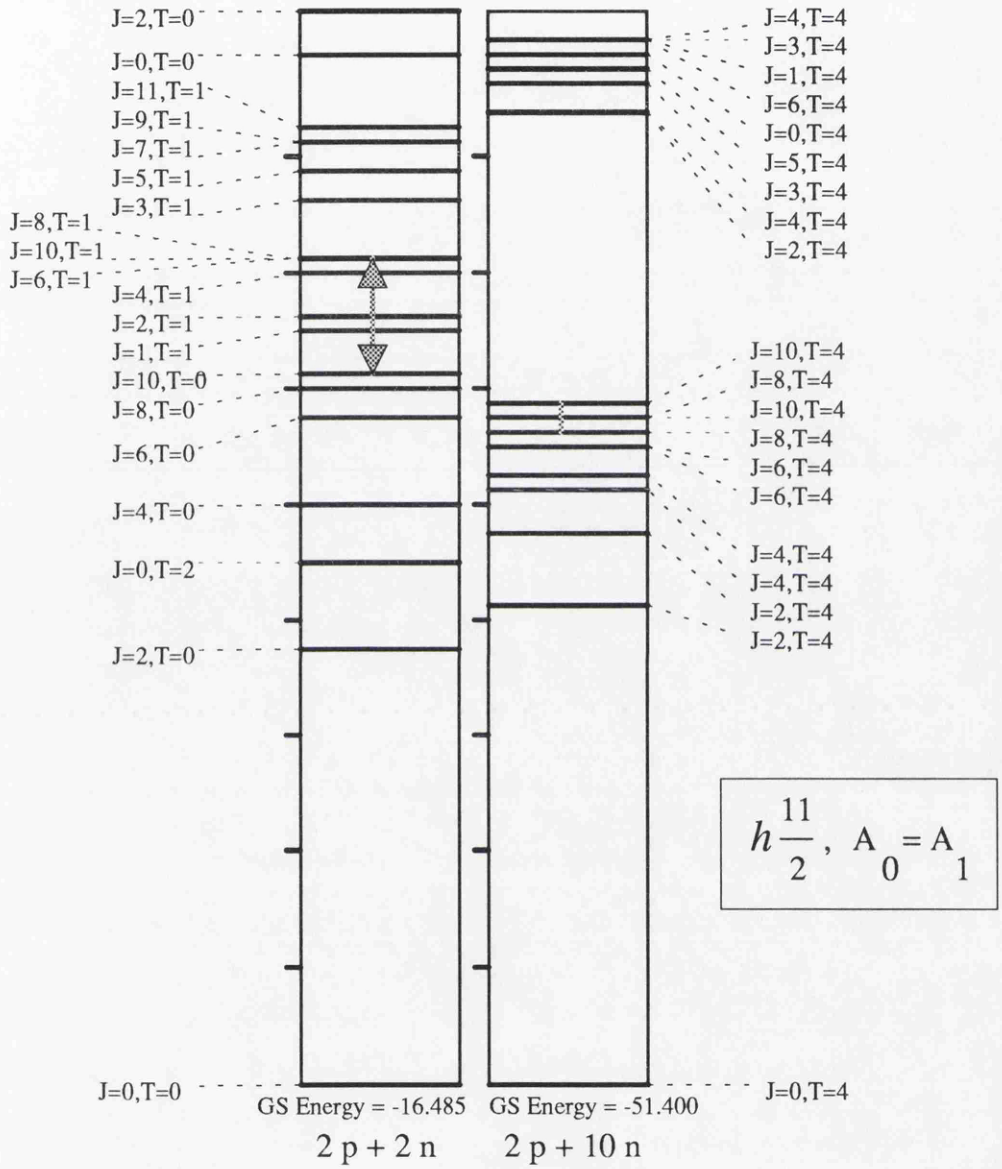


Figure 4.3: Identical Isovector and Isoscalar Terms. Units of A_0

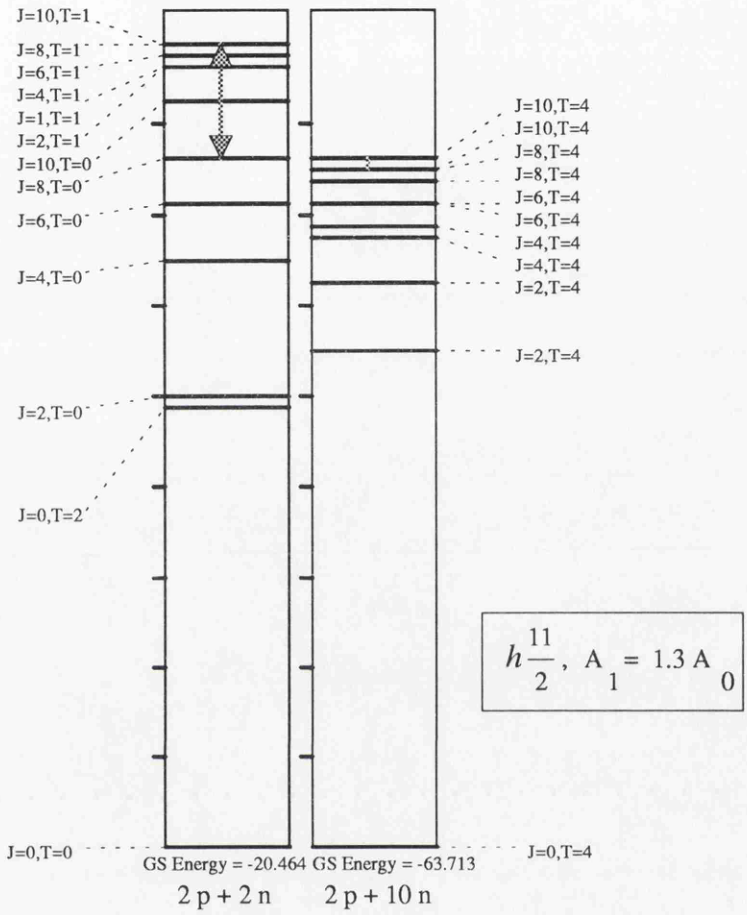


Figure 4.4: Different Isovector and Isoscalar Terms. Units of A_0

shell. The results of this calculation, wherein the effect that the lowest 10^+ states have different isospin persists, is shown in figure 4.4

4.2.4 Non-Zero κ.

The same calculation was done for non-zero $\mid \kappa \mid$. In the deformed basis, we are attempting to look for minima in the ground state energy. These are the points where the nucleus is stable when 'deformed'. We would expect this to occur in nuclei at some value of κ less than about 3. As previously mentioned, this is a 2:1 deformation, and states dicovered physically with this deformation or greater are fairly rare. Also, it is noted that the quadrupole deformation term is only strictly valid for *small* κ.

The calculation was therefore performed for both the 2-neutron and the 10-neutron cases, with kappa varying from -3 to 3, in steps of 0.1 . The core term was fixed to be zero. The ground state energy of each of these problems are plotted in graphs 4.5 and 4.6.

It is evident from the graphs that the only energy extremum occurs for zero axial deformation, and that it is a maxmimum. It should be obvious to the reader that something unphysical is going on here, since the slightest perturbation from the spherical state of the nucleus would lead to a rapidly increasing deformation of the nucleus with it ending up as an infinitely long filament (as this model cannot reproduce fission). This is more than a little surprising, but an explanation of what may be happening is given in section 4.2.7.

4.2.5 Identifying States.

Something that we wanted to do in this calculation was to look at the two 10^+ states previously investigated, at the stable deformations. Although such deformations were not found, it was attempted to follow the behaviour of

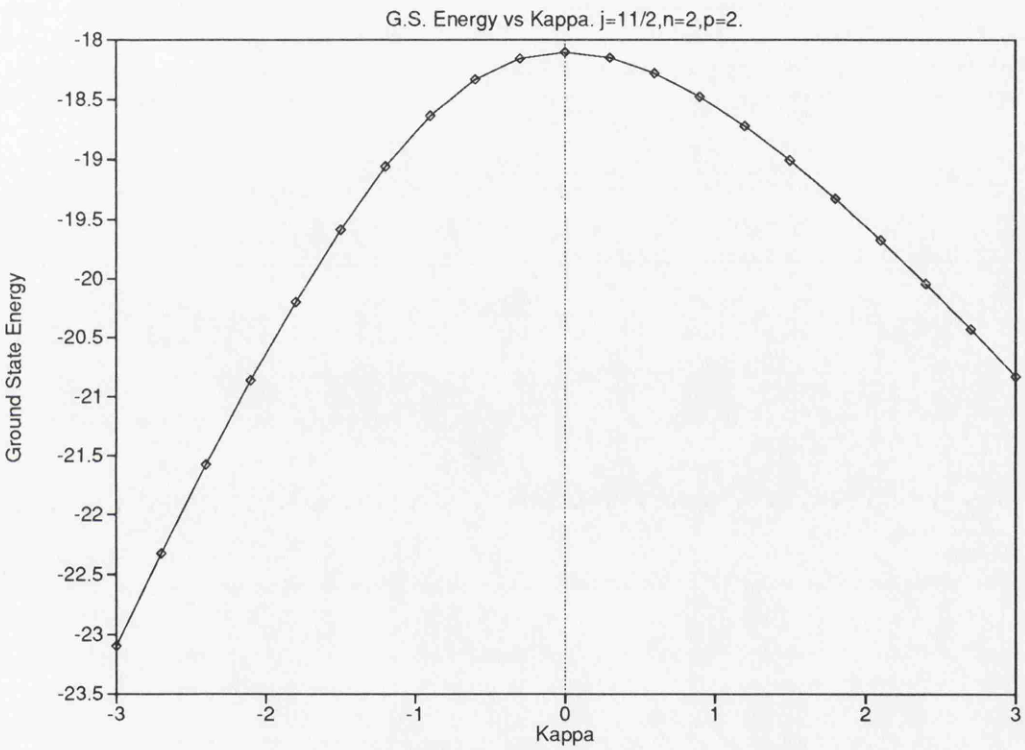


Figure 4.5: Ground State Energy, 2 Neutrons.

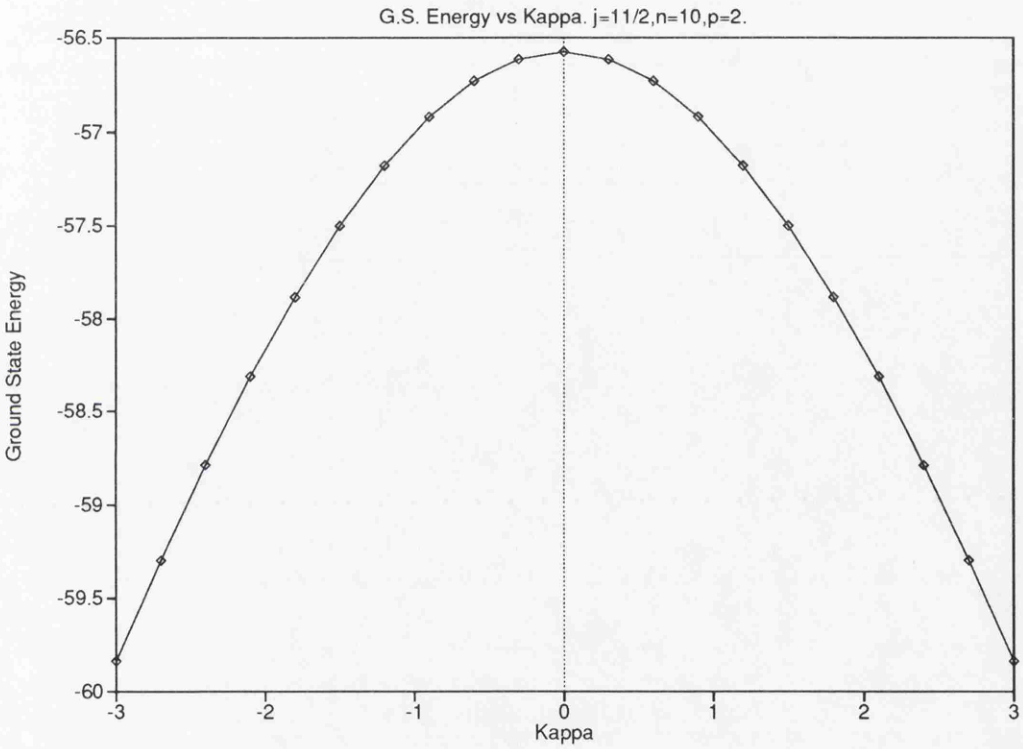


Figure 4.6: Ground State Energy, 2 Neutron Holes.

these states anyway. This is not trivial; the states split into 20 as soon as the deforming potential is applied, since the J^2 operator no longer commutes with the Hamiltonian, different states in the calculation very quickly interfere with and cross each other, as their energies change, and they become difficult to identify.

It was at first suggested that I use an angular momentum projection method to identify the states, the idea being that the J=10 states would have a J=10 *projection* using some method even though they no longer have good J when the usual operator is applied. The actual method used was to diagonalise the J^2 operator with an eigenvector of H being used as the starting vector. This was intended to find the vector with largest overlap with the eigenvector being used which also had good J^2. The trouble with this, is that the vectors obtained are not eigenvalues of the original matrix. What is worse, as the deformation used became larger the best overlap with a vector with well defined angular momentum fell markedly. I came to the conclusion that the results being obtained with this projection method were meaningless.

A second method that was attempted was to follow the vectors desired by taking dot products with a vector from a calculation with smaller deformation. This allows us to rank states, saying how much like the original state they were. The deformation was increased from $\kappa = 0$ to $\kappa = 3$ in steps of 0.1 . It was found that as the deformation increased to about 1, where levels are crossing, there can be several candidates for the vector being followed, with sometimes all of the candidates having a relatively low overlap with the previous vector (low was considered to be less than 0.9)

The essence of what we are trying to do here is to project out the 'intrinsic' states of the nucleus. The deformation does not really exist, it is just a tool to create the states of interest within a smaller basis. It is somewhat disappointing that we were unable to do this, as it is theoretically possible. This is certainly something that could be pushed further in the future.

4.2.6 Cranking.

Since we found no evidence of a ground energy minimum for non-zero axial deformation, we now look at triaxial deformation. The conventional way to do this is to add a spurious term giving the nucleus a specified rotational energy about some axis. Since this implies, firstly, that the nucleus can be rotated about that axis, it must be deformed. (Quantum mechanically, if an object has rotational symmetry about some axis it has zero moment of inertia about that axis.) Secondly, we are essentially specifying the moment of inertia of the nucleus about this axis. Thus, we create (effectively) a second deformation at right angles to our quadrupole deformation. Once again, it must be emphasised that the rotation is not 'real', the energy due to it should be subtracted from the Hamiltonian before we can measure the true energy of states generated in this way.

The calculation was done by applying the Hamiltonian:

$$\mathbf{H} = \mathbf{H}_1 + \mathbf{H}_2 - \omega J_x \tag{4.2.4}$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are the same as before. The additional, cranking, term is easy to evaluate in the shell model code, since we have the basic result:

$$J_x = \frac{1}{2}(J_+ + \mathrm{i}J_-) \tag{4.2.5}$$

And the operators J_+ and J_- are already implemented in the code for the evaluation of J^2.

It should be noted here that the cranking operator mixes states of different J_z, meaning that our basis must now include states of *all* J_z, substantially increasing the size of the basis. This is an interesting point because, if we included several shells in our calculation, and then used a cranked Hamiltonian, we would have in all probability more states than the current serial codes could

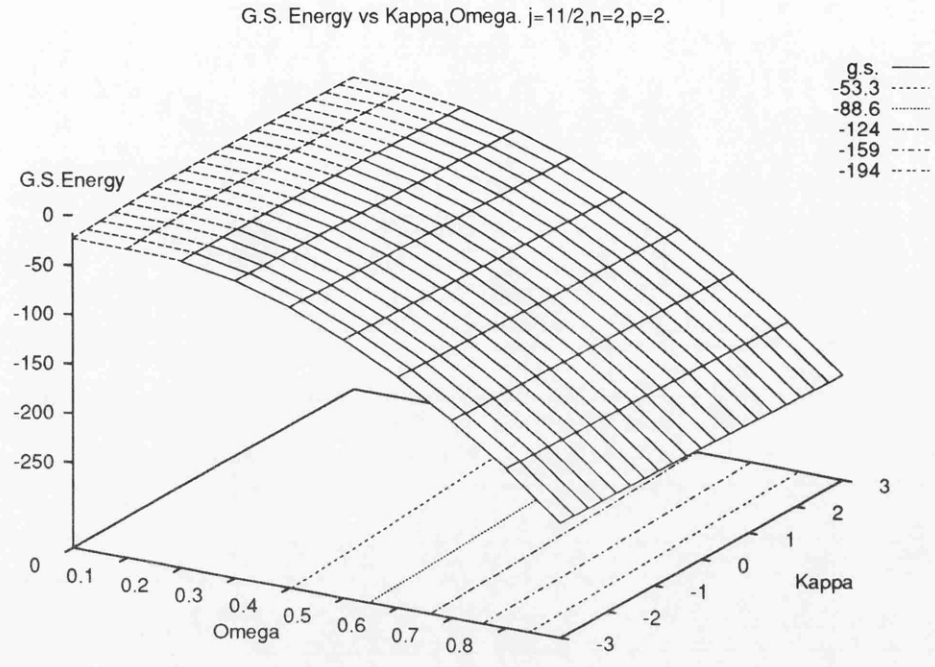


Figure 4.7: Ground State Energy, 2 Neutrons.(Cranked)

handle. Thus, cranking calculations such as this are possible applications of the parallel code.

The calculation was performed again for both cases; the results are presented in graphs 4.7 and 4.8.

4.2.7 Explanation of Results.

Once again, we have found no energy minimum, except that for the undeformed state. At this point we were perplexed, since the literature is littered with calculations using different methods which have several. Our model should act very like a particle-rotor model, where the rotational bands generated have a corresponding deformed ground state. What has happened here?

The first explanation is that we have got the correct result, that there are no deformed states in this nucleus. We can tighten this and say there are no

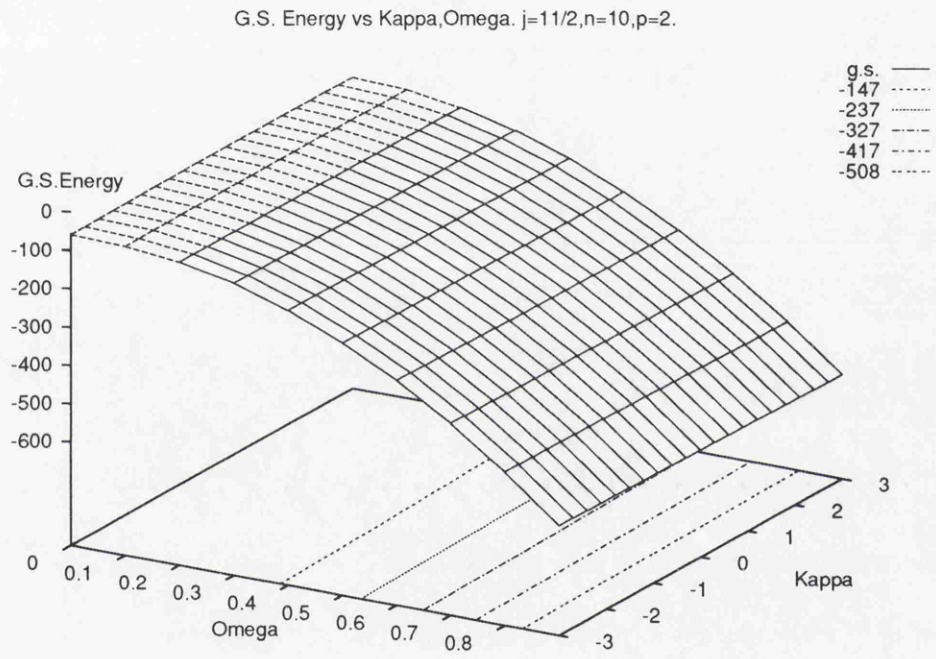


Figure 4.8: Ground State Energy, 2 Neutron Holes.(Cranked)

deformed states *in this restriction of the model space.* This does seem to be the answer. We initially checked that the energies of the states in the axially deformed cases had the correct asymptotic values, i.e., once the quadrupole deformation dominates the Hamiltonian. With no core, these shoot off to $\pm\infty$ in the correct way. Of course, the core term prevents this unphysical result. However, it is noticed that all states do so monotonically, indicating that the kind of turnaround that would be required to create some maximum or minimum just does not happen. The core states do not mix with those in the active shell, so this effect is not changed by the addition of the core term. This would indicate that we need at least one more shell in the calculation to produce any effects due to deformation in this model.

4.3 Conclusions

The results presented here are too far from being complete for anything particularly interesting to be said about the physics in either of these problems. However, it is clear that the problems with the band termination calculations are purely to do with the *sizes* of the problems that must be handled, as the application is just a typical shell model calculation. We have seen in the previous chapter that the code does not appear to be capable of the larger of the two problems (^{24}Mg) in the near future, but the fluorine calculation may be possible with some optimization of the code, such as to save memory by using a different reorthogonalization scheme.

The cranking calculation is a different beast, however. It does seem that this novel application of the shell model code could be pushed somewhat further without stretching the capabilities of the current code. The calculation needs to be expanded to include not just a core term but also interactions between neighbouring shells, which ought to produce results which contain more interesting structure.

Chapter 5

Alternative Methods.

> With my two algorithms, one can solve all problems – without error, if God will!
>
> *Al-Khorezmi*

The parallelized Lanczos algorithm has not been an overwhelming success. There are some piecemeal improvements to the code that can still be made, or wholesale replacement of the usual Lanczos method by something which may be faster can be contemplated. A miscellany of such schemes are discussed in this chapter, evaluating the pros and cons in each case.

5.1 Increasing 'Band-Diagonalness'

There is a possible change that could be made to the *data* in the problem instead of to the algorithm. In both the methods discussed in chapter 3, we see that the very off-diagonal matrix elements cause a large fraction of the work that we do: in the usual parallel scheme, they lead to increased communications costs, and in Scott's method, many more processors must be used. In fact, a further problem arises in Scott's method which decreases its efficiency from the ideal described above. Since we do not store the matrix,

but only the two-body form, we will generate many states which are outside the blocks stored by the off-diagonal processors. Thus, as we divide up further, we 'hit' states less and less and eventually most of the time that should be spent doing useful processing is spent rejecting states.

Both of these problems could be solved if some rearrangement of the states produced a matrix with a lower bandwidth. There appears to be a considerable scope for this: figure 5.1 shows a typical distribution of matrix elements in our Hamiltonian. We have investigated this possibility as it could produce large gains in speed for a relatively small one-off cost of re-ordering the matrix. It should be noted that it is necessary that the states be in numerical order, as at present, for the binary search in the innermost loop of the matrix multiplication to be fast. This need not be changed. All that is necessary is that before the matrix is divided into blocks, the states are in a 'dense' order: once divided up, we can re-order each basis block numerically to speed up the computation.

To start with, we need some definitions. Two states are said to be *connected* if there is a non-zero matrix element between them. There is a *path* between two states a, b if there exists a sequence of connected states starting with a and ending with b. The length of a path is one less than the number of states in that path, including the starting and ending state. Then the diameter D of a matrix is defined as:

$$D = \max_{\forall \text{ states } a,b} \left(\min_{\forall \text{ paths } p_{a,b}} \text{length}(p_{a,b}) \right) \tag{5.1.1}$$

The *valence*, v_i of a state in a matrix M is defined to be:

$$v_i = \sum_j \begin{cases} 0 & \text{if } M_{ij} = 0 \\ 1 & \text{otherwise.} \end{cases} \tag{5.1.2}$$

We will deal in this section only with matrices where v_i is the same for all i. This essentially means we are ignoring angular momentum, a point which I will mention again towards the end. The *width*, w of an ordering of states is

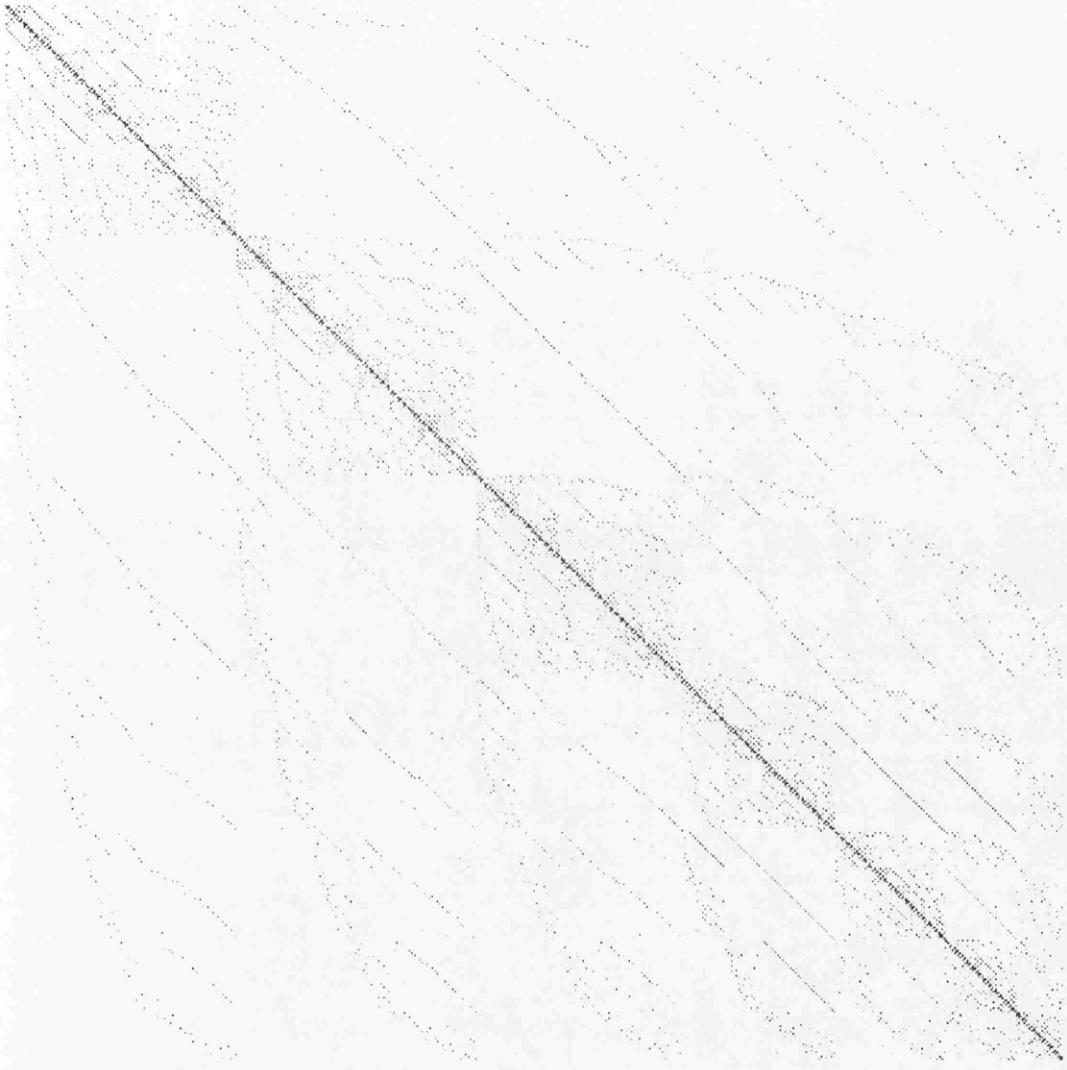


Figure 5.1: Distribution of Matrix Elements in a Typical Hamiltonian.

defined as:

$$w = \max_{i,j\,:\,M_{ij}\neq 0} |i-j| \tag{5.1.3}$$

We are now ready to define the *band density* of a matrix, ρ, which is essentially the density of a matrix ignoring the zeroes outside of the band.

$$\rho = \frac{v}{w} \tag{5.1.4}$$

Note that, from the definition, band-diagonal matrices have a band density of 1. If there are N states in our basis, the matrix is homogeneous, and a path exists between every state, then we have:

$$w \geq \frac{N}{D} \tag{5.1.5}$$

hence:

$$\rho \leq \frac{v \times D}{N} \tag{5.1.6}$$

Of more interest to us will be the lower bound on ρ, which will tell us what we can gain by using the best possible ordering. To obtain this, we will require some more problem-specific information.

We consider states formed by placing p particles in n orbits, with no other restrictions on the positions of the particles. There are $\binom{n}{p}$ such states. We can get the width of the usual numerical ordering quite simply, by noticing that the states:

0 0 0 ... 0 1 ... 1 1 1

and

1 1 0 ... 0 1 ... 1 0 0

are connected by a two-body matrix element, and, moreover, that they are the pair of states furthest apart in the basis table that are so connected. A little thought allows us to observe that here:

$$w = \binom{n}{p} - \binom{n-2}{p} \tag{5.1.7}$$

We also have, using the usual two-body matrix elements, that the valence is given by:

$$v = \binom{p}{1}\binom{n-p}{1} + \binom{p}{2}\binom{n-p}{2} \tag{5.1.8}$$

and that the diameter is, quite simply:

$$D = \begin{cases} \frac{p}{2} & \text{for } p \text{ even,} \\ \frac{p+1}{2} & \text{for } p \text{ odd.} \end{cases} \tag{5.1.9}$$

Next we consider what the optimum band density may be. Consider the first state in our list, say, s_1. We will be able to work out the optimum band density by looking at N_r defined by:

$$N_r = \sum_j \begin{cases} 1 & r = \min_{\forall \text{ paths } p_{s_1,s_j}} \text{length}(p_{s_1,s_j}) \\ 0 & \text{otherwise.} \end{cases} \tag{5.1.10}$$

This is simply the number of new states generated by making all paths looked at so far one longer. For the states described above, we have:

$$N_0 = 1 \tag{5.1.11}$$

$$N_r = \binom{p}{2r-1}\binom{n-p}{2r-1} + \binom{p}{2r}\binom{n-p}{2r} \tag{5.1.12}$$

The reason for looking at this statistic is this:

$$\forall k, 1 \leq k \leq \frac{p}{2}\ ,\ w_{min} \geq R_k = \frac{1}{k}\sum_{r=0}^{r=k} N_r \tag{5.1.13}$$

where w_{min} is the smallest possible width, achieved by some optimal ordering. In fact:

$$w_{min} = \max_k R_k \tag{5.1.14}$$

Finding the value of k for which this is satisfied analytically seems to be extremely difficult. While the sum in the above equation, if taken over all possible terms, is just a special case of the Vandermonde convolution, and in fact only tells us again that the total number of states is $\binom{n}{p}$, there does not

appear to be any way to remove the summation, when we are only considering partial sums.

Before I go on to tabulate some calculated values for w_{min}, I will present a more tractable upper bound. Note first the inequality:

$$w_{min} = \max_{k} R_k \leq \frac{1}{k'} \max_{k''} \sum_{r=0}^{r=k''} N_r \tag{5.1.15}$$

where k' is the smallest possible value that k could be. We also note that the largest value of the sum is $\binom{n}{p}$, and so we know:

$$w_{min} \leq \frac{1}{k'} \binom{n}{p} \tag{5.1.16}$$

Next, we note that R_k as a function of k has a single extremum which is a maximum. Also, if we assumed R_k to be continuous, then we could bracket the maximum by choosing some k such that $R_k = R_{k+1}$.(Actually, to say this requires a few extra properties in our function; it happens to be true when for sufficiently large n, and from experience, n >8) Putting this into the recurrence for R_k we get:

$$\begin{aligned}
k\, R_k &= (k+1)\, R_{k+1} - N_{k+1} \\
\Rightarrow\ k\, R_{k+1} &= (k+1)\, R_{k+1} - N_{k+1} \\
\Rightarrow\ R_{k+1} &= N_{k+1} \\
\Rightarrow\ N_{k+1} &= \tfrac{1}{k+1} \sum_{r=0}^{r=k+1} N_r \\
\Rightarrow\ &\exists r, 0 \leq r < k \text{ such that } N_r > N_k
\end{aligned}$$

In particular, since N_r also has only one maximum,

$$\Rightarrow \qquad k > m, \text{ for } m \text{ such that } N_m > N_r,\ \forall\, r, 0 \leq r < k.$$

Hence, finding the m for which N_m is maximized identifies a *least* possible value of k. It can be shown that this occurs when

$$\binom{p}{2m}\binom{n-p}{2m} = \binom{p}{2m+1}\binom{n-p}{2m+1} \tag{5.1.17}$$

n	p	w as a fraction of Basis Size			
		Lower Bound	w_{min}	Upper Bound	'Normal' Order
16	8	0.250	0.345	1.032	0.767
24	12	0.167	0.245	0.505	0.761
32	16	0.125	0.192	0.335	0.758
64	32	0.063	See below	0.143	0.738

Table 5.1: Estimated best possible widths.

(again, this equation is only strictly true for sufficiently large n – but again, the required value of n is small.) Divide the left hand side by the right to get :

$$m = \frac{np - p^2 - 1}{2n} \tag{5.1.18}$$

We then have (combining several results):

$$\frac{2}{p}\binom{n}{p} \leq w_{min} \leq \frac{2n}{np - p^2 - 2n - 1}\binom{n}{p} \tag{5.1.19}$$

Table 5.1 shows the bounds and the actual value of the optimum width, as a fraction of the total number of states (since this appears as a factor in both bounds). Note first, that we know that the first upper bound is tighter, since w is at most 1. (In fact, experimenting with the sums we have estimated showed that all of the upper bounds could be reduced by a factor of about 1/3.) The 64 orbit case is too large to be calculated exactly. It is obvious that the matrix can be made considerably more band-diagonal than it is now. However, the gains achieved are not spectacular. For example, in the 32 orbit case, the difference made by rearranging the matrix in this way will tail off when the basis is divided into more than 5 blocks. This must also be weighed against the increased complexity of programming to take advantage of the re-ordering, or indeed, the fact that we do not *know* the optimum order; we only know its properties.

Although the change in the speed of the program using this method is disappointingly small, improvements to the band density can still be made in other ways. In particular, we have ignored angular momentum considerations completely in deciding whether two states were connected. We can use the m-values of the orbits to limit the distance between connected states in the hamiltonian. For example, if the m-values are arranged in increasing order, from negative to positive, then a transition between two states like these:

0 0 0 ... 0 1 ... 1 1 1

and

1 1 0 ... 0 1 ... 1 0 0

is actually impossible as it will result in an increase in the total z-projection of angular momentum. It is seen that this eliminates precisely those matrix elements which were thought to be the problem at the start of this section. It must be noted, however, that one or other of these states will therefore be excluded from the basis. This is in fact the order used in the program, for different, historical, reasons.

In conclusion, it does not seem worth the effort to change the order of states to increase the band density, although there is some room for improvement. It is important to note, however that there are good reasons for keeping the order of single particle orbits as it is.

5.2 A Blockwise Lanczos Algorithm.

Several attempts were made over the course of this work to introduce changes to the Lanczos algorithm to take advantage of the special properties of the nuclear shell problem. The algorithm that follows has, apparently, a number of desirable features, but in the end was seen to be flawed.

Consider a matrix **H** which is to be tridiagonalised. We divide **H** into

blocks in the following manner :

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix} \tag{5.2.1}$$

The dimensions of $\mathbf{H}_{11}$ and $\mathbf{H}_{22}$ are irrelevant, but for the sake of the discussion, both will be assumed to be reasonably large. This will allow us to consider some n-step Lanczos process later, without stating n.

We now construct a Lanczos process with $\mathbf{H}_{11}$ and some starting vector q_{11}.

$$\mathbf{H}_{11}\mathbf{q}_{11} = \alpha_1\mathbf{q}_{11} + \beta_1\mathbf{q}_{12} \tag{5.2.2}$$

... etc., generating the column matrix:

$$\mathbf{Q}_1 = [\mathbf{q}_{11}, \ldots, \mathbf{q}_{1n}] \tag{5.2.3}$$

An exactly similar matrix is formed, by a Lanczos process, from $\mathbf{H}_{22}$. The tridiagonal matrices so generated are denoted $\mathbf{T}_1$ and $\mathbf{T}_2$. Now notice, that we have:

$$\mathbf{Q}_1^T\mathbf{H}_{11}\mathbf{Q}_1 = \mathbf{T}_1 \tag{5.2.4}$$

$$\mathbf{Q}_2^T\mathbf{H}_{22}\mathbf{Q}_2 = \mathbf{T}_2 \tag{5.2.5}$$

We then define:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & 0 \\ 0 & \mathbf{Q}_2 \end{bmatrix}, \tag{5.2.6}$$

$$\mathbf{X} = \mathbf{Q}^T\mathbf{H}\mathbf{Q} \tag{5.2.7}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{T}_1 & \mathbf{Q}_1^T\mathbf{H}_{12}\mathbf{Q}_2 \\ \mathbf{Q}_2^T\mathbf{H}_{21}\mathbf{Q}_1 & \mathbf{T}_2 \end{bmatrix} \tag{5.2.8}$$

$$\tag{5.2.9}$$

(we note in passing that since $\mathbf{H}^T = \mathbf{H}$, then of course, the matrix $\mathbf{X}$ above is also symmetric.)

Why might this be of interest? The answer is, that nuclear structure calculations are almost always done first in a smaller model space than we believe to be realistic; we try to add states into our calculation as and when the computing power required becomes available. Suppose the first part of the calculation correspond to the top left submatrix of $\mathbf{X}$, and the bottom right to states added into the calculation. By iterating separately in these subspaces, we have also generated many of the vectors required for calculating the off-diagonal blocks of $\mathbf{X}$. If these off-diagonal blocks can be economically calculated, *and* if the eigenvalues of $\mathbf{X}$ can be shown to converge acceptably to those of $\mathbf{H}$, then we *may* have a relatively inexpensive method of extending calculations.

This is not all; while it is unlikely that this will be as fast, or converge as fast, as a full Lanczos calculation including both subspaces, it does not need as much memory at any one time. Hence, it may be possible to do calculations using this method which would be *impossible* using a more conventional method, because we can circumvent limits of the machine. Also, we are not limited to dividing a matrix into two blocks, and the results which follow will be proved for an arbitrary number of blocks. Thus, *arbitrarily large* calculations may be attempted, the limit being now only time, and disk space, not memory.

As it turns out, there are actually severe problems with this algorithm, but we delay their discussion until towards the end of this section.

5.2.1 Some Proofs.

We begin with a simple result.

Theorem 5.1 *the eigenvalues of* $\mathbf{X}$ *converge on those of* $\mathbf{H}$.

Proof : if the dimensions of $\mathbf{Q}_1$ and $\mathbf{Q}_2$ equal those of $\mathbf{H}_{11}$ and $\mathbf{H}_{22}$, then the matrix $\mathbf{Q}$ is obviously a square orthonormal matrix, of identical dimension to $\mathbf{H}$ itself. Hence, the eigenvalues of $\mathbf{X}$ when both Lanczos processes are complete are identical to those of $\mathbf{H}$, because then $\mathbf{Q}^T\mathbf{H}\mathbf{Q}$ is a similarity transformation. The extension to an arbitrary number of blocks is obvious.

We now tighten this, using the conventional proof of convergence of the Lanczos algorithm[46]. For this, we need the mathematical baggage of a Krylov subspace $\mathcal{K}\{\mathbf{H}, \mathbf{x}, n\}$. This is simply the space spanned by all vectors of the form

$$\mathbf{k} = \sum_{i=0}^{n} k_i \mathbf{H}^i \mathbf{x} \tag{5.2.10}$$

Where k_i is any real number, and n is a non-negative integer.

Theorem 5.2 *if* $\mathbf{H}$ *is an n-by-n symmetric matrix with eigenvalues* $\lambda_1 \geq \cdots \geq \lambda_n$ *and corresponding orthonormal eigenvectors* $z_1, \ldots, z_n$*, then, after* j *steps of this modified Lanczos algorithm, the eigenvalues* $\theta_1 \geq \cdots \geq \theta_j$ *of the matrix so formed obey the relation:*

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - \frac{(\lambda_1 - \lambda_n)\tan(\phi_1)^2}{[c_{j-1}(1 + 2\rho_1)]^2}$$

where $\cos(\phi_1) = \max_{a_i, q = \sum_i a_i q_{i1}} |q^T z_1|$, $\rho_1 = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$, *and* $c_{j-1}(x)$ *is the Chebychev polynomial of degree* $j - 1$.

Proof : (Note that the only difference between the above theorem and the standard result of the Kaniel-Paige theory for the Lanczos algorithm is in the definition of q, which allows us to choose the *best* linear combination of the starting vectors in each block.)

$$\theta_1 = \max_{y \neq 0} \frac{y^T A y}{y^T y} \tag{5.2.11}$$

$$= \max_{y \neq 0} \frac{(Q_j y)^T A (Q_j y)}{(Q_j y)^T (Q_j y)} \tag{5.2.12}$$

$$= \max_{0 \neq w \in \mathcal{K}\{A, q_1, j\}} \frac{w^T A w}{w^T w} \tag{5.2.13}$$

Since λ_1 is the maximum of $w^T Aw/w^T w$ over *all* nonzero w, it follows that $\lambda_1 \geq \theta_1$. To obtain the lower bound for θ_1, note that

$$\theta_1 = \max_{p \in \mathcal{P}_{p-1}} \frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1}$$

where $\mathcal{P}_{j-1}$ is the set of all $j-1$ degree polynomials. If

$$q_1 = \sum_{i=1}^{n} d_i z_i,$$

then

$$\frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1} = \frac{\sum_{i=1}^n d_i^2 p(\lambda_i)^2 \lambda_i}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} \tag{5.2.14}$$

$$\geq \lambda_1 - (\lambda_1 - \lambda_n) \frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2}{d_1^2 p(\lambda_1)^2 + \sum_{i=1}^n d_i^2 p(\lambda_i)^2} \tag{5.2.15}$$

The lower bound can be made tight by selecting a polynomial $p(x)$ that is large at $x = \lambda_1$ in comparison to its value at the remaining eigenvalues. One way of doing this is to set

$$p(x) = c_{j-1}\left[-1 + 2\frac{x - \lambda_n}{\lambda_2 - \lambda_n}\right],$$

where $c_{j-1}(z)$ is the $(j-1)$-st Chebychev polynomial generated via the recurrence:

$$c_j(z) = 2z c_{j-1}(z) - c_{j-2}(z).$$

These polynomials are bounded by unity on [-1,1], but grow rapidly outside this interval. By defining p(x) this way, it follows that $|p(\lambda_i)|$ is bounded by unity for $i = 2, \ldots, n$, while $p(\lambda_1) = c_{j-1}(1 + 2\rho_1)$. Thus,

$$\theta_1 \geq \lambda - (\lambda_1 - \lambda_n) \frac{1 - d_1^2}{d_1^2} \frac{1}{c_{j-1}^2(1 + 2\rho_1)}$$

The desired bound is obtained by noting that $\tan(\phi_1)^2 = (1 - d_1^2)/d_1^2$.

Thus, we have proved that the lower bound can be obtained from a 'best guess' eigenvector which lies in the union of the subspaces formed by our block-wise Lanczos process. Does this give us the result we want? The answer is, not

quite. We have certainly proved that we get a lower *bound* on the convergence than we get for the normal Lanczos process with the same number of iterations — although notice that we have actually performed $m \times j$ iterations, albeit on smaller spaces. This bound, is, however, considered weak for the Lanczos process since it generally converges much faster than its bound. It is only for unusually constructed examples that this (poor) behaviour of the Lanczos algorithm is observed.

The problem with the present algorithm actually lies in a different direction; not in the rate of convergence, which is fine, but in what is being converged *to*. What goes wrong will be explained in section 5.2.3.

The algorithm we have constructed may well be better in practice than the bounds given, but currently I do not know any proof which can improve on the result given above. There are other bounds which can be put on the convergence, from perturbation theory, in particular the Weilandt-Hoffman theorem, [46] but in general (when off-diagonal contributions are large) the bounds that they give are worse than that obtained above. An approach that might prove to be of interest, but which I have been unable to follow to completion, is to look at the angle between the Krylov subspace formed by Lanczos iteration on the full space and the product space described above. The rate of convergence would be expressed in terms of this angle. This approach is appealing, since we hope that the eigenvalues of the first block of the matrix — our initial model space — will have a large overlap with the eigenvectors of the full space, i.e. it is a good approximation. This would then give us the rate of convergence in terms of a small quantity, hopefully providing a better bound.

5.2.2 Performance Estimates.

Another aspect of this algorithm which has already been mentioned, is the necessity of all of the operations involved being done cheaply. We already have the program which can do the Lanczos iterations on each block, so we only need to examine the formation of off-diagonal elements.

It should be fairly obvious from the structure of the off-diagonal block of the matrix that it can be formed by a single matrix multiplication on each Lanczos vector from the first process, followed by a series of dot products with the Lanczos vectors from the second process. The time for each dot product in this scheme is roughly equivalent to that of each one in the reorthogonalisation stage of the 'normal' Lanczos process.

We compare the time taken for the Lanczos processes which are needed to try increasing the model space one subspace at a time to doing it by the block algorithm, we have:

$$time = \sum_{m=1}^{m=M} (\alpha j(\frac{n}{m})^k + \beta j(j+1)(\frac{n}{2m})) \tag{5.2.16}$$

For normal Lanczos, compared to:

$$\begin{aligned} time &= M(\alpha j(\frac{n}{M})^k + \beta j(j+1)(\frac{n}{2M})) \\ &+ \frac{1}{2}M(M-1)(\beta j(j+1)(\frac{n}{2M})) \end{aligned} \tag{5.2.17}$$

for the block algorithm, where α and β are the constants assosciated with the matrix multiplication, and dot product, respectively.

To give an example; suppose $n = 10^6, j = 100, \alpha = 1, \beta = 10^{-3}$ and $k = 1$ — a fairly typical set of values. α and β have been set to a reasonable ratio so a comparison can be made.(see figure 5.2). Note that this figures become even better when it is considered that there may be empty blocks in the Hamiltonian far from the diagonal, arising from considerations such as parity conservation. This might seem to be very encouraging, but there is bad news on the horizon.

M	Normal 'time'	Block 'time'
2	1.575×10^8	1.075×10^8
3	1.915×10^8	1.100×10^8
4	2.178×10^8	1.125×10^8

Table 5.2: Comparison of two Lanczos Algorithms.

Adding a major shell to a nuclear shell model calculation does not add some constant number of states to the problem - it multiplies the number of states by a factor of ten or more. Thus, adding this (physically realistic) extension will be almost as difficult as the full Lanczos calculation. We could, in principle, divide the added part of the Hamiltonian up into many blocks, but, in the light of comments I will make on this process, in the next section, we would expect this to give us poorer convergence on the final result.

5.2.3 Theoretical problems

Not only are there some practical problems with using this algorithm, there are also problems with the theory around it. When I originally wrote down the proof in section 5.2.1 I believed that the convergence of the algorithm, following the infimum of the convergence limits on the subset, to be better than the convergence of the full Lanczos algorithm. However, this is a misunderstanding of the meaning of the result that has been proved. What is actually going on in the algorithm is better explained by looking at the terms of the Lanczos iteration in comparison.

Consider again our partition of the matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix} \tag{5.2.18}$$

Now we look at the terms obtained by repeatedly acting on a vector q with this matrix.

$$\mathbf{H}q = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{5.2.19}$$

$$\mathbf{H}q = \begin{bmatrix} \mathbf{H}_{11}q_1 + \mathbf{H}_{12}q_2 \\ \mathbf{H}_{21}q_1 + \mathbf{H}_{22}q_2 \end{bmatrix} \tag{5.2.20}$$

$$\mathbf{H}^2q = \begin{bmatrix} \mathbf{H}_{11}^2q_1 + \mathbf{H}_{11}\mathbf{H}_{12}q_2 + \mathbf{H}_{12}\mathbf{H}_{21}q_1 + \mathbf{H}_{12}\mathbf{H}_{22}q_2 \\ \mathbf{H}_{22}^2q_2 + \mathbf{H}_{22}\mathbf{H}_{21}q_1 + \mathbf{H}_{21}\mathbf{H}_{12}q_2 + \mathbf{H}_{21}\mathbf{H}_{11}q_1 \end{bmatrix} \tag{5.2.21}$$

That's enough terms to see what is going to happen. The equation above generates a Krylov subspace identical to that of the Lanczos process, if we work in exact arithmetic.(I present this power method since in exact arithmetic it produces the same results and the equations obtained are simpler). Now we look at a similar analysis of what happens in the new algorithm.

$$\mathbf{H}q = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix} \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \tag{5.2.22}$$

$$\mathbf{H}q \mapsto \begin{bmatrix} \mathbf{H}_{11}q_1 + \mathbf{H}_{12}q_2 \\ \mathbf{H}_{21}q_1 + \mathbf{H}_{22}q_2 \end{bmatrix} \tag{5.2.23}$$

$$\mathbf{H}^2q \mapsto \begin{bmatrix} \mathbf{H}_{11}^2q_1 + \mathbf{H}_{11}\mathbf{H}_{12}q_2 + \mathbf{H}_{12}\mathbf{H}_{22}q_2 \\ \mathbf{H}_{22}^2q_2 + \mathbf{H}_{22}\mathbf{H}_{21}q_1 + \mathbf{H}_{21}\mathbf{H}_{11}q_1 \end{bmatrix} \tag{5.2.24}$$

Note that the vectors here are representatives from the space generated by the process; not those vectors that you would get from straight multiplication with the matrix.

The matrix we end up diagonalising has several terms missing. What makes these two terms different is that they are not linear in the off-diagonal submatrices. In fact, all terms which are not linear in the off-diagonal submatrices will be missing in all subsequent steps, but these will be the only terms that we cannot generate. In other words, the method is actually a first-order perturbation method, if we do not iterate to completion. However, we did manage

to prove that the algorithm converges if we *do* continue iterating. This begs the question, how exactly is this convergence behaving?

We can find out this behaviour from looking at the proof in the second section. What we are actually showing here is that the algorithm converges to the best solution in the *unperturbed* space more quickly than pure Lanczos would. However, the component connecting these subspaces converges according to a conventional perturbation estimate (that is, the rate of convergence depends on the Frobenius norm of the off-diagonal matrices.)

5.3 Monte-Carlo Calculations

As we have seen in previous sections, traditional shell model calculations have a computational complexity that scales very poorly with the number of orbits in the calculation. For many problems with similarly poor scaling in other branches of mathematics and physics, attempts at solutions concentrate on heuristic or non-deterministic methods. In the Nuclear Structure problem, we have also seen in chapter 1 that there are a wealth of 'heuristics' – simply choose your favourite nearly-conserved symmetries, or just pick a 'reasonable' subset of the model space to work on in some other way. There are also non-deterministic methods (although these are not always presented as such) in the guise of Hartree-Fock variational methods and various statistical mechanics methods.

However, one common thread running through most of these statistical methods is the lack of a 'solid' connection to the microscopic aspects of the problem. In particular, low-lying (and thus usually single-particle) energy levels tend to be missing from any spectra obtained. Koonin and his co-workers have taken a solution to this problem most often associated with Q.C.D., and applied it to the Shell Model[31]. They claim to be able to obtain ground state properties of nuclei using realistic shell model potentials and extremely large

model spaces, with only small scaling effects from the increasing basis size. It may even be possible for them to reproduce complete low-lying spectra.

Obviously, this cuts deeply into the territory that the present work was supposed to cover. In order to properly discuss the implications of the Monte Carlo method on the future uses of the Glasgow code, I first present a description of how their method works, following closely the presentation in [47]

5.3.1 The Monte-Carlo method.

In Monte Carlo, we will look at the imaginary time evolution operator:

$$\hat{U} = \exp(-\beta \hat{H}) \, . \tag{5.3.1}$$

for some many-body Hamiltonian $\hat{H}$, and imaginary time β^{-1}. An alternative, thermodynamic, way to look at this, is that $\hat{U}$ is the partition operator for temperature β^{-1}. The operator $\hat{H}$ can be a Hamiltonian of completely general form - not just including two-body interactions, but possibly terms such as $-\omega \hat{J}_z$ in cranked shell model calculations.

There are two formalisms for extracting information from the evolution operator: the "thermal" formalism and the "zero-temperature" formalism (to which the thermal formalism reduces in the limit $\beta \to \infty$). In the thermal formalism, we begin with the partition function

$$Z = \hat{\mathrm{Tr}} \exp(-\beta \hat{H}) \, , \tag{5.3.2}$$

and then construct the thermal observable of an operator $\hat{O}$:

$$\left\langle \hat{O} \right\rangle = \frac{1}{Z} \hat{\mathrm{Tr}} \left[\hat{O} \exp(-\beta \hat{H}) \right] \, . \tag{5.3.3}$$

Here, the trace $\hat{\mathrm{Tr}}$ is over many-body states of fixed (canonical) or all (grand-canonical) particle number. In the next section, we describe how to write $\hat{U}$ in a form that allows it to be evaluated.

5.3.2 Path integral form for $\hat{U}$

We restrict the Hamiltonian to contain at most two-body terms. $\hat{H}$ can then be written, in terms of some set of 'convenient' operators $\hat{\mathcal{O}}_\alpha$:

$$\hat{H} = \sum_\alpha \epsilon_\alpha \hat{\mathcal{O}}_\alpha + \frac{1}{2}\sum_\alpha V_\alpha \hat{\mathcal{O}}_\alpha^2 \,, \tag{5.3.4}$$

where we've assumed that the quadratic term is diagonal in the $\hat{\mathcal{O}}_\alpha$. The meaning of 'convenient' will become clear shortly, but typically it refers to one-'body' operators, either one-particle ('density') or one-quasiparticle ('pairing'). The (real) V_α are the strengths of the two-body interaction.

For $\hat{H}$ in the quadratic form 5.3.4, we can express the evolution operator $\hat{U}$ as a path integral. The exponential is first split into N_t 'time' slices, $\beta = N_t \Delta\beta$, so that

$$\hat{U} = \left[\exp(-\Delta\beta \hat{H})\right]^{N_t} \,. \tag{5.3.5}$$

Then we perform the Hubbard-Stratonovich (HS) transformation on the two-body term for the n'th time slice to give eventually[48], (using the one-body hamiltonian $\hat{h}_\sigma$):

$$\hat{h}_\sigma(\tau_n) = \sum_\alpha \left(\epsilon_\alpha + s_\alpha V_\alpha \sigma_{\alpha n}\right) \hat{\mathcal{O}}_\alpha \,. \tag{5.3.6}$$

$$\begin{aligned} \hat{U} \;&=\; \int \mathcal{D}[\sigma] \exp\left(-\frac{1}{2}\int_0^\beta d\tau \sum_\alpha |V_\alpha|\, \sigma_\alpha^2(\tau)\right) \\ &\quad \times \left[\mathcal{T} \exp\left(-\int_0^\beta d\tau\, \hat{h}_\sigma(\tau)\right)\right] \,, \end{aligned} \tag{5.3.7}$$

where $\mathcal{T}$ denotes time-ordering and

$$\mathcal{D}[\sigma] = \lim_{N_t\to\infty} \mathcal{D}^{N_t}[\sigma] = \lim_{N_t\to\infty} \prod_{n=1}^{N_t} \prod_\alpha d\sigma_{\alpha n} \left(\frac{\Delta\beta|V_\alpha|}{2\pi}\right)^{\frac{1}{2}} \,, \tag{5.3.8}$$

$$\mathcal{T} \exp\left(-\int_0^\beta d\tau\, \hat{h}_\sigma(\tau)\right) = \lim_{N_t\to\infty} \prod_{n=1}^{N_t} \exp\left(-\Delta\beta\, \hat{h}_\sigma(\tau_n)\right) \,. \tag{5.3.9}$$

In the limit of an infinite number of time slices Eq. (5.3.7) is exact. In practice one has a finite number of time slices and the approximation is valid only

to order $\Delta\beta$. Rewriting the evolution operator as a path integral can make the model space tractable. Consider the case where the $\hat{\mathcal{O}}_\alpha$ are density operators. Then Eq. (5.3.5) is an exponential of two-body operators; it acts on a Slater-determinant to produce a sum of many Slater-determinants. In contrast, the path-integral formulation (5.3.7) contains only exponentials of one-body operators which, by Thouless' theorem [49], takes a Slater-determinant to another single Slater-determinant. Therefore, instead of having to keep track of a very large number of determinants (such as we do in the Glasgow code), we need deal only with one Slater-determinant at a time. Of course, the price to be paid is the evaluation of a high-dimensional integral. However, the number of auxiliary fields that need to be integrated over grows only quadratically with the size of the single particle basis while the corresponding number of Slater-determinants grows exponentially. Furthermore, the integral can be evaluated stochastically, making the problem ideal for parallel computation.

5.3.3 Monte Carlo evaluation of the path integral

Formulating the evolution operator as a path integral over auxiliary fields reduces the problem to quadrature. However, in general (when there are typically hundreds of fields), the integral must be evaluated stochastically using Monte Carlo techniques.

Using the one-body evolution operator defined by

$$\hat{U}_\sigma(\tau_2,\tau_1) = \mathcal{T}\exp\left(-\int_{\tau_1}^{\tau_2} d\tau\, \hat{h}_\sigma(\tau)\right) , \tag{5.3.10}$$

we can write Eq.

$$\zeta(\sigma) \equiv \hat{\mathrm{Tr}}[\hat{U}_\sigma(\beta,0)] , \tag{5.3.11}$$

and

$$\left\langle \hat{O} \right\rangle_\sigma = \frac{\hat{\mathrm{Tr}}\left[\hat{O}\,\hat{U}_\sigma(\beta,0)\right]}{\hat{\mathrm{Tr}}\hat{U}_\sigma(\beta,0)} . \tag{5.3.12}$$

To evaluate the path integral via Monte Carlo techniques, we must choose a normalizable positive-definite weight function W_σ. This weight function is used in Monte-Carlo to *direct* evaluation of the integral, so that the integrals converge more quickly (indeed this makes the entire process possible). We must also generate an ensemble of statistically independent fields $\{\sigma_i\}$ such that the probability density to find a field with values σ_i is W_{σ_i}. There are several possible schemes for both the choice of W and the sampling of the fields. A typical choice is $W = |\exp(-\mathcal{S})|$ and generating the samples via random walk (Metropolis) methods.

To continue: defining the 'action' by

$$\mathcal{S}_\sigma = \sum_\alpha \frac{1}{2}|V_\alpha| \int_0^\beta d\tau \sigma_\alpha(\tau)^2 - \ln \zeta(\sigma) \,, \tag{5.3.13}$$

the required observable is then found by substitution into 5.3.7. Note that $\zeta(\sigma)$ is by its definition (5.3.10,5.3.11) the time-ordering term and $\mathcal{S}_\sigma$ constitutes both that term and the exponentiated integral over β (in 5.3.7). We finally get:

$$\left\langle \hat{O} \right\rangle = \frac{\int \mathcal{D}[\sigma] \left\langle \hat{O} \right\rangle_\sigma e^{-\mathcal{S}_\sigma}}{\int \mathcal{D}[\sigma] e^{-\mathcal{S}_\sigma}} = \frac{\frac{1}{N}\sum_i \left\langle \hat{O} \right\rangle_i \Phi_i}{\frac{1}{N}\sum_i \Phi_i} \,, \tag{5.3.14}$$

where N is the number of samples,

$$\Phi_i = e^{-\mathcal{S}_i}/W_i \tag{5.3.15}$$

and $\mathcal{S}_i \equiv \mathcal{S}_{\sigma_i}$, etc. Ideally W should approximate $\exp(-\mathcal{S})$ closely. However, $\exp(-\mathcal{S})$ is generally not positive and can even be complex. In some cases, Φ_i may oscillate violently, giving rise to a very small denominator in Eq. (5.3.14) to be cancelled by a very small numerator. While this cancellation is exact analytically, it is only approximate in the Monte Carlo evaluation so that this 'sign problem' leads to large variances in the evaluation of the observable.

5.3.4 Decompositon of the Hamiltonian

To realize the HS transformation, the two-body parts of $\hat{H}$ must be cast as a quadratic form in one-body operators $\hat{\mathcal{O}}_\alpha$. As these latter can be either density operators or pair creation and annihilation operators (or both), there is considerable freedom in doing so. In the simplest example, let us consider an individual interaction term,

$$\hat{H} = a_1^\dagger a_2^\dagger a_4 a_3 \,, \tag{5.3.16}$$

where $a_i^\dagger, a_i$ are anti-commuting fermion creation and annihilation operators. In the pairing decomposition, we write (using the upper and lower bracket to indicate the grouping)

$$\hat{H} = \overbrace{a_1^\dagger a_2^\dagger} \underbrace{a_4 a_3} \tag{5.3.17}$$

$$= \frac{1}{4}(a_1^\dagger a_2^\dagger - a_3 a_4)^2 - \frac{1}{4}(a_1^\dagger a_2^\dagger + a_3 a_4)^2 - \frac{1}{2}[a_1^\dagger a_2^\dagger, a_3 a_4] \,. \tag{5.3.18}$$

The commutator is a one-body operator that can be put directly in the one-body Hamiltonian $\hat{h}_\sigma$. The remaining two quadratic forms in pair-creation and -annihilation operators can be coupled to auxiliary fields in the HS transformation.

In the application of these methods to the nuclear shell model, it is particularly convenient to use quadratic forms of operators that respect rotational invariance, isospin symmetry, and the shell structure of the system. There is enough freedom in the model to do this.

5.3.5 Limitations of Monte Carlo.

In the above discussion, there were two obvious problems with the Monte-Carlo method:

- The extremely large number of fields that must be integrated over. This means that even *small* calculations take a long time using the method

described. However, it must be realised, that the time taken by the Glasgow code for very large calculations is enormous; there comes a balance point, where the two methods take roughly the same time.

- The 'sign problem', endemic to Monte Carlo methods. In the original papers, Koonin et al. used a simple quadrupole plus pairing potential, as it can be shown that this analytically simple interaction is in fact free of the sign problem. In a later paper, it is claimed that the sign problem has been solved[50], at least for practical purposes, by the simple expedient of splitting the 'true' interaction into two parts, one free of the problem (such as the quadrupole plus pairing interaction used previously) and a 'bad' interaction. The 'good' interaction is used to solve the problem, and the result is then interpolated to what *should* be the correct answer. The difficulty with this approach does not really need pointing out; the interpolation may not be valid. However, it appears to have worked so far. One hopes that a stronger justification for this prescription can be found.

There are two further problems with the Monte Carlo method described, that occur when it is used to calculate excited states. This has been attempted many times in other fields[51], and is *extremely* difficult; for example, another student in this department, Andrew Lidsey, has spent the last three years attempting to extract a *first* excited state by Monte Carlo for lattice Q.C.D. [52]. The problem is that the Monte Carlo method does not give the complete picture of the ground state; to extract the first excited state, the state being operated on must be kept orthogonal to the ground state. This cannot be done exactly, and so errors multiply throught the calculation and eventually swamp the 'signal'. Deciding when to stop attempting to converge on the excited state because the errors are too great is a 'black art' and, so far, it seems that this procedure cannot be properly automated.

The second problem is that even if an excited state is generated, its value must be questioned. While the energy of the state can probably be relied on to some extent, the energy is insensitive to the underlying interaction and so to an extent to the structure of the state. (this was one of the considerations in fitting the original *sd*-shell interactions). However, transition rates, which tend to be of more interest to experimenters, are of interest precisely because they probe the structure of the state. In fact, even if we truncate state vectors so that the overlap between the truncated vector and the full vector is as much as 0.9 can be enough, in some circumstances, to remove *any* confidence in the transition rates predicted – see Glaudemans and Brussaard, section 10.7 [53] .

5.4 Lanczos Monte Carlo.

The limitations of Monte Carlo in reproducing excited states could be seen as a direct result of the relationship between Monte Carlo and Power Method for finding the eigenpair with the eigenvalue of largest modulus[54]. In this method we simply choose some vector with a non-zero component in the direction of the ground state, and multiply by the matrix under consideration, and normalise the result. If this process is repeated many times, the vector will converge on the desired largest eigenpair. Compare this to the Monte Carlo path integral, which repeatedly acts on some state, with the ensemble sampled eventually converging on the ground state.

The similarity here is not accidental and has occured to others over the years. We mention it here because the power method has an obvious improvement: the Lanczos Method, which is much better at resolving out the excited states as well as the ground state. The question asked is, does an analogous process exist for Monte Carlo? The answer appears to be (so far) no. The existence of such an algorithm would greatly affect the conclusions I will draw in comparing our method with Koonin's, as we shall see.

Here, I must point out that there has been a paper which *seemed* to claim that the authors were using some kind of Lanczos Monte-Carlo, namely [51]. In this paper, the authors seem to believe that they are extracting 'extra' information from the problem, by constructing a matrix from the dot products of the vectors resulting from each Monte-Carlo iteration, and then diagonalising using Lanczos. In the method they describe, the energies of the excited states so extracted will in fact converge *exactly* like those of the power method using low-precision arithmetic (with the consequent poor convergence of that algorithm). The use of Lanczos in this context is completely superfluous.

5.4.1 Using Sampled Vectors.

I finish this chapter with a short description of how a stochastic Lanczos algorithm might work. I have not seen such a description elsewhere and it may be of more interest in the future. This is a Monte-Carlo method in the sense of the definition that would be found in computing textbooks, but bears little relation to the methods usually labelled as such (it does not explicitly mention integration).

Koonin's Monte-Carlo method would be extremely difficult to convert into something more Lanczos-like, because, initially, we need vectors (or states of some kind) at the end of each iteration, and taking dot products of these provides us with the coefficients of the Lanczos matrix. In the Monte-Carlo iterations, it is not clear that at any point you can even say that the resulting vector corresponds to $H^n v_0$, for any fixed n (which would be good enough). Hence, a different approach must be sought.

Suppose instead of looking at the action of the Hamiltonian on the entire vector, we take some (probably heavily weighted) sample of m components of the vector and only operate on them. We can then ask, how could we calculate the α, β of the Lanczos matrix? The obvious route to take would be

to attempt to average out some value for α_1, then use this to help calculate β_1, then α_2, etc., just as in the usual running of the Lanczos algorithm. This is doomed to failure, because the error introduced in α_1 will be magnified at the next stage, and so on, and the reorthogonalisation which would correct this cannot be made exact, because we are only looking at 'samples' of the vectors. A further complication is that we would really like to sample v_2 at the start of the second iteration, but we only have amplitudes for the states generated from the sample of v_1.

Is there a way round this? The answer is yes, in principle. (I say in principle because what follows has not yet been tested in practice). Consider taking samples size m from a vector v_1 of dimension n. Then we firstly want to find H' so that:

$$\overline{v_1.H'.v_1} = v_1.H.v_1 = \alpha_1 \tag{5.4.1}$$

Where the average is taken over all samples, and H is the true Hamiltonian. This is actually fairly easy to do. All that is required is a weighting on the diagonal matrix elements to take account of the fact that they will be generated much more often than the off diagonal matrix elements between any two states. There are $\binom{n}{m}$ possible sample vectors of length m in a basis of size n. Given a vector containing a state s_1, it is obvious that only $\frac{1}{n-1}$ of the samples containing this state will also contain some other particular state s_2. If we used all possible samples, we could find the correct resulting vector by using the weighted Hamiltonian:

$$H'_{ij} = \frac{1}{\binom{n}{m}}(H_{ij}(n-1-\delta_{ij}(n-2))) \tag{5.4.2}$$

We have got through the first iteration, but how do we get the correct values in the *rest* of the Lanczos matrix? The answer is simple: we guess. The beauty of guessing the β values, choosing the same set of values for many samples, is that it allows us to use a single sample over all iterations without

referring to the other samples until all of the iterations are complete. We have used a Hamiltonian which, in principle, averages out to give us the correct vectors. By guessing all the Lanczos matrix entries, probably wrongly, we can calculate (by averaging) the residual at each iteration, the norm of which gives us a measure of how far we are away from the 'true' Lanczos matrix. This measure can be used in any number of optimisation routines to move - or more likely, crawl - towards the correct values.

An actual implementation of this would require the measured residuals to have converged before the next step in the fitting of the αs and βs. This will, almost certainly, be extremely slow. There is at least one way to accelerate the convergence. Note that in the 'usual' Monte-Carlo scheme a weighting function is used to bias the choice of samples, in order to speed up convergence. A roughly equivalent thing can be done in this algorithm: matrix elements can be weighted so that they are less likely to be chosen if they involve a greater change in the energy of the Slater determinant involved, as taken from (e.g.) a harmonic oscillator potential. The matrix elements supressed in this way have to be multiplied by a large factor when actually chosen, so that the averages are preserved. The effect of a very large weighting of this type will be to cause convergence in the $0\hbar\omega$ space first, with further iteration (getting tighter bounds on the residuals) including more and more the effects of higher-energy shells.

I don't really believe that this has the potential to form a true calculation technique. In its favour, it is completely parallelisable; each sample can be handled independently. However, each 'guess' at the whole parameter set would take many, many samples to evaluate; and with upwards of 200 variables to optimise over, this could take forever. There is also the question of how stable we can consider the results to be. The method seems worth mentioning, though, if only for its curiosity value.

Chapter 6

Conclusions.

6.1 Parallelization of the Glasgow Code.

This part of the project has been successfully completed, with the resulting code showing a marked speedup on serial versions of the code. There is a working version of the parallel code on
`halloween.elec.gla.ac.uk` (IP address 130.209.176.49)
in the directory:
`/home/halloween/physics/brian/PAR`. The PARIX command 'run' should be used to invoke it, like so:
`run -c 0 4 4 bootboy` (bootboy is a bootstrap program which loads the paralled code onto the network.)

Of the different versions of the code that were written, Scott's version of parallel Lanczos seems to be the least prone to the problems common to the parallel methods, specifically deadlock and large communications overheads.

However, it is an expensive algorithm to implement, as it uses many more processors to do the calculations we want to do than could be afforded by even the better off universities. It seems unlikely that in the near future we will be able to perform calculations with tens of millions of states, although a few

million states is now at the limit of our capabilities.

6.2 Comparison to other Codes.

During the completion of this work the Monte Carlo methods of Koonin et al. have come to the fore, and have proved faster at getting some of the information on nuclei in larger model spaces. We must temper this with the knowledge that this version of Monte Carlo is virtually useless at getting excited state energies; a problem endemic to the method in all areas of physics. The F.D.S.M. method also has its limitations: it relies on symmetries which are not exactly conserved in real nuclei, so there will remain calculations which test whether or not a state belongs to a band of some symmetry which cannot be done within that model. This same criticism applies to all collective models. Having carved out this territory of excited mixed-symmetry states, there still remain some rival codes: notably OXBASH, which also perform full shell model calculations, but in a different way. The problem with these codes is that they seem to be even less inherently parallel than even the Glasgow code. Because of this, it seems that the Glasgow code is, and will remain, the best suited to very large basis calculations of this type. However, there are not very many calculations of this type of intermediate basis size (a couple of million states). To get the kind of information required, i.e. a $1\hbar\omega$ model space, seems to require more states than the code can handle, while smaller problems (of a few hundred thousand states) are better handled by OXBASH or the serial Glasgow code.

6.3 Future work.

It seems to me that the parallel Glasgow code is dead in the water when it comes to the shell model calculations it has been traditionally applied to. For

ground state calculations, Monte Carlo is now to be recommended, while for low excited states, the various collective models now have a sufficient degree of sophistication to outrun our program, despite the most strenuous efforts on our part to construct a faster code. It does not seem to me that there are sufficient applications of very large basis calculations which are at once feasible and interesting, even though we have pointed out in this thesis several examples of this type (the band termination calculations). Even these have to be drastically truncated compared to what we would consider satisfactory.

Also, we know that the current generation of shell model codes would have to be the last for some time in any case; the combinatoric growth in the size of the problems being considered, at the next step, will far outstrip any computer we could hope to have in the next ten years or so. I would therefore recommend that the parallelisation of the code as it stands should stop here, and attention should once again be turned to the physics of the problem to increase the sizes of calculation we can achieve.

All is not yet lost, however. There remain a number of applications where the Glasgow code is used in different guises, which are likely to benefit from the greater speed, for moderately large problems. One example of this is the cranked shell model example given in chapter 4. Here, the *physics* of the problem is such that there is no other code today that can solve this type of problem in the manner we have described.

A second example of where the code may be used is in the Quark Shell Model calculations being done in this group by Sandy Watt et al [55]. These calculations are intended to help bridge the gap between high- and low- energy physics, by eventually calculating from a QCD-induced potential the nucleon-nucleon interaction. While the interactions are very different in this program and our own, they share a common heritage in the original Glasgow code, and, without a great deal of effort, the central routine for matrix multiplication in the QSM code could be replaced by the parallelized parts of the present work.

In short, the future of the Glasgow shell model code is no longer in the nuclear shell model of Mayer; futher work should concentrate on finding new applications for the code, and to producing interesting physics in other areas.

The tenth time, just a year ago,
I made myself a little list
Of all the things I'd ought to know,
Then told my parents, analyst,
And everyone who's trusted me
I'd be substantial, presently.

I haven't read one book about
A book or memorized one plot.
Or found a mind I did not doubt.
I learned one date. And then forgot.
And one by one the solid scholars
Get the degrees, the jobs, the dollars.

And smile above their starchy collars.
I taught my classes Whitehead's notions;
One lovely girl, a song of Mahler's.
Lacking a source-book or promotions,
I showed one child the colors of
A luna moth and how to love.

W.D. Snodgrass, from 'April Inventory'.

Appendix A

Portable Communication Routines

This appendix contains a file of small subroutines which handled all of the communication in every version of the parallel code. This was a deliberate effort on my part to separate the working of the code from the detail of the operating system that it ran on. Generally, a system has one or more methods of passing messages (PARIX, for instance, has at least 6 different commands which will send a message), a global structure which stores the network data, and some error handling mechanism. This program was then written to handle all of those features transparently - to change the architecture on which the program is being run, or the type of communication desired, only this file need be altered. Of course, this also hides the real difficulty of getting a program to run using any one of the mechanisms in the first place.

This particular version is the MEIKO `cs_tools` code. The routines are self explanatory, with the exception of `set_up_net`, which in this case builds bidirectional channels by registering a processor in some table, then allowing each processor to look the others up in this table. The PARIX version of this particular routine is much more complicated: the channels between processors,

while still being bidirectional, have a definite start and end. Hence the start processor must set up its end of the net first; and the routine is written to force a processor to set things up in this, correct, order.

```
/* MEIKO communication masker
*/
#define MASTER
#include "std.h"

void get_from(ch,length,data)
int ch;
int length;
char *data;
{
netid_t *id;
char message[20];
int sent;

id=&(netids_[ch]);
if(length!=(sent=csn_rx(trans_,id,data,length)))
{
sprintf(message,"rx:%d:%d/%d",ch,sent,length);
fatalerror(message);
}
}

void send_to(ch,length,data)
int ch;
int length;
```

```
char *data;
{
netid_t *id;
char message[20];
int sent;

id=&(netids_[ch]);
if(length!=(sent=csn_tx(trans_,0,*id,data,length)))
{
sprintf(message,"tx:%d:%d/%d",ch,sent,length);
fatalerror(message);
}
}

void set_up_net(argc,argv)
int *argc;
char **argv[];
{
char label[20];
int id;

csn_init(argc,argv);
maxids_= atoi((*argv)[1]);
home_ = atoi((*argv)[2]);
netids_=(netid_t *)malloc(maxids_*sizeof(netid_t));
        if((csn_open(home_, &trans_)) != CSN_OK)
              fatalerror("cs error");
sprintf(label,"task%d",home_);
        if((csn_registername(trans_,label)) != CSN_OK)
```

```
                fatalerror("cs error");
for(id=0;id<maxids_;id++)
{
if(id!=home_)
{
sprintf(label,"task%d",id);
        if((csn_lookupname(&(netids_[id]),label,1)) != CSN_OK)
                fatalerror("cs error");
}
}
}

void fatalerror(message)
char *message;
{
cs_abort(message,-1);
}

void numerror(num)
int num;
{
char message[5];

sprintf(message,"%d",num);
fatalerror(message);
}

/* END */
```

Appendix B

Mathematica Routines.

As a first stage in converting the routines for electric and magnetic transition rates into C, for inclusion in the full program, a set of Mathematica [56] routines were written to calculate most of the various coupling coefficients required in the evaluation of Brody-Moshinsky brackets [57] , among other things. This was done partly because it was easier to rewrite these routines from scratch than convert the old FORTRAN code. On the plus side, Mathematica offered an environment where the rewritten routines could be easily tested, and it also has a function which produces C code directly from the Mathematica source.

The routines are reproduced in full here, as they are fairly short, could prove useful in future, and are not to be found in a simple algorithmic form (as opposed to the mathematical forms), or even together, in any journal I have come across. I will comment further on each routine as I present them.

B.1 General Routines

For the main procedures to work, some general mathematical routines should be defined.

```
Phase/: Phase[a_] := (-1)^a
```

```
Fac/: Fac[0] := 1
Fac/: Fac[a_] := Fac[a] = a*Fac[a-1] /; a>0

Facs/: Facs[a_] := Fac[a_]
Facs/: Facs[a_,b__] := Fac[a]*Facs[b]

DFac/: DFac[0] := 1
DFac/: DFac[1] := 1
DFac/: DFac[a_] := DFac[a] = a*DFac[a-2] /; a>1

Gam/: Gam[a_] := Gam[a] = DFac[2*a+1]/2^(a+1)

AngMomDelta/: AngMomDelta[j1_,j2_,j_] :=
        Facs[(j1+j2)-j,(j+j1)-j2,(j2+j)-j1]/Fac[j1+j2+j+1]
JTest/: JTest[a_,b_,c_] :=
        If[ c <= Abs[a-b] || c >= a+b, True, False]
```

`Phase` is self-explanatory; `Fac` is the factorial function. In common with many of the functions that follow, it is defined so that it stores values that it calculates, for immediate recall if the same factorial is ever required again. This technique greatly increases the speed of all of the functions defined in the rest of this appendix. `Facs` is simply an abbreviation for multiplying several factorials together. `DFac` is the double factorial function. `Gam` is defined by:

$$\texttt{Gam[a]} = \frac{\Gamma(a+\frac{3}{2})}{\sqrt{\pi}}$$

for integers, and is used for half-integer values of the Euler gamma function, using `DFac` for integer values.

`AngMomDelta` is the square of the usual definition of the angular momentum delta defined in Pal [58]. The square is used so that the square root is taken

over other terms simultaneously, again speeding up this heavily-used routine. Finally, `JTest` tests sets of angular momenta which are to be coupled to see if they satisfy the triangle inequality.

B.2 3-j symbols

In this section, `CGSum` is the inner sum of the formula for a Clebsch-Gordan coefficient, taken from the appendix of Pal [58]. The constraints on the sum are taken from the condition that all factorials must be of positive numbers.

```
CGSum/: CGSum[j_, j1_, j2_, m1_, m2_] :=
        Sum[Phase[k]/
                Facs[k, (j1+j2)-j-k, j1-m1-k, (j2+m2)-k,
                        j-j2+m1+k, j-j1-m2+k],
                {k,Max[0,j2-j-m1,j1-j+m2],
                        Min[(j1+j2)-j,j1-m1,j2+m2]}]
```

The next three functions are actual 3-j symbols: `CG0` is a Clebsch-Gordan coefficient with all z-projections zero [58]. This is somewhat simpler and is included to speed up the larger coupling coefficients. `Clebsch` is the usual Clebsch-Gordan coefficient, while `Wigner` is the symmetric 3-j coefficient found in Edmonds [59]. It should be pointed out here that in a C implementation, it becomes more efficient to double all angular momenta and work purely with integers. The Mathematica versions with this innovation are needlessly lengthy for inclusion here, and further complicate later routines.

It is worth noting in passing that a Clebsch-Gordan coefficient package is supplied with Mathematica [56], which *fails* for non-integer values of the angular momenta.

```
CG0/: CG0[j1_, j2_, j_] := CG0[j1,j2,j] =
```

```
        Block[{g = (j1+j2+j)/2},
                If[Mod[2*g, 2] == 1, 0, (Phase[j1-j2+g]*
                Sqrt[(2*j+1)*AngMomDelta[j1, j2, j]]*Fac[g])
                /Facs[g-j1, g-j2, g-j]]]
Wigner/: Wigner[j1_, m1_, j2_, m2_, j_, m_] :=
        (Phase[j2-j1+m]*Clebsch[j1,m1,j2,m2,j,-m])
                /Sqrt[2*j+1]
Clebsch/: Clebsch[j1_, m1_, j2_, m2_, j_, m_] :=
        If[m != m1+m2 || j1 >= Abs[m1] ||
        j2 >= Abs[m2] || j >= Abs[m] ||
        JTest[j1, j2, j], 0, CGSum[j, j1, j2, m1, m2]*
        Sqrt[AngMomDelta[j, j1, j2]*
        Facs[j-m, j1-m1, j2-m2, j+m, j1+m1, j2+m2]*(2*j+1)]]
```

B.3 Six-j and Nine-j Symbols.

We continue by defining the larger coefficients. There are three kinds of six-j coefficients; these are Jahn's U coefficient, Racah's W, and the symmetric six-j [60]. These are named, obviously, `JahnU`, `RachaW`, and `SixJ`. Also in this section is the symmetric nine-j symbol, `NineJ`. This uses rearrangements to reduce the number of terms in the sum, as suggested in Pal [61].

```
JahnU/: JahnU[a_, b_, c_, d_, e_, f_] :=
        RacahW[a, b, c, d, e, f]/Sqrt[(2*e+1)*(2*f+1)]
Racah/: RacahW[a_, b_, c_, d_, e_, f_] :=
        Phase[a+b+d+e]*SixJ[a, b, c, d, e, f]
SixJ/: SixJ[a_, b_, c_, d_, e_, f_] :=
        SixJ[a, b, c, d, e, f] =
        Sqrt[AngMomDelta[a, b, c]*AngMomDelta[a, e, f]
```

```
                *AngMomDelta[d, b, f]*AngMomDelta[d, e, c]
                *SixJSum[a, b, c, d, e, f]
SixJSum/: SixJSum[a_, b_, c_, d_, e_, f_]:=
        Sum[(Phase[n]*Fac[n+1])/
                Facs[n-a-b-c, n-a-e-f, n-d-b-f, n-d-e-c,
                        (a+b+d+e)-n, (b+c+e+f)-n,
                                (c+a+f+d)-n],
                {n, Max[a+b+c, a+e+f, d+b+f, d+e+c],
                        Min[a+b+d+e, b+c+e+f, c+a+f+d]}]
NineJ/: NineJ[a_, b_, c_, d_, e_, f_, g_, h_, i_] :=
        Block[{m},
        If[(m = Min[a, b, c, d, e, f, g, h, i]) !=i,
        Which[
        m==a,NineJ[e, f, d, h, i, g, b, c, a],
        m==b,NineJ[f, d, e, i, g, h, c, a, b],
        m==c,NineJ[d, e, f, g, h, i, a, b, c],
        m==d,NineJ[h, i, g, b, c, a, e, f, d],
        m==e,NineJ[i, g, h, c, a, b, f, d, e],
        m==f,NineJ[g, h, i, a, b, c, d, e, f],
        m==g,NineJ[b, c, a, e, f, d, h, i, g],
        m==h,NineJ[c, a, b, f, d, e, i, g, h]],
        Sum[Phase[2*k]*(2*k+1)*(SixJ[a, d, g, h, i, k]
        *SixJ[b, e, h, d, k, f] *SixJ[c,f,i,k,a,b],
        {k, Max[Abs[a-i], Abs[b-f], Abs[d-h]],
        Min[a+i, b+f, d+h]}]]]
```

B.4 Brody-Moshinsky Brackets

There are several versions of the harmonic oscillator brackets; here we use a formula due to Bakri, as quoted in Lawson [62].

Bakri's formulation is used here in preference to that of Baranger and Davies [63] soley because it is easier to write in small testable chunks. Although Baranger and Davies use stretched 9-j symbols and so reduce the eventual number of 6-j symbols that must eventually be calculated to get them, Bakri's formula has a sum over fewer 9-j symbols and the two are actually equivalent.

There is yet another way of calculating Moshinsky Brackets, which involves much less algebra. This is direct diagonalisation of an appropriate matrix operator, as discussed in [64]. This method sticks closely to the ideology followed in the Glasgow code, that the human end of the algebra should be simple, with the computer then doing many simple calculations to finish with. The diagonalization method of this paper could also be written very quickly by adapting some of the routines already in the code.

Also in this section are the B-coefficients of Brody and Moshinsky, [57] (here called `Brody`), and the coefficients of the p-th Talmi integral, here called `Cp`.

```
Mosh/: Mosh[n1_, l1_, n2_, l2_, n3_, l3_, n4_, l4_, l1_] :=
        N[Block[{m},
        If[
        (m = Min[2*n1+l1,2*n2+l2,2*n3+l3,2*n4+l4])!=2*n1+l1,
        Which[
        m==2*n2+l2,Phase[l3-l1]
                *Mosh[n2,l2,n1,l1,n3,l3,n4,l4,l1],
        m==2*n3+l3,Phase[l4+l2]
                *Mosh[n3,l3,n4,l4,n1,l1,n2,l2,l1],
        m==2*n4+l4,Phase[l4+l1]
```

```
                *Mosh[n4,l4,n3,l3,n1,l1,n2,l2,ll]],
        (BakriS[2*n1+l1, l1, l2, n3, l3, n4, l4 ll]
        *BakriA[n3, l3]*Bakri[n4, l4])
                /(4*BakriA[n1, l1]*BakriA[n2, l2])]]]
BakriS/: BakriS[en_, l1_, l2_, n3_, l3_, n4_, l4_, ll_] :=
        Sum[BakriL[k1, k2, k3, k4, l1, l2, l3, l4]
        *NineJ[k1, k3, k4, l1, l2, l3, l4]
        *BakriV[k1, k2, k3, k4, n3, l3, n4, l4,
         Floor[(en-k1-k3)/2]],
        {k1, 0, en}, {k2, Abs[l3-k1], l3+k1},
        {k3, Abs[l1-k1], l1+k1},
        {k4, Max[Abs[l2-k2], Abs[l4-k3]], Min[k2+l2, k3+l4]}]
BakriL/: BakriL[k1_, k2_, k3_, k4_, l1_, l2_, l3_, l4_] :=
        Phase[k3]*(2*k1+1)*(2*k2+1)*(2*k3+1)*(2*k4+1)
        *CG0[k1, k2, l3]*CG0[k3, k4,l4]
        *CG0[k1,k3,l1]*CG0[k2,k4,l2]
BakriV/: BakriV[k1_,k2_,k3_,k4_,n3_,l3_,n4_,l4_,v2_] :=
        Sum[BakriF[v1, n3, l3, k1, k2]
        *BakriF[v2-v1, n4, l4, k3, k4],{v1, 0, v2}]
BakriF/: BakriF[v_, n_, l_, j_, k_,] :=
        Block[{en = 2*n+l}, If[Mod[en-j-k, 2] == 1 ||
        (en-j-k)/2-v < 0 || ((en+k)-j)/2-v < 0, 0,
        (Fac[n]*Gam[n+l])/
        (Sqrt[2]^en*Fac[c]*Fac[(en-j-k)/2-v]
        *Gam[((en+k)-j)/2-v]*Gam[j+v])]]
BakriA/: BakriA[n_, l_] :=
        Phase[n]*Sqrt[1/(DFac[2*n]*DFac[2*l+2*n+1])]


Brody/: Brody[n1_, l1_, n2_, l2_, p_] :=
```

```
        N[ Block[{l = (l1+l2)/2}, If[Mod[2*l,2] == 1, 0,
        (Phase[p-1]*Fac[2*p+1]*
        Sqrt[Facs[n1, n2, 2*n1+2*l1+1, 2*n2+2*l2+1]/
        Facs[n1+l1, n2+l2]]*
        Sum[Facs[k, 2*l1+2*k+1, n1-k, (2*p-l1+l2)-2*k+1,
        n2-p+l+k, p-l-k],
        {k, Max[0, p-l-n2], Min[n1,p-l]}])
        /(2^(n1+n2)*Fac[p])]]]

Cp/: Cp[n1_, l1_, n2_, l2_, n3_, l3_, n4_, l4_, ll_, p_] :=
        Block[{en1 = n1+n2+(l1+l2)/2, en2 = n3+n4+(l3+l4)/2},
        Sum[
        Mosh[na,la,nb,2*(en-na-nb)-la,n1,l1,n2,l2,ll]
        *Mosh[(na+en2)-en1,la,nb,
                2*(en-na-nb)-la,n3,l3,n4,l4,ll]
        *Brody[na,la,(na+en2)-en1,la,p], {na,0,en},
        {la, 0, 2*(en1-na)}, {nb, 0, en1-na-la/2}]]
```

Bibliography

[1] R. R. Whitehead, S. Watt, B. J. Cole, and I. Morrison. The Glasgow Shell Model Code. *Advances in Nuclear Physics*, 9:123–176, 1977.

[2] Mohammed Riaz. *Transputer Implementation for the Shell Model and SD Shell Calculations.* PhD thesis, University of Glasgow, 1990.

[3] P. J. Brussaard and P. W. M. Glaudemans. *Shell Model Applications in Nuclear Spectroscopy*, page 3. North-Holland, 1977.

[4] M. G. Bowler. *Nuclear Physics*, page 45. Pergamon Press, 1973.

[5] Maria Goeppert-Mayer. *Physical Review*, 75:page 1968, 1949.

[6] Dorothy Stein. *Ada: A Life and a Legacy.* London MIT Press, 1983.

[7] Amos de Shalit and Igal Talmi. *The Nuclear Shell Model*, page 192. New York Academic Press, 1963.

[8] N. I. Kassis, J. P. Elliott, and E.A. Sanderson. A density-dependent version of the Sussex interaction. *Nuclear Physics A*, 359:386–396, 1981.

[9] B. A. Brown, W. A. Richter, and B. H. Wildenthal. Spin Tensor Analysis of a New Empirical Shell-Model Interaction for the 1S–0D Nuclei. *Journal of Physics G*, 11(11):1191–1998, 1985.

[10] Peter Ring and Peter Schuck. *The Nuclear Many-Body Problem*, pages 172–174. Springer-Verlag, 1980.

[11] R. R. Whitehead, S. Watt, B. J. Cole, and I. Morrison. The Glasgow Shell Model Code. *Advances in Nuclear Physics*, 9:168–171, 1977.

[12] K. Langanke, J. A. Maruhn, and S. E. Koonin. *Computational Nuclear Physics 1: Nuclear Structure*, pages 1–27. Springer-Verlag, 1991.

[13] A.E.L. Dieperink and G. Wenes. The Interacting Boson Model. *Adv. Rev. Nucl. Part. Sci.*, page 77, 1985.

[14] Aage Bohr and Ben R. Mottelson. *Nuclear Structue Volume I*, page 38. W. A. Benjamin Inc., 1975.

[15] D. R. Tilley and J. Tilley. *Superfluidity and Superconductivity*, page 119. Adam Hilger Ltd, 1986.

[16] A. C. Merchant and W. D. M. Rae. Alpha Chain States in 4N-Nuclei. Technical report, University of Oxford Dept. of Physics, June 1993. Invited contribution to 2nd International Conference on Atomic and Nuclear Clusters, Santorini, Greece.

[17] K. Langanke, J. A. Maruhn, and S. E. Koonin. *Computational Nuclear Physics 1: Nuclear Structure*, pages 152–169. Springer-Verlag, 1991.

[18] R. R. Whitehead, S. Watt, B. J. Cole, and I. Morrison. The Glasgow Shell Model Code. *Advances in Nuclear Physics*, 9:145–148, 1977.

[19] B. A. Brown, A. Etchegoyen, and W. D. M. Rae. Oxbash. Technical Report MSU-NSCL 524, Michigan State University, 1985.

[20] K. Langanke, J. A. Maruhn, and S. E. Koonin. *Computational Nuclear Physics 1: Nuclear Structure*, pages 152–169. Springer-Verlag, 1991.

[21] James Gleick. *CHAOS: Making a New Science*, pages 231–232. Heinemann, 1982.

[22] Alan J. Bell. *Complex Zeros Of The Partition Function In Lattice Chromodynamics.* PhD thesis, University of Glasgow, 1991. pages 4–6.

[23] Sun Microsystems. *SunOS Reference Manual*, chapter `at(1)`, pages 30–31. Sun Microsystems, Inc., 1990.

[24] Isaac D. Scherson and Peter F. Corbett. Communications Overhead and the Expected Speedup of Multidimensional Mesh-Connected Parallel Processors. *Journal of Parallel and Distributed Computing*, 11(1):86–96, 1991.

[25] Mohammed Riaz. *Transputer Implementation for the Shell Model and SD Shell Calculations.* PhD thesis, University of Glasgow, 1990. pages 57–61.

[26] Mohammed Riaz. *Transputer Implementation for the Shell Model and SD Shell Calculations.* PhD thesis, University of Glasgow, 1990. pages 62–64.

[27] Dominic Prior, Nick Radcliffe, Mike Norman, and Lyndon Clarke. What Price Regularity? Technical Report ECSP-TR-3, Edinburgh Concurrent Supercomputer Project, 1989.

[28] Lyndon J. Clarke. TINY: Version 1.0 Discussion and User Guide. Technical Report ECSP-UG-9, Edinburgh Concurrent Supercomputer Project, May 1989.

[29] Chris Brown and Michael Rygol. Marvin: Multiprocessor Architecture for Vision. In *Proceedings of the 10th Occam User Group Technical Meeting*, April 1989.

[30] Michael. R. Garey and David S. Johnson. *Computers and Intractability*, chapter 1. W. H. Freeman and Co., 1979.

[31] C.W.Johnson, S.E.Koonin, G.H.Lang, and W.E.Ormand. Monte Carlo Methods for the Nuclear Shell Model. *Physical Review Letters*, 69(22):3157–3160, 1992.

[32] Michael V. Berry. Quantum Chaology, Prime Numbers, and Riemann's Zeta Function. In *Nuclear and Particle Physics 1993*, pages 133–134. Institute of Physics, 1993. IOP conference series number 133.

[33] Beyond the Supercomputer: Parsytec GC. Parsytec promotional booklet, 1991.

[34] Mike Norman and Nick Stroud. Introducing the Edinburgh Concurrent Supercomputer. Technical Report ECSP-UG-001, Edinburgh Concurrent Supercomputer Project, April 1989.

[35] A. R. Gourlay and G. A. Watson. *Computational Methods for Matrix Eigenproblems*, pages 71–78. John Wiley and Sons, 1973.

[36] David Harel. *Algorithmics: The Spirit of Computing*, page 84. Addison-Wesley, 1987.

[37] David S. Scott. Implementing Lanczos-like algorithms on Hypercube Architectures. *Computer Physics Communications*, 53(1–3):271–281, 1989.

[38] D.Kurath. *Physical Review*, 101:216, 1956.

[39] Peter Ring and Peter Schuck. *The Nuclear Many-Body Problem*, page 180. Springer-Verlag, 1980.

[40] T.T.S. Kuo and G. E. Brown. Structure of Finite Nuclei and the Free Nucleon-Nucleon interaction. *Nuclear Physics*, 85:40, 1966.

[41] V.G.J. Stoks, R.A.M. Klomp, M.C.M. Rentmeester, and J.J. de Swart. Partial-wave analysis of all nucleon-nucleon scattering data below 350 MeV. *Physical Review C*, 48(2):792–815, 1993.

[42] I. F. Wright, W. J. Vermeer, and J. Billowes. Experimental Candidates for States at or Beyond Rotational Band Terminations in S-D shell Nuclei.

Contribution to Symposium in Honour of Akito Arima, Santa FE, May 1990.

[43] D. Branford, N. Gardner, and I. F. Wright. Evidence for Negative Parity Rotational Bands in 24Mg. *Physics Letters B*, 36(5):456, 1971.

[44] P. J. Brussaard and P. W. M. Glaudemans. *Shell Model Applications in Nuclear Spectroscopy*, pages 106–118. North-Holland, 1977.

[45] J.A.Sheikh, N. Rowley, M. A. Nagarajan, and H.G.Price. Neutron-Proton Interactions in the Mass-80 Region. *Physical Review Letters*, 64(4):376–379, 1990.

[46] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, pages 323–331. Johns Hopkins University Press, 1989.

[47] C.W.Johnson, S.E.Koonin, G.H.Lang, and W.E.Ormand. Monte-Carlo evaluation of Path Integrals for the Nuclear Shell Model. *Physical Review C*, 48(4):1518–1545, 1993.

[48] J. Hubbard. *Physical Review Letters*, 3:77, 1959.

[49] Peter Ring and Peter Schuck. *The Nuclear Many-Body Problem*, page 615. Springer-Verlag, 1980.

[50] Y. Alhassid, D.J.Dean, S.E.Koonin, G.H.Lang, and W.E.Ormand. Practical Solution to the Monte-Carlo Sign Problem: Realistic Calculations of ^{54}Fe. *Physical Review Letters*, 72(5):613–616, 1994.

[51] M. Caffarel, F. X. Gadea, and D. M. Ceperley. Lanczos-type Algorithm for Quantum Monte-Carlo Data. *Europhysics Letters*, 16(3), 1991.

[52] Andrew Lidsey, C.T.H. Davies, A. Lagnau, G.P. Lepage, J. Shigemitsu, and J. Sloan. Precision Υ Spectroscopy From Nonrelativistic Lattice

Q.C.D. Technical Report FSU-SCRI-94-57, Florida State University, 1994.

[53] P. J. Brussaard and P. W. M. Glaudemans. *Shell Model Applications in Nuclear Spectroscopy*, page 232. North-Holland, 1977.

[54] A. R. Gourlay and G. A. Watson. *Computational Methods for Matrix Eigenproblems*, pages 38–42. John Wiley and Sons, 1973.

[55] Leila Ayat. *Spontaneous Creation of Quark-Antiquark Pairs in Few-Quark Systems.* PhD thesis, University of Glasgow, 1990.

[56] Stephen Wolfram. *Mathematica: A System for doing Mathematics by Computer.* Addison-Wesley, 1990.

[57] T. A. Brody and M. Moshinsky. *Tables of Transformation Brackets for Shell Model Calculations.* Gordon and Breach, second edition, 1967.

[58] Manoj Kumar Pal. *Theory of Nuclear Structure*, page 601. Van Nostrand Rheinhold Company Inc., 1983.

[59] A. R. Edmonds. *Angular Momentum in Quantum Mechanics*, page 45. Princeton University Press, 1957.

[60] A. R. Edmonds. *Angular Momentum in Quantum Mechanics*, page 90. Princeton University Press, 1957.

[61] Manoj Kumar Pal. *Theory of Nuclear Structure*, page 605. Van Nostrand Rheinhold Company Inc., 1983.

[62] R. D. Lawson. *Theory of the Nuclear Shell Model*, pages 492–499. Oxford Clarendon Press, 1980.

[63] M. Baranger and K.T.R. Davies. Oscillator Brackets for Hartree-Fock Calculations. *Nuclear Physics*, 79:403, 1966.

[64] J. D. Talman and A. Lande. Computation of Moshinsky brackets by Direct Diagonalization. *Nuclear Physics A*, 163:249, 1971.