COMPUTATIONAL CHALLENGES FOR 3-BODY FORCES IN THE SHELL MODEL

Hai Ah Nam
San Diego State University
Computational Science, PhD Student

Collaboration with: Calvin Johnson, SDSU
W. Erich Ormand, LLNL

This work performed under the auspices of the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.
The Shell Model

- Typical eigenvalue problem

\[ H \Psi_i = E_i \Psi_i \]

- Construct many-body basis states \( |\phi_i\rangle \) so that

\[ \Psi_i = \sum_n C_{in} \phi_n \]

- Calculate Hamiltonian matrix elements

\[ H_{ij} = \langle \phi_j | H | \phi_i \rangle \]
- Diagonalize to obtain eigenvalues & eigenvectors

\[ H = \sum_i \varepsilon_i a_i^+ a_i + \frac{1}{4} \sum_{ijkl} V_{ijkl} a_i^+ a_j^+ a_k a_l \]

\[ H = \sum_i \varepsilon_i a_i^+ a_i + \sum_{ijklmn} V_{ijklmn} a_i^+ a_j^+ a_k^+ a_l a_m a_n \]

\[ \phi = \frac{1}{\sqrt{A!}} \begin{pmatrix} \phi_1(r_1) & \phi_2(r_1) & \cdots & \phi_A(r_1) \\ \phi_1(r_2) & \phi_2(r_2) & \cdots & \phi_A(r_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(r_A) & \phi_2(r_A) & \cdots & \phi_A(r_A) \end{pmatrix} \]

\[ = a_1^+ \ldots a_j^+ a_i^+ |0\rangle \]
Shell Model Codes

- **Oak Ridge (1969)**
  - Coefficients of Fractional parentage

- **Glasgow (1977)**
  - Good Jz (M-scheme)
  - J restored in diagonalization

- **OXBASH (1985)**
  - J-projected M-scheme
  - Smaller matrices

- **RITSSCHIL (1985)**
  - CFP

- **DUSM (1989)**
  - Permutation groups

- **ANTOINE (1991 & 1999)**
  - M-scheme
  - Apply matrix on-the-fly
  - Large dimensions

- **NATHAN (1998)**
  - J-projected similar to ANTOINE
  - "Hybrid" M-scheme-CFP code

- **REDSICK (now)**
  - Based on ANTOINE papers
  - M-scheme
  - Three-body interactions

- **CMICHSM & MFD (now)**
Effects of 3 Body Interactions

- Increase total binding energy
- Increase spin-orbit splitting
  - Improve low-lying excitation spectra
    (correct ground state spin)
- Spin-observables
  - Magnetic moment
  - Gamow-Teller transition strengths

- BETTER AGREEMENT WITH EXPERIMENT!!

| $^6$Li basis space | $|E_{gs}|(1^+0)$ | Exp | AV8$'$+TM$'$ (99) | AV8$'$ |
|---------------------|----------------|-----|------------------|--------|
|                     | 31.995         | 31.036 | 28.406           |

(2003)
Investigations of
$^6,7$Li, $^6$He,
$^7,8,10$Be,
$^{10,11,12}$B, $^{12}$N,
$^{10,11,12,13}$C

Source: Navratil and Ormand, Phys. Rev. C 68, 034305

<table>
<thead>
<tr>
<th>$^{11}$B$\rightarrow^{11}$C basis space</th>
<th>Exp</th>
<th>AV8$'$+TM$'$ (99)</th>
<th>AV8$'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(GT; $\frac{3}{2}^-$ $\rightarrow$ $\frac{1}{2}^-$)</td>
<td>0.345</td>
<td>0.315</td>
<td>0.765</td>
</tr>
<tr>
<td>B(GT; $\frac{3}{2}^-$ $\rightarrow$ $\frac{3}{2}^-$)</td>
<td>0.309</td>
<td>0.591</td>
<td>0.909</td>
</tr>
<tr>
<td>B(GT; $\frac{3}{2}^-$ $\rightarrow$ $\frac{5}{2}^-$)</td>
<td>0.961$^a$</td>
<td>0.517</td>
<td>0.353</td>
</tr>
<tr>
<td>B(GT; $\frac{3}{2}^-$ $\rightarrow$ $\frac{7}{2}^-$)</td>
<td>0.961$^a$</td>
<td>0.741</td>
<td>0.531</td>
</tr>
<tr>
<td>B(GT; $\frac{3}{2}^-$ $\rightarrow$ $\frac{3}{2}^-$)</td>
<td>0.444$^b$</td>
<td>0.625</td>
<td>0.197</td>
</tr>
</tbody>
</table>
Current Investigations

- Higher p-shell, low sd-shell nuclei
  - $^9\text{Be}$, $^{15}\text{O}$, $^{16}\text{O}$, $^{17}\text{O}$
- 2-body results

The three nucleon interaction plays a critical role in determining the structure of nuclei BUT is computationally challenging

H. Nam

September 26, 2007: Institute for Nuclear Theory
Lanczos (iterative method)

\[
\begin{align*}
\hat{H}v_1 &= \alpha_1 v_1 + \beta_1 v_2 \\
\hat{H}v_2 &= \beta_1 v_1 + \alpha_2 v_2 + \beta_2 v_3 \\
\hat{H}v_3 &= \beta_2 v_2 + \alpha_3 v_3 + \beta_3 v_4 \\
\hat{H}v_4 &= \beta_3 v_3 + \alpha_4 v_4 + \beta_4 v_5
\end{align*}
\]

- Ideal for solving a large sparse matrix
- \( \sim 100-200 \) iterations for the lowest 10 eigenvalues
- Matrix-vector multiplication, vector dot products for \( \alpha \)'s, \( \beta \)'s
- Memory and run-time bottleneck
Why is it so difficult?

- Large dimensions of the Hamiltonian matrix
  - grows dramatically with # of particles & valence space
  \[ \text{Dim} = \left( \begin{array}{c} N^p_{\text{sps}} \\ n^p \\ N^n_{\text{sps}} \\ n^n \end{array} \right) \]
e.g. $^{60}\text{Zn}$ (fp – shell)
  \[ \binom{20}{10} \binom{20}{10} = 3.4 \times 10^{10} \]

- Large SPARSE matrix
  - Only store non-zero matrix elements

<table>
<thead>
<tr>
<th>Nuclide</th>
<th>space</th>
<th>$N_{\text{val}}$</th>
<th>$Z_{\text{val}}$</th>
<th>Dim basis</th>
<th>Sparsity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^{20}\text{Ne}$</td>
<td>sd</td>
<td>2</td>
<td>2</td>
<td>640</td>
<td>13.0</td>
</tr>
<tr>
<td>$^{24}\text{Mg}$</td>
<td>sd</td>
<td>4</td>
<td>4</td>
<td>28,503</td>
<td>0.74</td>
</tr>
<tr>
<td>$^{28}\text{Si}$</td>
<td>sd</td>
<td>6</td>
<td>6</td>
<td>93,710</td>
<td>0.34</td>
</tr>
<tr>
<td>$^{46}\text{V}$</td>
<td>pf</td>
<td>3</td>
<td>3</td>
<td>121,440</td>
<td>0.36</td>
</tr>
<tr>
<td>$^{48}\text{Cr}$</td>
<td>pf</td>
<td>4</td>
<td>4</td>
<td>1,963,461</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Example

- Dim = $10^8$, Sparsity = 0.005%
  - # of m.e. = $10^8 \times 10^8 \times 0.005% = 5 \times 10^{11}$ matrix elements
  - Single precision - real(4)
    - = 2 TB to store (2,000 GB)
  - Dim = $10^9$
    - = 200 TB (200,000 GB)
Shared & Distributed Computing Environment

Memory (RAM) ~ 2 - 4 GB/proc
- **Thunder** - Linux: [1024 nodes / 4096 CPUs] 4 CPUs/node w/ 8 GB shared memory (8192 GB total)
- **uP** - IBM: [108 nodes / 864 CPUs] 8 CPUs/node w/ 32 GB shared memory (3456 GB total)

Run-time ~ 12 hours
(DAT time ~ 48 hours)

Parallel File System (Disk)
- Thunder: 338 TB
- uP: 130 TB
- Not all available to 1 user (small fraction)
More difficulties with 3-Body Forces

- Basis dimensions are the same (2 \rightarrow 3 body) but H is less sparse
  - \( H_{ij} = H_{pp} + H_{nn} + H_{pn} \) (2 body)
  - \( H_{ijk} = H_{ppp} + H_{nnn} + H_{ppn} + H_{nnp} \) (3 body)
  - More non-zero matrix elements
  - MORE MEMORY INTENSIVE

- Increase in run-time
  - ^10\text{B}, 4 \hbar \Omega; \quad \text{Basis Dim} = 581,740
    - 2-body has \( \sim 145 \times 10^6 \) non-zero elements
      - \( \sim 1\)- 2 CPU-hr for lowest ten states
    - 3-body has \( \sim 2.2 \times 10^9 \) non-zero elements
      - > 200 CPU-hr

- As if it wasn’t already challenging!!

"One of us is in serious trouble!"
Sparsity Increases with 3 Body Forces

\[ ^{20}\text{Ne} \]
\[ \text{sd-shell} \]
\[ 640 \text{ basis states} \]

\[ ^{6}\text{Li} \]
\[ 2\text{hw} \]
\[ 800 \text{ basis states} \]
Memory Limitation Solutions

- **Store matrix elements on disk**
  - Requires much disk space, fast i/o.
  - Disk access is ~1000 times slower than RAM access
  - (e.g., OXBASH, Glasgow-Los Alamos, CMICHSM)

- **Store matrix elements in RAM**
  - Limited by # of nodes available.
  - 3,000 processor @ 2GB = 6,000 GB RAM
  - (e.g. MFD)

- **On-the-fly: Recompute the many-body matrix elements**
  - Re-compute on each iteration from the two- (and three-) body matrix elements
  - **Efficient if you only compute non-zero matrix elements – NEED TO KNOW WHICH ARE NON-ZERO!!!**
  - (e.g. ANTOINE, REDSTICK)

---

Dim = $10^8$, Sparsity = 0.05%  
= 2 TB (2,000 GB)

Dim = $10^9$  
= 200 TB (200,000 GB)
Shell Model Code

- On-the-fly construction of Hamiltonian matrix
- Created by W. Erich Ormand & Calvin W. Johnson
- Fortran 90 & MPI
- 2-body version
  - 65+ subroutines
  - 16,000+ lines of code
- 3-body version
- Release spring 2008

Needs Improvements

- Run-time
- Current structure will have memory issues outside of lanczos bottleneck

To reach frontier computations we need optimized code (serial and parallel parts)

Dim > $10^8$

Algorithms Load Balancing
REDSTICK Development

- **2004 – 2005**
  - Huh? What’s REDSTICK?

- **2006 – 2007**
  - New “Jump” algorithm applied to various subroutines in 2 body version
  - Improves run-time performance

- **Summer 2007**
  - New 3-body code with similar jump type algorithm

- **In Progress**
  - Analysis of parallelization schemes
  - Implementation of parallelization schemes to 3 body code
  - Improves memory utilization
**Good Coding Sense**

- **Avoid expensive mathematical operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Min Cycles per iteration (L1 Cache)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i) = y(i)$</td>
<td>1.7</td>
</tr>
<tr>
<td>$x(i) = x(i) + y(i)$</td>
<td>1.7</td>
</tr>
<tr>
<td>$x(i) = x(i) + s*y(i)$</td>
<td>1.7</td>
</tr>
<tr>
<td>$x(i) = 1/y(i)$</td>
<td>15.1</td>
</tr>
<tr>
<td>$x(i) = \sqrt{y(i)}$</td>
<td>18.1</td>
</tr>
</tbody>
</table>

- **Avoid branching within inner loops**

```fortran
do i = 1, N
    do j = 1, N
        y(j,i) = x(j,i) / r(i)
        end do
    end do

do i = 1, N
    oner = 1.0d0 / r(i)
    do j = 1, N
        y(j,i) = x(j,i) * oner
        end do
    end do
```

```fortran
do i = 1, n
    if (r < 1.0e-16) then
        do i = 1, n
            a(i) = 0.0; b(i) = 0.0; c(i) = 0.0
            end do
        else
            a(i) = x(i) / r
            b(i) = y(i) / r
            c(i) = z(i) / r
            end if
        end do
```
Algorithm Improvement: Compare vs. Jump

- Efficiently establish supporting arrays to determine non-zero matrix elements
  - E.g. \( \hat{H}^{pn} = \sum_{ijkl} V_{ijkl}^{pn} \pi_i^{+} \pi_j^{+} \nu_k^{+} \nu_l \)
  - \( H_{pn} = \sum_{ijkl} \left\langle \Phi_f^{p} | \pi_i^{+} \pi_j^{+} | \Phi_i^{p} \right\rangle \left\langle \Phi_f^{n} | \nu_k^{+} \nu_l | \Phi_i^{n} \right\rangle V_{ijkl}^{pn} \)

- “Jump” Algorithm

  \[ \Phi_i \] \[ \Phi_f \]

  \[ \pi_a^{+} \pi_b^{+} \]

  COMPAR\(^E\) initial to final states to see if they’re connected by a one-body operator

  \[ \left\langle \Phi_f \right| \]

  Create all possible JUMP states produced from the operators

  Search the final states for the jump state

  \( N^2 \rightarrow N \log(N) \): Increase subroutine performance by a factor of 5-10 (depending on dim.)
Parallelization Analysis: Load Balancing

WORK DISTRIBUTION ANALYSIS

TAU Output

$^{11}$C, 5hbw

H. Nam
September 26, 2007: Institute for Nuclear Theory
MPI 101

\[ H v_1 = v_2 \implies \]

Each CPU gets a copy of \( v_1 \) and \( v_2 \)
ARPACK (PARPACK) – optimized eigensolver (MFD)
- Limited by system resources
- \(10^B \sim 4\) hours w/ 3500 processors

Hybrid Programming Model – MPI & OpenMP
- Decreases minimum memory requirement

Single Instruction, Multiple Data → Multiple Instruction, Multiple Data
What Else?

- Reconsider algorithms for efficient parallelization

  - Breaks up the large dimension vector

\[
H \mathbf{v}_1 = \mathbf{v}_2 \Rightarrow \begin{bmatrix}
  v_1 \\
  v_2 \\
  \vdots \\
  v_N
\end{bmatrix} = \begin{bmatrix}
  \mathbf{v}_1 \\
  \mathbf{v}_2 \\
  \vdots \\
  \mathbf{v}_N
\end{bmatrix}
\]

\[v_1, v_2 \sim 10^9, \text{real(4)s} \Rightarrow 8\text{GB}\]

Cannot be stored on 1 node

- Storage and on-the-fly
  - 1 proc starts with on-the-fly & another tries to retrieve m.e. If already calculated throw out else use retrieved values

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{bmatrix}
= \begin{bmatrix}
  a_{11}v_1 + a_{12}v_2 + a_{13}v_3 \\
  a_{21}v_1 + a_{22}v_2 + a_{23}v_3 \\
  a_{31}v_1 + a_{32}v_2 + a_{33}v_3
\end{bmatrix}
= \begin{bmatrix}
  \mathbf{a}_1 \\
  \mathbf{a}_2 \\
  \mathbf{a}_3
\end{bmatrix}
\begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{bmatrix}
\]
The Frontier of NCSM Calculations

- Two Body (m-scheme): \(7 \leq N \leq 11, N_{\text{max}}=10\)
  \[N \geq 12, N_{\text{max}} = 6\]
  - Basis dimensions of \(10^8\) have been achieved
  - \(N_{\text{max}}=8\) requires restructuring of code

- Three Body: Practical limit is \(N_{\text{max}} = 6\) for all p-shell nuclei
  - Thus far results (\(N_{\text{max}}=6\) up to \(^6\text{Li}\) (\(^{12}\text{C}^?\)), \(N_{\text{max}}=4\) up to \(^{13}\text{C}\))
  - Our investigations include \(^{15}\text{O},^{16}\text{O},^{17}\text{O},^{9}\text{Be}\) (4hbw... 6hbw?)

- Four Body: \(N_{\text{max}} = 4\)
  - Not only limited by the number of matrix elements
  - Other limiting factors
    - Dimensions of the supporting vectors (1-body jumps, 2-body jumps, \(H_{\text{ppp}}, H_{\text{nnn}}\))
      - E.g. \(\sim 4\) billion two-body jumps (integer(4)) for \(^{15}\text{O}\), 6hbw = 16 GB
    - Lanczos vector storage
Future of HPC

- **Bigger and better?**
  - IBM Power6
  - If we need ~ 200,000 GB →
    (> 3000 nodes w/ 64 GB each)

- **Bring out the big guns!**
  - BlueGene/L
  - 65536 nodes / 131072 processors
  - 512 MB… yikes! → ~ 33,500 GB total

- **Hope**
  - DARPA, High Productivity Computing
  - Use what you’ve got wisely.
  - Collaborate with Computer Scientists
3-Body Forces are essential for ab initio nuclear structure calculations

Computationally challenging due to memory and run-time performance limitations

On-the-fly diagonalization methods provide greatest scaling potential (or a combination)

Need efficient algorithms (suited for parallel optimization)

Need to analyze workload distribution

Need efficient parallelization schemes

Considerable effort is still needed!!
Acknowledgements

- Calvin W. Johnson, SDSU
- W. Erich Ormand, N-Division, LLNL
- Greg Bronevetsky, CASC, LLNL
- Petr Navratil, N-Division, LLNL
- Department of Energy