

Accelerating Full Configuration Interaction Calculations for Nuclear Structure*

Philip Sternberg[†] Pieter Maris[‡] Esmond Ng[†] Masha Sosonkina[§]
Hung Viet Le[§] James Vary[†] Chao Yang[†]

July 7, 2008

Abstract

One of the emerging computational approaches in nuclear physics is the full configuration interaction (FCI) method for solving the many-body nuclear Hamiltonian in a sufficiently large single-particle basis space to obtain exact answers - either directly or by extrapolation. The lowest eigenvalues and corresponding eigenvectors for very large, sparse and unstructured nuclear Hamiltonian matrices are obtained and used to evaluate additional experimental quantities. These matrices pose a significant challenge to the design and implementation of efficient and scalable algorithms for obtaining solutions on massively parallel computer systems. In this paper, we describe the computational strategies employed in a state-of-the-art FCI code MFDn (Many Fermion Dynamics - nuclear) as well as techniques we recently developed to enhance the computational efficiency of MFDn. We will demonstrate the current capability of MFDn and report the latest performance improvement we have achieved. We will also outline our future research directions.

1 Introduction

The direct solution of the quantum many-body problem transcends several areas of physics and chemistry. Nuclear physics faces the multiple hurdles of a very strong interaction, three-nucleon interactions, and complicated collective motion dynamics. We aim to solve for the structure of light nuclei addressing all three hurdles simultaneously by direct diagonalization of the nuclear many-body Hamiltonian in a harmonic oscillator single-particle basis.

The main tool we use to study nuclear structure is the software package MFDn (Many Fermion Dynamics for nuclear structure) developed by Vary and his collaborators at Iowa State University [7, 8]. In MFDn, the

*The computational results presented was obtained at the National Energy Research Scientific Computing Center (NERSC), which is supported by the Director, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under contract number DE-AC02-05CH11232. Research was supported in part by the UNEDF SciDAC Collaboration under DOE Grant DE-FC02-07ER41457.

[†]Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720.

[‡]Department of Physics, Iowa State University, Ames, IA 50011.

[§]Ames Laboratory, Iowa State University, Ames, IA 50011

nuclear Hamiltonian is evaluated in a large harmonic oscillator single-particle basis and diagonalized by iterative techniques to obtain the low-lying eigenvalues and eigenvectors. The eigenvectors are then used to evaluate a suite of experimental quantities to test accuracy and convergence issues. In several respects, the approach is similar to the Full Configuration Interaction (FCI) method in other fields. We often obtain convergence, either by direct diagonalization or simple extrapolation, and we then claim we have the result of a FCI calculation.

One key feature of a FCI calculation is the large dimension of the matrix Hamiltonian it can produce. The dimension of the matrix characterizes the size of the many-body basis used to represent a nuclear many-body Hamiltonian. In general, the larger the basis set, the higher the accuracy of the energy estimation and other computable quantities one can obtain. For example, in order to directly calculate the binding energy of ^{12}C to an accuracy of 1 MeV with a realistic nucleon-nucleon interaction, we estimate the size of the harmonic oscillator basis would be about 10^{12} .

Despite its large dimension, the Hamiltonian matrix produced in a FCI calculation is sparse, meaning the matrix contains a large number of zero entries. Therefore, the computational method used to solve the matrix eigenvalue problem must take advantage of the sparsity structure of the Hamiltonian. The large dimension and the irregular sparsity structure of the Hamiltonian matrix pose a significant challenge to the algorithmic design, data structure specification, and parallelization and memory management strategies in MFDn for large-scale distributed-memory computer systems. Furthermore, because the sparsity pattern of the matrix is not known in advance, we must determine it quickly during runtime. The naive approach of probing each matrix element exhaustively (to see if it is zero) is prohibitively expensive. Fortunately, the inherent combinatorial structure of the problem allows us to develop an efficient scheme that can identify large blocks of zeros without checking each individual element within these blocks.

The original design of the MFDn code took into account the standard parallel computing issues such as communication and load balancing. The code has run successfully on as many as 15,400 CPUs and has solved problems with more than 10^9 degrees of freedom. However, our recent investigation as part of the US Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) Program has identified a number of areas that can be further improved. In this paper, we will describe the general computational strategies employed in the design of MFDn as well as a number of techniques we recently developed to accelerate the computational speed of MFDn. We will demonstrate the current capability of MFDn and report the performance improvement we have achieved. We will also suggest a number of future research directions that will allow us to tackle even more challenging problems and advance the frontier of computational nuclear science.

2 Numerical method

The structure of an atomic nucleus with k nucleons is described by a many-body wavefunction $\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$, where $\mathbf{r}_j \in \mathbb{R}^3$, $j = 1, 2, \dots, k$. The wavefunction satisfies the many-body Schrödinger equation

$$H\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k) = \lambda\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k), \quad (1)$$

where H is a many-body Hamiltonian that relates a nucleus configuration defined by Ψ to the energy of the system. We denote the energy of the system by λ . The many-body Hamiltonian H is defined as

$$H = \frac{1}{k} \sum_{i < j} \frac{(\mathbf{p}_i - \mathbf{p}_j)^2}{2m} - \sum_{i < j=1}^k V_n(\mathbf{r}_i - \mathbf{r}_j), \quad (2)$$

where m is the nucleon mass, \mathbf{p}_i is a momentum operator, and $V_n(\mathbf{r}_i - \mathbf{r}_j)$ is a two-body potential operator that describes the interaction between the i th and j th nucleons. A more accurate treatment of the problem may include 3-body interacting potentials. Clearly, the wavefunction Ψ is an eigenfunction of H associated with the eigenvalue λ . It is normalized so that $\int_{\Omega} |\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)|^2 d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_k = 1$, where $\Omega = \Omega_1 \times \Omega_2 \dots \times \Omega_k$ and $\Omega_i \subseteq \mathbb{R}^3$. Furthermore, the integral $\int_{\Delta\Omega} |\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)|^2 d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_k$ represents the probability of finding nucleons 1, 2, \dots , k simultaneously in $\Delta\Omega \subseteq \Omega$.

In addition to the normalization constraint, $\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$ must also satisfy, among several conditions that we will describe later, the antisymmetry requirement: $\Psi(\mathbf{r}_1, \dots, \mathbf{r}_i, \dots, \mathbf{r}_j, \dots, \mathbf{r}_k) = -\Psi(\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_i, \dots, \mathbf{r}_k)$. When appropriate boundary conditions are defined for (1), the energy λ of the system becomes quantized and assumes a discrete set of values.

For nuclei that consist of a few nucleons (less than five), there are several methods to solve (1) directly. However, as k becomes larger, the size of the problem will become so large that approximate methods are necessary. One way to overcome the dimensionality explosion is to project the many-body Hamiltonian into a lower dimensional subspace \mathcal{S} that contains a set of basis functions $\{\Phi_i(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)\}_{i=1}^m$. Choosing an appropriate set of basis functions is the key to obtaining accurate approximations to the eigenvalues and eigenfunctions of (2). The basis functions used in MFDn are *Slater determinants* defined as

$$\Phi_a(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k) = \frac{1}{\sqrt{k!}} \begin{vmatrix} \phi_{a_1}(\mathbf{r}_1) & \phi_{a_2}(\mathbf{r}_1) & \dots & \phi_{a_k}(\mathbf{r}_1) \\ \phi_{a_1}(\mathbf{r}_2) & \phi_{a_2}(\mathbf{r}_2) & \dots & \phi_{a_k}(\mathbf{r}_2) \\ \vdots & \vdots & & \vdots \\ \phi_{a_1}(\mathbf{r}_k) & \phi_{a_2}(\mathbf{r}_k) & \dots & \phi_{a_k}(\mathbf{r}_k) \end{vmatrix}, \quad (3)$$

where ϕ_{a_i} is the eigenfunction associated with the a_i -th eigenvalue of a (single-particle) harmonic oscillator Hamil-

tonian $h = \mathbf{p}^2/2m + v_h(\mathbf{r})$, where $v_h(\mathbf{r})$ is quadratic with respect to \mathbf{r} . The use of Slater determinants is a standard technique used in quantum mechanics. In quantum chemistry, a sufficiently large subspace expansion using the Slater determinant basis is known as the full configuration interaction (FCI) calculation.

In this paper, we will define the index of $\Phi_a(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$ by a strictly increasing k -tuple of integers, i.e. $a = (a_1, a_2, \dots, a_k)$, where a_i is simply the index of the single-particle eigenfunction that appears in the i th column of the Slater determinant. We will refer to $\Phi_a(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$ or simply a as a *many-body basis state*. We will call each component of a a *single-particle state*. In a computer implementation, a can be represented by a binary string. The a_i -th elements of the string are set to one for $i = 1, \dots, k$, while all other elements are set to zeros.

If the index of the maximum allowed single-particle energy (eigenvalue counting degeneracy) is a_{max} , the total number of different $\Phi_a(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$ is $\binom{a_{max}}{k}$, which can be extremely large. However, a majority of these functions can be eliminated because they do not satisfy the additional constraints (based on traditional symmetries of H) regarding the magnetic projection, parity and total oscillator energy [9]. A many-body basis state satisfying all three of these conditions is a *valid* state. We will denote the set of all valid many-body basis states $\{a\}$ by \mathcal{A} . The size of \mathcal{A} will be denoted by $n = |\mathcal{A}|$. Suppose the desired many-body wavefunction can be well represented by a linear combination of the basis functions Φ_a ($a \in \mathcal{A}$), i.e.,

$$\Psi = \sum_{a \in \mathcal{A}} c_a \Phi_a, \quad (4)$$

where $c_a \in \mathbb{R}$, we can then solve (1) by computing eigenpairs of a projected Hamiltonian \hat{H} , where

$$\hat{H}_{a,b} = \int_{\Omega} (\Phi_a^* H \Phi_b) d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_k. \quad (5)$$

Because H is self-adjoint, \hat{H} is real symmetric. The eigenvector of \hat{H} associated with the desired eigenvalue (energy) gives the coefficients c_a in (4).

Clearly, the dimension of \hat{H} , which is the total number of valid many-body basis states n , depends on the total number of nucleons (k) contained in the nucleus of interest, on the largest single-particle state (a_{max}) allowed in $\Phi_a(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k)$ subject to the symmetry constraints, and on a chosen limit of total oscillator quanta (N_{max}) above the minimum for the nucleus. The value of a_{max} is fixed by first selecting N_{max} . For a large nucleus and large a_{max} value, n can be extremely large. For example, for an oxygen nucleus that consists of 8 protons and 8 neutrons, n is nearly 10^9 if $a_{max} = 572$ (corresponding to $N_{max} = 8$). However, the number of nonzero elements in \hat{H} is typically very small, as we will show below.

Note that the integral in (5) consists of both one-body integrals involving the first term of (2) and two-body integrals involving the second term of (2). It follows from the mutual orthogonality of all single-particle eigenfunctions

ϕ_ℓ ($\ell = 1, 2, \dots, a_{max}$) that a one-body integral in (5) becomes zero when a and b differ by more than one single-particle state, and a two-body integral becomes zero when a and b differ by more than two single-particle states. This observation allows us to determine many of the zero entries of \hat{H} without evaluating the numerical integral in (5).

Empirical evidence suggests that the probability of two randomly chosen but valid many-body basis states sharing more than $k - 2$ single particle states is relatively low. As a result, \hat{H} is extremely sparse. For a fixed N_{max} value, including three-body interactions in the potential term of (2) will produce more nonzero elements in \hat{H} . However, as N_{max} increases, the overall percentage of nonzero elements is still extremely low. Figure 1 shows both the growth of the matrix dimension (n) with respect to N_{max} and the growth of the number of nonzero elements in \hat{H} with respect to n for a variety of nuclei. In practice, we observe that the number of nonzeros in \hat{H} is proportional to $n^{3/2}$.

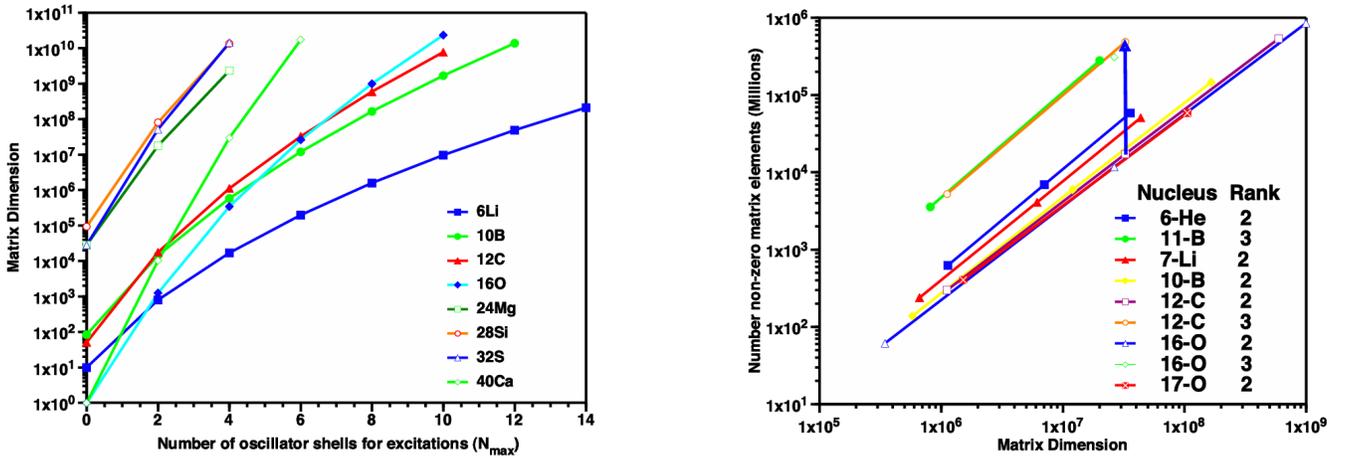


Figure 1: The plot on the left shows the growth of the matrix dimension (n) with respect to N_{max} for a variety of nuclei. The plot on the right shows the growth of number of nonzero matrix elements in \hat{H} with respect to n for a set of nuclei as a function of Hamiltonian matrix dimension. Rank 2 (3) signifies NN (NN+NNN) interactions are present.

We should point out that locations of the nonzero elements of \hat{H} do not follow a structured pattern one often sees in the finite difference or finite element discretization of differential operators. In fact, the locations of these nonzero elements are not known in advance. They must be determined efficiently at runtime. This is a major challenge of the nuclear FCI calculation that we will address in the next section.

A natural algorithm for computing a selected few eigenvalues and their corresponding eigenvectors of \hat{H} is an iterative method that does not require storing all $n \times n$ matrix elements. In nuclear physics, the eigenvalues of interest are those at the low end of the spectrum of \hat{H} because they describe the ground and the first few excited states of the nucleus. In MFDn, these eigenvalues are computed by the Lanczos method, which projects \hat{H} into a *Krylov* subspace $\mathcal{K}(\hat{H}, v_0) = \text{span}\{v_0, \hat{H}v_0, \dots, \hat{H}^{\ell-1}v_0\}$ of dimension $\ell \ll n$, where $v_0 \in \mathbb{R}^n$ is an arbitrarily chosen starting vector. If $V = (v_1, v_2, \dots, v_\ell)$ consists of an orthonormal basis of $\mathcal{K}(\hat{H}, v_0)$, the Lanczos method can be

described by

$$\hat{H}V = VT + fe_\ell^T, \tag{6}$$

where $T = V^T \hat{H}V$ is an $\ell \times \ell$ tridiagonal matrix that represents the projection of \hat{H} into $\mathcal{K}(\hat{H}, v_0)$, f is a residual vector that satisfies $V^T f = 0$, and e_ℓ is the ℓ -th column of the identity matrix. Approximations to eigenvalues of \hat{H} can be obtained by computing eigenvalues of the much smaller matrix T . If q is an eigenvector of T associated with the eigenvalue θ , then $z = Vq$ is the approximation to an eigenvector of \hat{H} .

It is well known that well separated extremal eigenvalues converge rapidly in the Lanczos iteration [5]. Convergence can be further improved by carefully choosing the starting vector v_0 and refining it using the implicitly restarted Lanczos algorithm developed in [6] and implemented in [4]. The major cost of the algorithm is the matrix-vector multiplication $w \leftarrow \hat{H}v$ required at each iteration.

In a standard application, one performs a sufficient number of Lanczos iterations to obtain the lowest fifteen states of the nucleus under investigation. Following that, the fifteen eigenvectors are employed to evaluate a suite of quantities to compare with experimental data. A standard suite includes electromagnetic moments and transition rates as well as weak interaction transition matrix elements.

3 Implementation details

In this section, we will briefly describe the distributed-memory parallel implementation of the nuclear FCI calculation developed in MFDn and present several recently developed techniques that have led to a significant performance improvement of the FCI calculation.

The recent modifications we made in MFDn were primarily aimed at improving the efficiency of the calculation, which is measured in terms of processing speed (i.e., elapsed time). We focused on the in-core version of the code in which the nonzero elements of \hat{H} are generated once in parallel and stored in the local memory of different processors. Although the use of this version is limited by the amount of memory available on each processor, it is significantly faster than the alternative approaches that would require either storing the matrix elements on disks and repeatedly retrieving them through I/O in subsequent calculations, or computing the matrix elements on the fly whenever they are needed.

Our discussion will focus on three main steps of the FCI computation:

1. The generation and distribution of the many-body basis states - This step essentially determines how the matrix Hamiltonian is partitioned and distributed in subsequent calculations.
2. The construction of the sparse matrix Hamiltonian \hat{H} - This step is performed simultaneously on all processors. Each processor will construct its portion of \hat{H} defined by the many-body basis states assigned to it. Because

the positions of the nonzero elements of the Hamiltonian is not known a priori, the key to achieving good performance during this step is to quickly identify the locations of these elements without evaluating them numerically first.

3. The calculation of the eigenvalues and eigenvectors using the Lanczos iteration - The major cost of the Lanczos iteration is the computation required to perform sparse matrix-vector multiplications of the form $y \leftarrow \hat{H}x$, where x, y are both vectors. Performing efficient orthogonalizations of the Lanczos basis vectors is also an important issue to consider.

The computational scheme we present here attempts to achieve scalable performance by maintaining a good load balance among different processors and minimizing interprocessor communications. In addition, it also contains several heuristics that reduce the number of integer and floating-point operations in the Hamiltonian calculation. These heuristics are developed by exploiting the unique combinatoric properties of the many-body configurations.

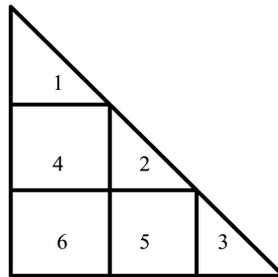


Figure 2: The projected Hamiltonian \hat{H} is partitioned and distributed among 6 processors.

3.1 Many-body basis state generation and distribution

Because \hat{H} is symmetric, we generate and store only the lower triangular part of the matrix to minimize memory usage. The lower triangular part of \hat{H} is partitioned into rectangular blocks and distributed among different processors. Figure 2 shows how submatrix blocks are mapped to different processors. Each block is labeled by a processor identification (pid) number that ranges from 1 to n_p , where n_p is the total number of processors in use. Due to the particular distribution pattern shown in Figure 2, the choice of n_p is not arbitrary. If we let n_d be the number of diagonal blocks in the partition, then $n_p = n_d(n_d + 1)/2$. In the following, we will refer to the processors to which the diagonal blocks of \hat{H} are assigned as the diagonal processors. They will be labeled by 1 through n_d . In MFDn, row and column communication groups are created to allow information to be passed among processors associated with row or column blocks of \hat{H} .

3.1.1 Load balance objectives

The size and nonzero structure of each sub-matrix block of \hat{H} are completely determined by how the many-body basis states are ordered and partitioned. If all valid many-body basis states are partitioned into n_d disjoint subsets S_1, S_2, \dots, S_{n_d} , then the nonzero structure of the (i, j) th sub-block in Figure 2 is determined by the many-body basis states contained in S_i and S_j .

To achieve a good load balance among different processors in terms of both memory usage and the number of floating-point operations performed in subsequent calculations, we would like to generate and partition the many-body basis states so that

- $|S_i|$ is roughly the same for all $1 \leq i \leq n_d$, and
- the number of many-body basis state pairs $(a, b) \in S_i \times S_j$ that will produce a nonzero $\hat{H}_{a,b}$ is roughly the same for all $1 \leq j \leq i \leq n_d$.

3.1.2 Enumeration of many-body basis states

Before we discuss how these two objectives can be achieved in an efficient manner, we first describe how many-body basis states are generated. The easiest way to generate all many-body basis states is to enumerate them in a lexicographical order defined below. Recall from Section 2 that a many-body basis state for a nucleus with k nucleons is a strictly increasing k -tuple of integers in the interval $[1, a_{max}]$, where a_{max} is the maximum single-particle state allowed. A many-body basis state $a = (a_1, a_2, \dots, a_k)$ is said to be *lexicographically less than* another many-body basis state $b = (b_1, b_2, \dots, b_k)$ if and only if there is a j for which $a_j < b_j$ and $a_i = b_i$ for all $i < j$. For example, if $a_{max} = 9$, then $(1, 3, 4, 8)$ is succeeded by $(1, 3, 4, 9)$, which is in turn succeeded by $(1, 3, 5, 6)$, and $(1, 3, 8, 9)$ is succeeded by $(1, 4, 5, 6)$.

We begin the enumeration process by first finding the lexicographically smallest many-body basis state that passes the validity test (which involves checking whether the magnetic projection, parity and total oscillator energy constraints are satisfied.) A new many-body basis state is created by incrementing the previously generated many-body basis state $a = (a_1, a_2, \dots, a_k)$. The incrementing process proceeds from the tail of a towards its head by raising one single-particle state a_j at a time subject to the constraints that a_j cannot exceed a_{max} and the ascending order among all single-particles states in the k -tuple must be preserved. To be specific, if a_{j+1} is larger than a_{max} or a_{j-1} , we will increase a_{j-1} instead. A new many-body state is checked for magnetic projection, parity, and total oscillator energy constraints. Only the states that pass these validity tests are kept.

Because protons and neutrons are treated separately in MFDn, a combined many-body basis state is represented by a disjoint set of integers associated with the proton and neutron single-particle states respectively. In other words, we can think of an MFDn many-body basis state as a pair of strictly increasing tuples with integer entries

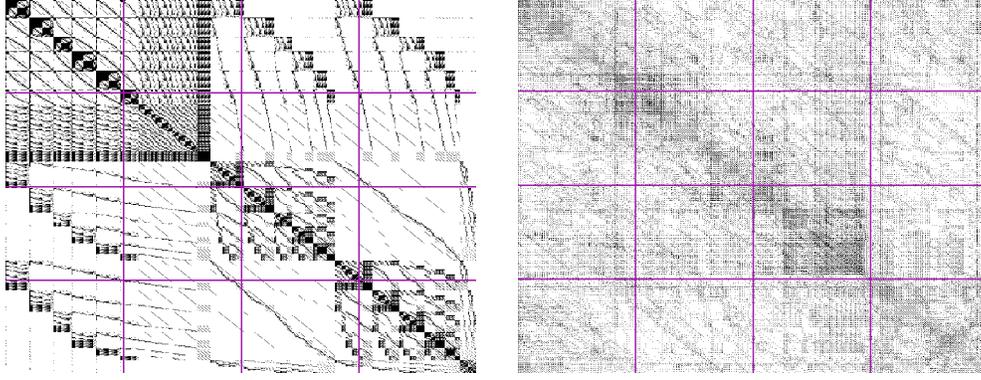


Figure 3: An uneven distribution of the nonzero elements of \hat{H} resulting from lexicographic order (left) and a more evenly distributed nonzero \hat{H} resulting from cyclic assignment of many-body basis states.

that belong to $[1, a_{max}]$. Incrementing a combined state can therefore be expressed by first attempting to increment the neutron many-body basis state, and if it cannot be incremented, incrementing the proton many-body basis state and resetting the neutron many-body basis state to its lowest possible values.

3.1.3 Cyclic distribution of many-body basis states

Let n be the total number of valid many-body basis states for a particular nucleus and choice of a_{max} . A naive way to distribute these many-body basis states is to divide the lexicographically ordered states evenly into n_d groups S_1, S_2, \dots, S_{n_d} so that all many-body basis states in S_i are lexicographically less than all many-body basis state in S_j for $i < j$. Although this approach satisfies the first load balance objective discussed in Section 3.1.1, it tends to violate the second objective, which requires each (distributed) submatrix block to have approximately the same number of nonzero elements.

A more favorable distribution scheme, which is implemented in MFDn, is to assign many-body basis states to S_1, S_2, \dots, S_{n_d} in a cyclic fashion so that two adjacent states in the lexicographically ordered sequence will not be assigned to the same S_i . To be more specific, the i th valid many-body basis state will be assigned to S_j for $j = \text{mod}(i - 1, n_d) + 1$. The reason why this distribution scheme tends to produce a more even distribution of the nonzero matrix elements lies in the fact that in the overwhelming majority of the cases, consecutive many-body basis states a and b will differ only in their last one or two single-particle states. As a result, if $\hat{H}_{c,a}$ is a nonzero element, $\hat{H}_{c,b}$ is also likely to be nonzero. Hence, by assigning a and b to different groups (processors), we are likely to achieve similar sparsity pattern on all processors. Figure 3 shows how cyclic distribution produces a more load balanced distribution of the nonzero elements of \hat{H} for a small test case.

3.1.4 Parallel generation of many-body basis states

In the previous implementation of MFDn, each many-body basis state is enumerated and validated by all the diagonal processors simultaneously. Only the processor that satisfies the cyclic mapping with respect to the valid many-body basis state just generated will keep the state and broadcast it to other processors that are in the same block row or column.

Although generating all many-body basis states in such a fashion ensures that

$$\max_{i,j} \left| |S_i| - |S_j| \right| \leq 1, \quad (7)$$

it is a sequential process because each diagonal processor must perform the same validity check on each many-body basis state, which typically takes a nontrivial amount of time.

To speed up the many-body basis state generation, we developed a new generation scheme that performs the validity check in parallel among all the diagonal processors. The main idea behind this scheme is to have each diagonal processor perform validation only on every n_d -th incrementally enumerated many-body basis state thereby reducing the number of validity checks performed on each diagonal processor by a factor of n_d .

To be specific, the i th diagonal processor starts the generation process by incrementing the smallest possible many-body basis state (which may not be valid) $i - 1$ times so that each diagonal processor will possess a different many-body state. After a validity check is performed, each valid state is kept on the processor on which it is generated and broadcast to the processors that are grouped in the same block row or column. An invalid state is simply skipped. Each processor will then increment the current many-body basis state n_d times before performing another validity check on the new state. Performing an n_d -fold increment is necessary because it is generally nontrivial to identify a many-body basis state that is n_d positions away from another many-body basis state in a lexicographically ordered sequence. Fortunately, incrementing a many-body basis state has low computational cost. This process continues until all possible many-body basis states have been generated.

Because not all many-body basis states enumerated on each of the diagonal processor are valid, there is generally no guarantee that (7) would hold. However, in practice, the relative standard deviation of the number of valid many-body basis states stored on different processors is typically around 0.5%. The resulting load imbalance is minimal. If in some cases better balance was required, the processors could redistribute the excess states with a negligible amount of communication. In fact, in some cases the load balancing of nonzeros is better under this distribution scheme than with the simpler cyclic assignment.

3.2 Hamiltonian construction

Once each processor receives two sets of many-body basis states S_i and S_j that can be viewed as the row and column indices of the matrix elements in the (i, j) th submatrix block of \hat{H} , it can begin to construct its portion of \hat{H} .

In MFDn, the construction of \hat{H} is performed in two steps. In the first step, the locations of the potentially nonzero elements of \hat{H} are identified. The numerical values of these elements are calculated in the second step. In the previous version of MFDn, a special compressed storage scheme that records the difference between row indices, modulo n , was used to represent the distributed sparse Hamiltonian on each processor. In the most recent version, we use the standard compressed column storage scheme [1] to store the matrix.

As we pointed out in Section 2, the mutual orthogonality of the single-particle eigenfunctions implies that the integral in (5) is zero when \hat{H} contains a K -body potential and when two many-body basis states a and b differ by more than K single-particle states.

In the following, we will call a pair of combined proton-neutron many-body states that differ by no more than K single-particle states an *interacting pair*. Checking whether a pair of many-body basis states is an interacting pair can be achieved by a bitwise operation performed on binary representations of these states. Even though performing such a bitwise operation is faster than evaluating (5) by numerical integration, the naive approach of performing bitwise operations on all pairs of many-body basis states in $S_i \times S_j$ is prohibitively expensive.

Because \hat{H} is extremely sparse, it contains many blocks of zeros. The efficiency of Hamiltonian construction can be greatly improved if we can identify a large block of zeros without performing bitwise operations for each matrix element within such a block. This type of strategy has been developed in MFDn by partitioning the list of single-particle states into g disjoint groups and creating blocks of many-body basis states by mapping each many-body basis state a to a g -tuple ($g < k$) that serves as the identifier for the block to which a is assigned. In the following we will illustrate how the g -tuple is defined given a partition P of single-particle states, and how a block of zeros can be quickly identified by comparing two g -tuples. Moreover, we will show that the partition of the single-particle states and the corresponding blocking of the many-body basis states can be performed recursively, and a hierarchical multi-level partitioning scheme can lead to further performance improvement of the Hamiltonian construction process.

Let \mathcal{A} be the set of all valid many-body basis states. Recall that a many-body basis state can be represented by a k -tuple $a = \{a_1, a_2, \dots, a_k\}$, where $1 \leq a_1 < a_2 < \dots < a_k$. Let P be a partition of a list of single-particle states within $[1, a_{max}]$, i.e., $P = \{p_1, p_2, \dots, p_g\}$, where the p_i 's are pairwise disjoint subsets of $[1, a_{max}]$ such that $\bigcup_{i=1, g} p_i = [1, a_{max}]$. We define a function $B_P : \mathcal{A} \rightarrow \mathbb{Z}^g$ by $B_P(a) = (u_1, u_2, \dots, u_g)$, where u_i is the number of elements of a that belong to p_i . For example, if $a = \{1, 3, 4, 7, 8, 9\}$ and

$$P = \{\{1, 2, 3\}, \{4\}, \{5\}, \{6, 7, 8, 9, 10\}\},$$

$B_P(a) = (2, 1, 0, 3)$. We call the g -tuple defined by $B_P(a)$ the P -configuration of a . Note that the maximum number of P -configurations associated with a set of many-body basis states represented by k -tuples is the number of solutions to the equation

$$\sum_{i=1}^g x_i = k, \quad (8)$$

where $x_i \in \mathbb{Z}$ and $0 \leq x_i \leq k$. In practice, the exact number of P -configurations, which is also the number of column or row blocks created for \hat{H} , is far less than the number of solutions to (8). However, that number is generally larger than g . Figure 4 illustrates how an artificially created small Hamiltonian is blocked by a partition of the single-particle states listed below the figure.

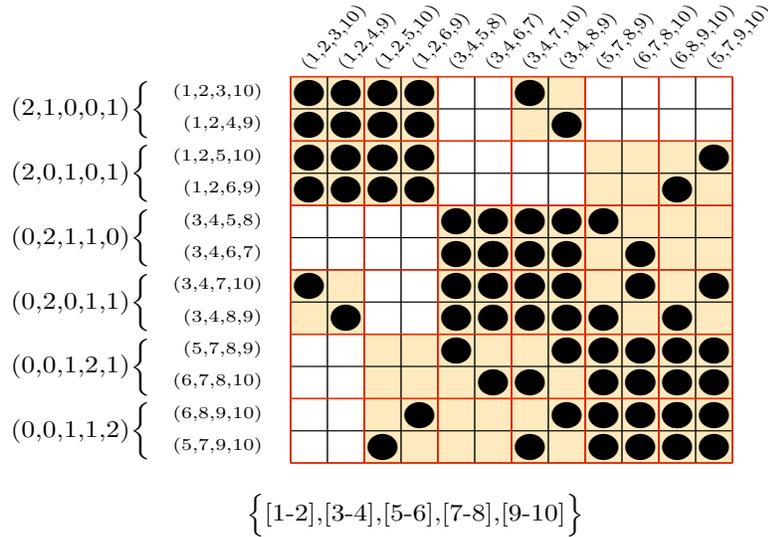


Figure 4: Twelve many-body basis states are divided into six groups of size two according to a partition of $[1, 10]$. The 2×2 blocks of zeros, as determined by this blocking scheme, are unshaded. Each interacting pair is designated by a black circle.

It is not difficult to see that if $\hat{H}_{a,b} \neq 0$, then $\|B_P(a) - B_P(b)\|_1 \leq 2K$, for any choice of P , where $\|B_P(a)\|_1$ denotes the 1-norm of the g -tuple. On the other hand, if

$$\|B_P(a) - B_P(b)\|_1 > 2K, \quad (9)$$

then $\hat{H}_{a,b}$ must be zero.

Once the P -configurations for all many-body basis states are obtained, we can use the condition (9) to identify a block of zeros by comparing two g -tuples associated with different P -configurations. If the 1-norm of the difference between these two g -tuples is larger than $2K$, the block indexed by these two P -configurations is a zero block. Hence no pairwise comparisons are needed between many-body basis states that are mapped to these two P -configurations.

When the 1-norm of the difference between two P -configurations is less than or equal to $2K$, the submatrix block

indexed by these two configurations may contain nonzero elements. However, this block may again be very sparse, and if possible, we would like to avoid making every pairwise comparison just to identify the location of a very small number of nonzeros. Therefore, we may apply the blocking scheme again within this block to identify smaller blocks of zeros.

A finer partitioning of the initial blocks created by the mapping function $B_P(\cdot)$ can be constructed by refining the initial partition of the single-particle states P . The refinement simply divides some of the p_i 's in P into disjoint subsets of single-particle states. For example,

$$P_1 = \{\{1, 2\}, \{3\}, \{4\}, \{5\}, \{6, 7, 8, 9, 10\}\},$$

gives a refinement of the initial single-particle partition $P_0 = \{\{1, 2, 3\}, \{4\}, \{5\}, \{6, 7, 8, 9, 10\}\}$. Using the refined partition P_1 , a many-body basis state $a = \{1, 3, 4, 7, 8, 9\}$ is mapped to a 5-tuple $B_{P_1}(a) = (1, 1, 1, 0, 3)$.

It is easy to see that the refinement of P_0 does not alter the coarser level block structure of the Hamiltonian determined by the mapping $B_{P_0}(\cdot)$. Hence, a multi-level blocking of the Hamiltonian that will allow us to quickly identify a large number zero blocks of different sizes can be achieved by judiciously choosing a multi-level partition of the single-particle states. Figure 5 gives a schematic illustration of what a three-level blocking of a Hamiltonian will look like. The shaded blocks represent the finest level blocks that contain nonzero matrix elements. In this particular case, a large block of zeros, the (2,2)-block bordered by solid lines, is identified at the coarsest level. Nine intermediate-sized zero blocks can be found at the second level. Sixteen small zero blocks can be seen at the finest level.

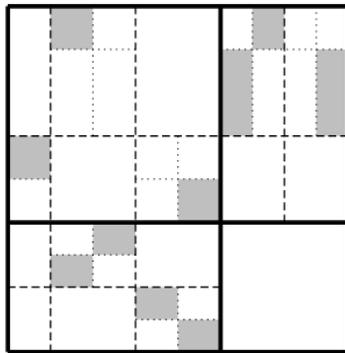


Figure 5: A three-level blocking of a portion of the Hamiltonian matrix \hat{H} distributed to an off-diagonal processor. The first (coarsest) level blocks are bordered by solid lines. The second level of blocks are bordered by thinner dashed lines. The finest level blocks are bordered by dotted lines, and those blocks containing nonzeros are shaded.

As we will show in Section 4.3, the use of a multi-level partition leads to significant performance improvement over a single-level partition, which was implemented in the previous version of MFDn. Our numerical experiments

also indicate that the amount of improvement can vary quite a bit among different partitions.

A practical question that remains to be addressed is how one should choose a partition of the single particle states at each level so that a significant number of nonzero matrix elements can be quickly identified. A related question is how many levels of partitions one should use. Ideally, at each level we want to check a relatively small number of blocks, and in so doing determine that a large percentage of blocks at the next finer level are zero. A partition that violates either of these properties will not give optimal performance. In practice, as the number of levels increases the percentage of blocks that can be excluded at each level will decrease due to the discrete nature of the problem. As illustrated by the timing results in Table 2, the benefit of adding partitions follows a trend of diminishing returns; in fact, every six-level partition we have examined results in greater execution time due to an increase in the number of block comparisons. As we tackle sparser matrices, the optimal number of partitions will increase, but there will always be some threshold beyond which no performance improvement can be realized.

Currently, the multi-level partition of the single-particle states is determined manually for each type of nucleus. Finding an optimal multi-level partition is a difficult problem that we will tackle in future studies.

3.3 The Lanczos iteration

MFDn uses the standard Lanczos algorithm to compute approximations to a few (typically 15 to 30) algebraically smallest eigenvalues of \hat{H} and the corresponding eigenvectors. The computational cost of the Lanczos iteration is dominated by the matrix-vector multiplications (MATVEC) $w \leftarrow \hat{H}v$ required to expand the Krylov subspace. To perform this operation efficiently on a lower triangular processor grid laid out in Figure 2, each input vector is partitioned among the diagonal processors. The sub-vector assigned to the i th diagonal processor is broadcast to other processors that belong to the i th column and row groups; see Figure 6. As a result, the (i, j) th off-diagonal processor will receive two sub-vectors v_i and v_j so that it can perform both $w_i \leftarrow \hat{H}_{i,j}v_j$ and $w_j \leftarrow \hat{H}_{i,j}^T v_i$, where $\hat{H}_{i,j}$ is the local piece of the Hamiltonian constructed on the (i, j) -th processor. Collective communications are performed among each row and column processor group to sum up contributions from all local computations. Due to the cyclic distribution of the many-body basis states, all local pieces of the Hamiltonian \hat{H} have approximately the same number of nonzeros. The even distribution of the nonzero elements among all processors allows a good load balance to be achieved for each MATVEC.

To eliminate the presence of spurious eigenvalues [5], MFDn performs full orthogonalization among the columns of V in (6). As the number of columns in V increases, orthogonalization can become a computational bottleneck if it is not effectively parallelized. To perform the orthogonalization process in parallel on all processors, MFDn remaps the triangular processor grid to an $n_d \times \lfloor (n_d + 1)/2 \rfloor$ rectangular grid, with new communicators defined for both the rows and columns of this grid. Due to space limitation, the details of the mapping will be described in a future publication. As an example, Figure 7 shows how a 15-processor triangular grid is mapped to a 5×3 rectangular processor grid.

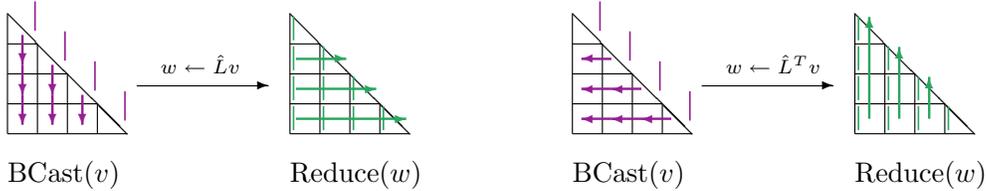


Figure 6: The communication pattern for multiplying the lower triangular (left) and upper triangular (right) parts of \hat{H} with the distributed input vector v . The arrows indicate how v is broadcast among different processor groups and how global sums are performed among different processor groups.

Columns of V are distributed in a cyclic fashion among the column groups of the rectangular grid. At the j th Lanczos iteration (for $j > n_d$), the orthogonalization process is carried out by performing inner product calculations between $\hat{H}v_j$ and all (distributed) columns of V in parallel on all processors. Global sums are performed within column groups to obtain $h = V^T(\hat{H}v_j)$, with the components of h distributed among the row groups. An additional global vector sum along the row groups is required to obtain $f = \hat{H}v_j - Vh$. The computational results we will

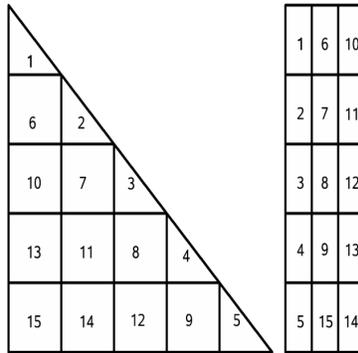


Figure 7: The mapping between a lower triangular 15-processor grid (left) to a 5×3 rectangular processor grid(right). The numbers in the figure represent processor IDs.

report in the next section show that this type of parallelization allows the Lanczos iteration to scale almost linearly to thousands of processors. However, such a scheme does not allow us to easily utilize level-2 BLAS operations as is done in many eigenvalue packages, such as PARPACK (a parallel version of ARPACK [3, 4]). Because PARPACK assumes a one-dimensional partition of the processors, orthogonalization can only be performed on the diagonal processors. When a small number of processors are used, a version of MFDn that uses PARPACK to compute the desired eigenvalues outperforms the existing MFDn calculation. However, when a large number of processors are used to solve large problems, performing orthogonalization using only the diagonal processors becomes a bottleneck.

4 Computational Performance

In this section, we report the performance of MFDn before and after the recent algorithmic and implementation improvements were made. All computations reported here were carried out on the Franklin cluster at NERSC. Franklin is a distributed-memory parallel system with 9,660 compute nodes. Each compute node consists of a 2.6 GHz dual-core AMD Opteron processor with a theoretical peak performance of 5.2 GFlop/sec. Each compute node has 4 GBytes of memory. We will first report the overall performance improvement achieved in the two versions of MFDn, and then take a closer look at the impact of the multi-level partition of the Hamiltonian on the performance improvement and the scalability of the computation.

4.1 Overall performance

Table 1 lists wallclock times used by MFDn to obtain the lowest 15 energy states associated with ${}^6\text{He}$, ${}^{12}\text{C}$, ${}^{13}\text{C}$ and ${}^{16}\text{O}$. Two-body potentials (denoted by the symbol “NN” in the table) and a combination of two- and three-body potentials (denoted by “NN+NNN”) were used in these calculations. In some of the calculations, we used a smaller many-body basis set with N_{max} set to 4. In others, we used a larger basis set with N_{max} set to 14. The column labeled by “old” refers to the timing collected for the pre-SciDAC version of the MFDn runs. The column labeled by “new” refers to the timing collected for the new version of the MFDn code that includes both the improved parallel version of the many-body basis generation procedure and a multi-level partitioning scheme for Hamiltonian construction. These calculations were performed using various numbers of processors. The fifteen lowest eigenpairs were computed. The total number of Lanczos iterations used was 400 for ${}^6\text{He}$ with $N_{max} = 14$ and ${}^{16}\text{O}$ with $N_{max} = 8$. For each of the three cases of C, the number of Lanczos iterations used was 500. We can clearly see from this table that the

Nucleus	N_{max}	potential type	matrix dim	CPU count	old	new
${}^6\text{He}$	14	NN	155,710,094	4,950	6,499	2,456
${}^{12}\text{C}$	4	NN	1,118,926	28	247	159
${}^{13}\text{C}$	4	NN+NNN	1,065,847	45	1,831	915
${}^{13}\text{C}$	6	NN+NNN	28,260,781	4,950	5,109	2,586
${}^{16}\text{O}$	8	NN	996,878,170	12,090	> 28,000	6,664

Table 1: MFDn timing (in seconds) for a range of realistic test problems.

recent changes we made in MFDn resulted in at least 50% improvement for the FCI calculations of all test cases. Generally, we see greater performance improvement when the Hamiltonian becomes sparser as the result of either an increased number of nucleons or an increase in N_{max} . Similar performance improvement was observed on other high performance computing platforms such as the Jaguar cluster at ORNL and the IBM Power 5 (Bassi) system at NERSC.

4.2 Performance gain within the components of MFDn

Figure 8 gives a more detailed picture of the relative contribution from the various components of the MFDn code to the overall performance improvement of the code on the ^{13}C calculation with $N_{max} = 6$ using three-body interactions. The three areas in which we see improvement are setting up the basis, evaluating the Hamiltonian, and the Lanczos process to compute the eigenstates of the Hamiltonian. The improvement in the basis setup time results from the

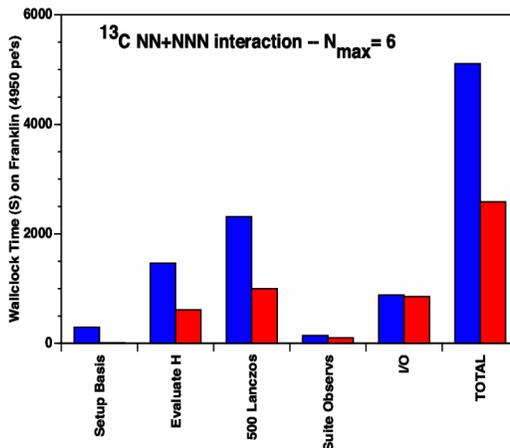


Figure 8: Performance improvement in sections of MFDn.

parallelization technique we discussed in Section 3.1.4. Previously the execution time of this part of the code was independent of the number of processors used; the new parallel algorithm reduces the required time by approximately a factor of 6.5.

By using a multi-level partitioning scheme as described in Section 3.2, the time to generate the Hamiltonian (“Evaluate H” in Figure 8) is reduced dramatically. The potential improvement in execution time depends primarily on the sparseness of the Hamiltonian, with greater gains possible for sparser matrices. Two of the directions in which future scientific discoveries lie are larger nuclei and larger model spaces (i.e., larger values of N_{max}); both of these will lead to progressively sparser Hamiltonians. The other parameter affecting the sparsity of the Hamiltonian is whether we are using two- or three-body interactions. The three-body Hamiltonian is denser than the two-body Hamiltonian by approximately a factor of 30. While this does impact the speedup we can achieve with these improvements, the increase in sparsity due to the other two factors is much more significant. We discuss a more detailed performance analysis of this feature in Section 4.3.

The performance of the Lanczos iteration (“500 Lanczos” in Figure 8) is improved by at least 50%. Much of this improvement can be attributed to using compressed column storage for the Hamiltonian rather than a custom sparse matrix storage scheme. The additional performance gain is the effect of reordering the rows and columns of the Hamiltonian according to a multi-level partition. This permutation of the rows and columns of the Hamiltonian results in more of the nonzeros being stored closer together on each processor, so the sequence of accesses to the input

and output vectors are more respectful of data locality. The reduction in cache misses leads to better performance.

4.3 The impact of multi-level partition on the efficiency of Hamiltonian construction

In Table 2, we show how the performance of the Hamiltonian construction is affected by using successively refined multi-level partitions in our recursive blocking scheme for ^{16}O , $N_{max} = 8$, with two-body interactions. The numbers reported in the table reflect the work done on a representative off-diagonal processor in a 12,090 processor run, with each row in the table corresponding to a multi-level partition with the specified number of levels. For each successive row in the table, a finer partition is added, further reducing the number of individual pairs of matrix elements that need to be tested to identify nonzeros of the Hamiltonian. The number of these bitwise comparisons performed among many-body basis states contained in the lowest-level nonzero blocks are recorded in column 3, and the total numbers of block (g -tuple) comparisons performed at 1 through ℓ levels are recorded in column 4. We can clearly see that as we increase the number of partition levels, the number of matrix element comparisons decreases dramatically. At the same time the number of block comparisons, which indicates the overhead required to achieve such a small number of matrix element comparisons, increases. There are over 4.1×10^{13} matrix elements on this processor, so even using only the two coarsest levels of blocking, the amount of work done is reduced by an order of magnitude.

Number of levels	seconds	matrix element comparisons	block comparisons
2	29,996	1.9×10^{12}	1.7×10^8
3	4,630	3.0×10^{11}	5.6×10^8
4	1,483	7.6×10^{10}	2.1×10^9
5	1,251	3.0×10^{10}	5.5×10^9

Table 2: Performance statistics for successively refined multi-level partitions for ^{16}O , $N_{max} = 8$.

4.4 Parallel scalability of the new MFDn

Figure 9 shows how the overall performance as well as each component of the new version of MFDn scales with respect to the number of processors used for ^{13}C with N_{max} set to 6. We define the speedup factor as the ratio between $T_{n_{cpu}}$ and T_{base} , where T_{base} is the wall clock time required to complete the entire MFDn run or one particular component of MFDn on 2,850 processors and $T_{n_{cpu}}$ is the wall clock time required to complete the computation on n_{cpu} processors. We observe that total MFDn timing scales superlinearly with respect to the number of processors when fewer than 4,950 processors are used. The superlinear scaling is primarily due to the significant improvements in the Hamiltonian construction (less recomputation of intermediate quantities) and many-body basis state setup phases of the calculation. It is also clear from the figure that the Lanczos iteration has a perfect linear speedup. This is due to a good load balance achieved by cyclic distribution of the many-body basis states, and effective orthogonalization. We should comment that this part of the calculation, which is dominated by floating-point operations, typically runs

at 20% of the peak performance. However, the overall flops count of MFDn is much lower due to the large number of integer operations performed in the Hamiltonian construction and a significant amount of I/O operations. We observe that the I/O operations, which does not yet scale with respect to the number of processors, represent a small fraction of the total computational cost when the number of processors is relatively small (below 5000). As the number of processors increases, the sequential bottleneck of I/O operations becomes more pronounced. It is ultimately responsible for suboptimal overall performance of MFDn for $n_{cpu} > 5000$.

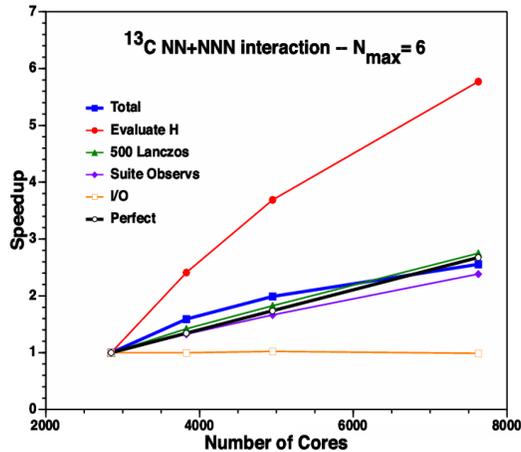


Figure 9: Scalability of MFDn.

5 Concluding remarks

We have described several techniques that help accelerate the computational speed of MFDn for nuclear structure calculations on massively parallel computer systems consisting of thousands or tens of thousands of processors. In particular, we demonstrated how many-body basis states can be generated and distributed in parallel. This new generation scheme not only reduces the setup time for FCI calculations significantly but also ensures that a good load balance is maintained during the Hamiltonian construction and eigenvalue calculation phases of the computation. We also showed that the amount of work required to determine the nonzero structure of the Hamiltonian can be reduced significantly by using a multi-level blocking scheme that can quickly identify a block of zeros by a single comparison between a pair of block identifiers. This multi-level blocking scheme also appears to improve the data locality of the distributed matrix Hamiltonian and results in performance improvement of the matrix-vector multiplications used in the Lanczos iteration.

The choice of a multi-level single-particle state partition, which ultimately determines the block structure of the Hamiltonian, is currently determined manually prior to each calculation based on intuition and experience. We will investigate strategies for choosing an optimal partition of the single-particle states during runtime in the future.

The cyclic distribution of many-body basis states, which we currently use in MFDn, allows the nonzero elements of the Hamiltonian to be distributed nearly uniformly among different processors. However, this distribution scheme has the tendency of destroying dense blocks of nonzeros. We plan to explore the possibility of using block cyclic distribution to further improve the memory locality of the Hamiltonian.

In addition to seeking implementation strategies that will take advantage of the memory hierarchy and other features of high performance computers, we will also explore alternative algorithms for eigenvalue computation. In particular, we will investigate the possibility of using the locally optimal preconditioned conjugate gradient (LOBPCG) algorithm [2], which computes the desired eigenvalues and eigenvectors by solving a constrained minimization problem. The key to the potential success of using this method lies in identifying an effective preconditioner that would limit the total number of LOPCG iterations to 20 or less, and this is part of our future investigation.

References

- [1] Iain S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report RAL 92-086, CERFACS, Chilton, Oxon, England, 1992.
- [2] A. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 22(2):517–541, 2001.
- [3] R. B. Lehoucq, D. C. Sorensen, P. Vu, and C. Yang. *ARPACK: An implementation of the Implicitly Re-started Arnoldi Iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix*, 1995. Available from ftp.caam.rice.edu under the directory pub/software/ARPACK.
- [4] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA, 1998.
- [5] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, 1980.
- [6] D. C. Sorensen. Implicit application of polynomial filters in a k -step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, 13(1):357–385, January 1992.
- [7] J.P. Vary. The many-fermion dynamics shell-model code, 1992. unpublished.
- [8] J.P. Vary and D.C. Zheng. The many-fermion dynamics shell-model code, *ibid.*, 1994. unpublished.
- [9] S. Wong. *Introductory Nuclear Physics*. Prentice-Hall, Englewood Cliffs, NJ., 1990.