Mathematical and Computational Applications, Vol. 11, No. 1, pp. 41-49, 2006.

TIME DEPENDENCE OF SHELL MODEL CALCULATIONS

E. Dikmen

Süleyman Demirel University, Isparta, Turkey, edikmen@fef.sdu.edu.tr

Abstract- In this study, time dependence of large-scale nuclear shell model calculations in a parallel cluster is investigated. Shell model calculations have been done for the light Sb isotopes by using serial and parallel versions of Drexel University Shell Model (DUSM) code and the time dependence of these calculations are obtained on the local Cyborg Parallel Cluster at Drexel University. Some analyses about what kind of parallel system we need to do the calculations have been done in order to do the nuclear shell model calculations for the other nuclei.

Keywords- Shell model, parallel system

1. INTRODUCTION

The shell model problem aims at solving the many-body system in a set of valence shells in terms of independent particle wave functions with the dynamics specified by the Hamiltonian

$$H = \sum \varepsilon_\alpha a_\alpha^+ a_\alpha + \frac{1}{2}\sum V_{\alpha\beta\gamma\delta}^{effective} a_\alpha^+ a_\beta^+ a_\delta a_\gamma \qquad (1.1)$$

The first term in this equation is the one-body term which describes the interaction of the valence nucleons with the closed core. The second term is the two-body term which describes the two-body interaction. The theoretical details of the calculations are given in Ref. [1]. The solution implies deriving systematically the energy spectra, electromagnetic transitions, and beta decay of series of nuclei in specific region of the periodic table.

The solution of nuclear model implied by the Hamiltonian above is extremely complex. It is normally divided in two sub-problems, each one being very complex in its own right. The first one is the establishment of an effective interaction $V^{effective}$ from the bare nucleon-nucleon force in a given valence space and of a set of single particle energies ε for sequences of nuclei. In order to obtain an effective two-body interaction for shell model studies, one starts with a free nucleon-nucleon (NN) interaction V which is appropriate for nuclear physics at low and intermediate energies. The second problem is that the many-body calculations that follows the establishment of the effective interaction. Each of these sub-problems is by itself very complex and challenging. The successes of the nuclear shell model can be traced to the development of powerful codes to implement the approach.

The calculations described basically above involved Hamiltonian matrices that reached dimensions in excess of 1,900,000. These are very large matrices that require particular care to handle. In this sense the nuclear shell model can be considered as the most computer intensive model in nuclear physics. This study extended significantly the calculations of ^{51}Ca, ^{52}Sc by Novoselsky et al. [2,3] who dealt with matrices of order about 50,000. Table 1 shows the range of dimensions encountered in modeling the Sb isotopes. Large-scale nuclear shell model calculations require enormous amount of CPU time and storage. The calculations of ^{110}Sb required the largest Hamiltonian dimension in this work. These are at the limit of the current Beowulf hardware at Drexel University, USA. There are two aspects of the calculations that prohibit us to go further: large CPU needs and tremendous storage capacity. There are many measurements of the larger Sb isotopes ^{111}Sb, ^{112}Sb, ^{113}Sb that beg for theoretical modeling; we are unable to do these calculations due to size. This article illustrates how we perform the calculations on a local parallel Beowulf system. In particular we address the need for parallel computing and the issue of the scalability of the Parallel DUSM code (DUPSM).

Table 1: Minimum and maximum dimensions encountered for the nuclear shell model calculations of light Sb isotopes.

	^{103}Sb	^{104}Sb	^{105}Sb	^{106}Sb	^{107}Sb	^{108}Sb	^{109}Sb	^{110}Sb
Min. Size	38	104	1175	2544	19227	31157	173770	213247
Max. Size	81	595	3655	18187	74731	258583	762253	1932826

2. ADAPTATION OF ALGORITHMS TO PARALLEL PLATFORM

An ideal program running on a parallel computer ought to be perfectly scalable, i.e., increasing the number of processors by a factor *N* should decrease the time spent to perform the task by a factor *1/N*. However, in practice, very few programs achieve this level of scalability. There are three major causes for this: existence of sequential parts in the algorithm, necessity of node to node communications, and load unbalance.

Adapting a sequential code to run on a parallel computer often requires some major changes in the algorithm in order to obtain good performance. A major guiding principle is to maximize the work done simultaneously on the different nodes. An important issue here is *data dependency*; for instance, if node 0 requires *x* after node 1 has calculated *x*, then it will have to wait until node 1 releases *x* before proceeding with *x*.

Even the best algorithms will often harbor some sections which are intrinsically sequential. This fact is expressed in Amdahl's law [4], which says that a program can only be sped up by the use of a parallel computer in the sections of the algorithm which is parallelized.

$$t = t_s + \frac{t_p}{N} \qquad (2.1)$$

where t_s is the time for the sequential part of the code, t_p is the time for the parallelizable part of the code, and N is the number of the processors. The parallel algorithms resulting in significant practical speed ups can be quantified by how well a code scales which is often measured as efficiency.

3. PARALLEL ASPECTS OF DUSM CODE

A parallel version of DUSM code, called Drexel University Parallel Shell Model (DUPSM) code, was recently written by using the message passing paradigm for distributed memory parallel machines. The code was implemented using PVM [5], a dynamic environment for parallel programming, supporting the message passing paradigm. Recall that the DUSM code involves two computational parts: building the Hamiltonian matrix and the Lanczos diagonalization procedure. Both computational parts, the construction of the Hamiltonian matrix and the Lanczos diagonalization, are fully parallelized by the domain decomposition in Hilbert space. The use of the permutational symmetry group introduces extra (unconserved) labels with which to label the basis; this splits the Hamiltonian matrix into independent submatrices. These submatrices are distributed to different processors, effectively giving a straightforward domain decomposition in Hilbert space. The Hamiltonian matrix is built up in blocks given by the permutation and the angular momenta labels on each slave machines. Then each block is stored on the local disk of each slave machine. Once the slave completes the calculation of the block it requests a new block to calculate. Each slave node continues to do so until all the independent submatrices of the Hamiltonian matrix are calculated and stored.

The Lanczos diagonalization procedure is also parallelized by using the domain decomposition method and master-slave algorithm. The Hamiltonian matrix is read piecewise from local disk of each node. The matrix multiplication is done piecewise on each node, then the resulting vector is gathered on the master node. The parallel input/output technique is extensively used whereby each node writes and reads on the local disk for increased I/O speeds. The Beowulf system uses a network switch to enhance communication speed and avoid communication collisions. Finally the master node calculates and diagonalizes Lanczos matrix.

The first computational phase of DUPSM code, calculation of the Hamiltonian matrix, is scalable and efficient. In this part all nodes use almost 99\% of the CPU and scales almost perfectly with the number of the processors. On the other hand, Lanczos diagonalization is scalable, but not efficient because of the limited disk input. That is why the parallel version of DUSM is good for the large calculation (the size of the Hamiltonian larger than about 2,000). It is better to use the serial version of DUSM code to avoid communication

collisions for small calculations – using the parallel version would require way too much communication for the actual amount of calculations to perform.

4. CONSRUCTION OF HAMILTONIAN MATRIX

The building of the Hamiltonian matrix uses the bulk of the CPU time in large calculations. In Figure 1 and 2 we plot the calculation time of the Hamiltonian matrix for different sizes versus the number of processors in the parallel system. In Figure 1 there are some fluctuations since any time disturbance from any one of the nodes is comparable to the calculation time. For the larger Hamiltonian sizes, these disturbances are negligible. This is reflected in the smoother curves in Figure 2.

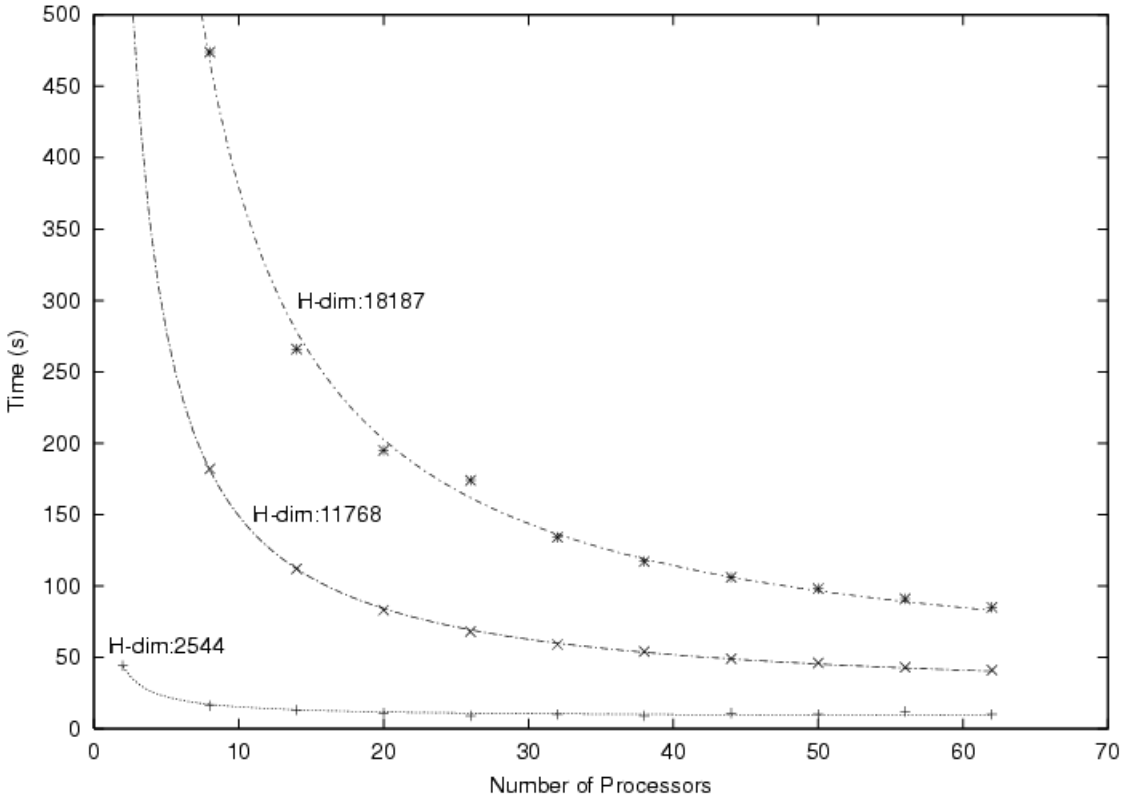

Figure 1: Time dependence of the construction of Hamiltonian matrix of small sizes to the number of the processors.

In both Figure 1 and 2, we can see that the curves almost converged to a constant for a larger number of processors (60 or more). This fixed amount of time in the calculations refers to the serial part of the DUPSM code mostly due to the overhead on the master node. It would require much effort to reduce this serial time.

Amdahl's law, Equ. (2.1), can fit these timings very well. The dashed lines in Figure 1 and 2 show these fits. The parameters t_s and t_p are given in Table 2.

Table 2: Fitted sequential and parallel timings.

Size	t_s	t_p
2544	8	71
11768	19	1296
18187	26	3533
31157	44	4911
52087	134	18315
74731	164	39596
146383	173	102995

The Hamiltonian matrix calculation scales almost perfectly with the number of the processors. Amdahl's law fits the timing data well. The scalar piece of the code is always relatively small. It scales roughly with the size.

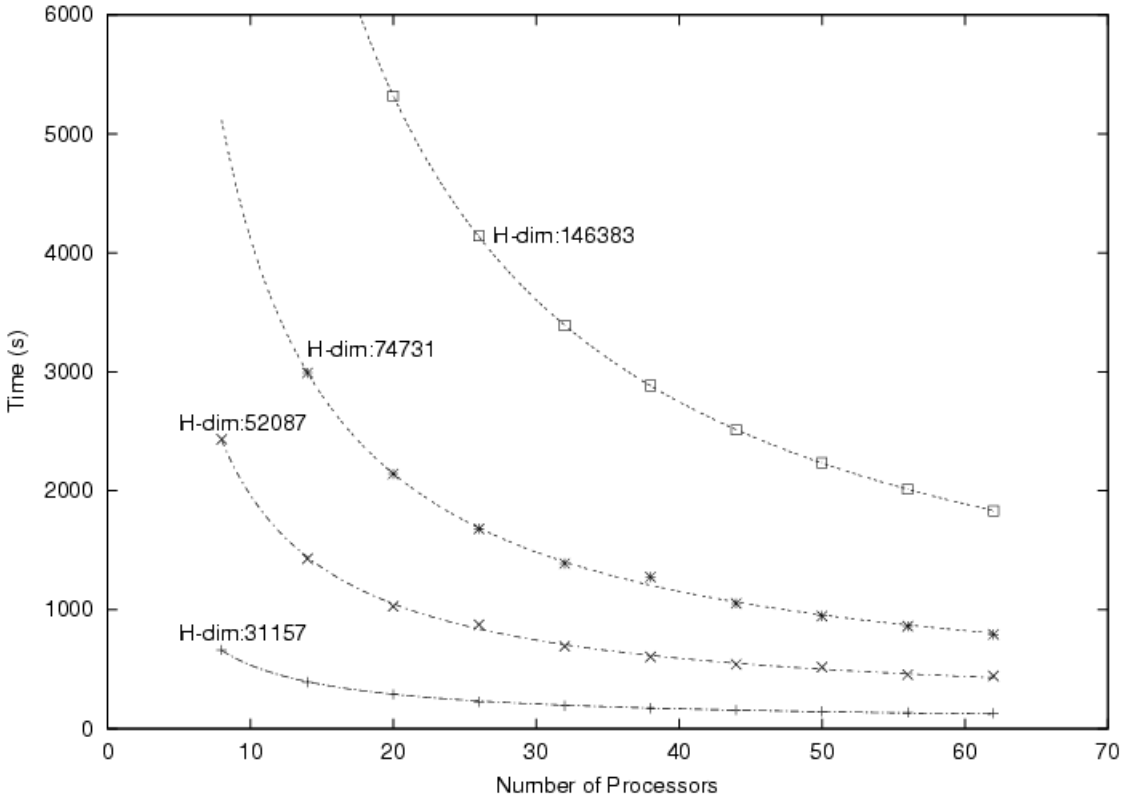

Figure 2: Time dependence of the construction of Hamiltonian matrix of relatively large sizes to the number of the processors.

The parallel piece of the code varies like $N^{1.8}$. This is the bulk of the time needed to compute the Hamiltonian matrix. This time should depend on N^2. However the occupation of the matrix diminishes mildly with size which contributes to a milder N dependence.

Figure 1 and 2 point to another aspect of the need for parallel computing, that of memory and storage. The largest Hamiltonian matrices doable on a single processor computer are of order of few thousands. The aggregate memory and disk space of a parallel computer allow much larger Hamiltonian sizes to be calculated and stored.

5. LANCZOS DIAGONALIZATION

The Lanczos diagonalization procedure in DUPSM code is also parallelized using the same domain decomposition method in Hilbert space. In this part the matrix elements of the Hamiltonian matrix and the left and right indices of the matrix elements are read from the local disk of each node. This requires extensive use of input and output (I/O) from disk and very little computing.

Once the product $H|\phi\rangle$ is performed on each node, the complete vector is gathered on node 0. This requires a significant amount of communication for large sizes. Then, node 0 extracts the new Lanczos vector, orthogonalizes it, normalizes it, and diagonalizes the Lanczos tri-diagonal matrix. This is done serially on node 0. Then node 0 broadcast this new vector to all the nodes.

Of course, this amount of communication and serial coding break the scalability of this part of the DUPSM code. It means that increasing number of the processors may not give the expected time reduction, but it still provides a solution for diagonalization of the Hamiltonian matrix in that it solves the storage problem.

In Figure 3 and 4 we plot the calculation times of the Lanczos diagonalization for different sizes of Hamiltonian matrix versus the number of processors. The calculation time scales with the number of processors up to about 28.

The system that we used to do the calculation, Cyborg Parallel Cluster at Drexel University, has double CPU on each node including node 0, but a single local disk. Therefore after 31 processes, the amount of I/O doubles on each subsequent channel. This results in a jump of the time for N=32 in Figure 3 and 4.

This feature which increases the I/O times prevents the scalability of the diagonalization part of the DUPSM code.

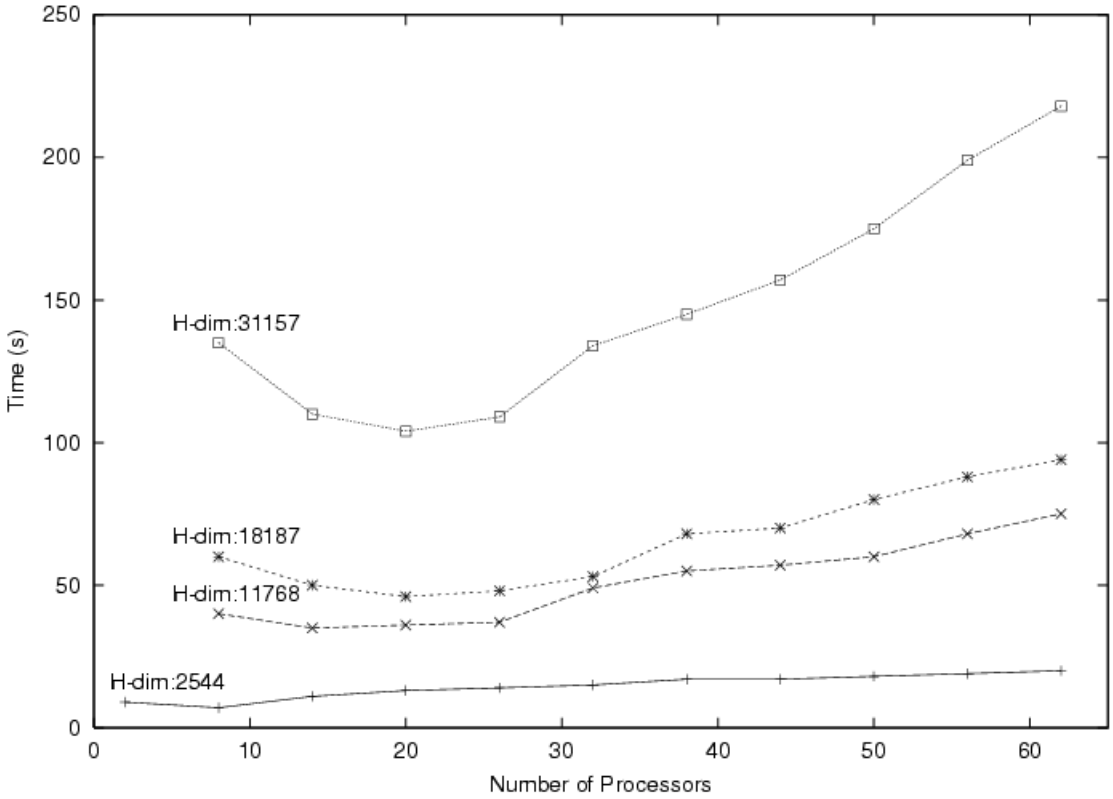

Figure 3: Time dependence of Lanczos diagonalization of the small size Hamiltonian matrix to the number of the processors.

In a parallel system, the increasing number of processors reduces the calculation time if the parallel code would scale perfectly well. In the case of the Lanczos diagonalization of the Hamiltonian matrix, the increasing number of processors also increases the communication in the internal network to gather and broadcast the Lanczos vectors. So, at some point the communication cost may be dominant against the time gain. This problem does not occur in the case of the construction of the Hamiltonian matrix since this part does not include extensive use of I/O and node-node communication.

Even if the Lanczos diagonalization part of DUPSM code does not scale perfectly, the calculation times are still reasonable and allow the calculations to be done. Note that the storage of the Hamiltonian matrices is only possible on the aggregate disk capacity of all the nodes.

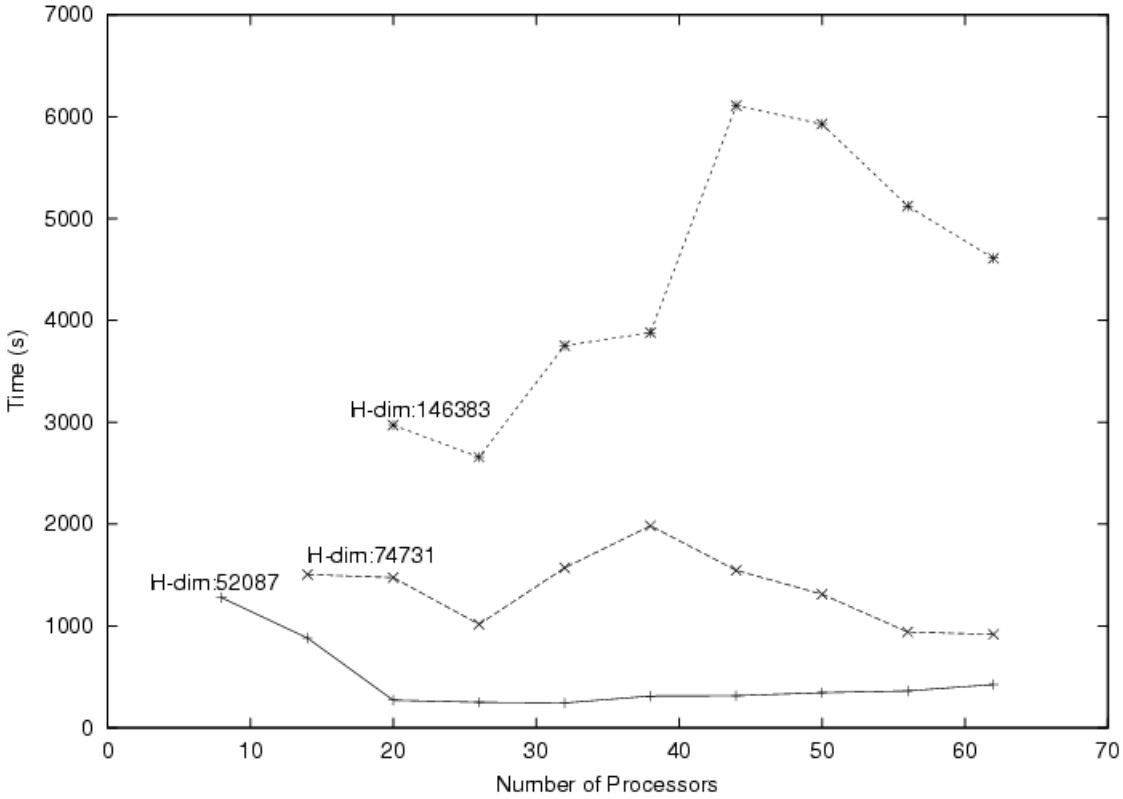

Figure 4: Time dependence of Lanczos diagonalization of the relatively large size Hamiltonian matrix to the number of the processors.

6. CONCLUSION

We have explored the capabilities of the DUPSM code by systematically performing large nuclear shell model calculations of the antimony isotopes from ^{103}Sb to ^{110}Sb. The code allowed us to handle matrices up to sizes 1.9M with no difficulty. The calculation of the Hamiltonian matrix is costly because of the large amount of computing time and the very large amount of disk capacity to store the Hamiltonian matrices. The first problem can be handled by using the latest fast CPU's and by increasing the number of nodes in the system. The second problem can also be handled by increasing the disk capacity of each slave machines. As of now our computer resources are somewhat limited in availability of temporary disk storage. Recomputing the Hamiltonian matrix was found less efficient than the parallel input from the disk.

REFERENCES

1. E. Dikmen, *Shell Model Studies in the sdgh Shell for the Proton Drip Line*, Ph.D. Thesis, Drexel University, PA, USA, 2002.

2. A. Novoselsky, M. Vallieres, O. La'adan, Full fp Shell Calculation of ^{51}Ca and ^{51}Sc, Phys. Rev. Lett. 79, 4341, 1997.

3. A. Novoselsky, M. Vallieres, Full fp Shell Calculation of ^{52}Sc, Phys. Rev. C 57, R19, 1998.

4. G. M. Amdahl, In AFIPS Conference Proceedings vol. 30, pp. 483-485, AIFPS Press, Reston, VA, 1967.

5. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM – Parallel Virtual Machine*, MIT Press, Cambridge, MA, 1994.