PROGRESS AND APPLICATIONS OF BIGSTICK CI CODE

Plamen Krastev, Calvin Johnson and Erich Ormand

1. San Diego State University
2. Lawrence Livermore National Laboratory

Outline:

- Reminders about BIGSTICK
- Past year accomplishments
- Remaining challenges
- Roadmaps

SciDAC - UNEDF meeting, Lansing MI, June 2010

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344
BIGSTICK:

REDSTICK → BIGSTICK

- General purpose M-scheme configuration interaction (CI) code
- On-the-fly calculation of the many-body Hamiltonian
- Fortran 90 and MPI
- 30,000+ lines in 30+ files and 150+ subroutines
- Faster set-up
- Faster Hamiltonian application
- Rewritten for “easy” parallelization
- New parallelization scheme
Key idea for on-the-fly algorithms:

- Represent an area by its boundary
- Factorization of problem
- Reduces dramatically memory load

= X
Key idea for on-the-fly algorithms:

Represent an area by its boundary

- Factorization of problem
- Reduces dramatically memory load

\[ |\alpha\rangle = |\alpha_p\rangle \times |\alpha_n\rangle \]
Key idea for on-the-fly algorithms:

Represent an area by its boundary

- Factorization of problem
- Reduces dramatically memory load

\[ |\alpha\rangle = |\alpha_p\rangle \times |\alpha_n\rangle \]

Hamiltonian can be factorized similarly

Proton states

Neutron states

No of states in total basis
Why “on-the-fly”?

- Reduced memory requirements
- Can fit larger problems on same machines

### Comparison of RAM requirements (2-body interactions only)

Does not include lanczos vector storage

<table>
<thead>
<tr>
<th>Nuclide</th>
<th>Space</th>
<th>Basis dim</th>
<th>matrix store</th>
<th>on-the-fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^{56}\text{Fe}$</td>
<td>$pf$</td>
<td>501 M</td>
<td>290 Gb</td>
<td>0.72 Gb</td>
</tr>
<tr>
<td>$^{7}\text{Li}$</td>
<td>$N_{\text{max}}=12$</td>
<td>252 M</td>
<td>3600 Gb</td>
<td>96 Gb</td>
</tr>
<tr>
<td>$^{7}\text{Li}$</td>
<td>$N_{\text{max}}=14$</td>
<td>1200 M</td>
<td>23 Tb</td>
<td>624 Gb</td>
</tr>
<tr>
<td>$^{12}\text{C}$</td>
<td>$N_{\text{max}}=6$</td>
<td>32M</td>
<td>196 Gb</td>
<td>3.3 Gb</td>
</tr>
<tr>
<td>$^{12}\text{C}$</td>
<td>$N_{\text{max}}=8$</td>
<td>590M</td>
<td>5000 Gb</td>
<td>65 Gb</td>
</tr>
<tr>
<td>$^{12}\text{C}$</td>
<td>$N_{\text{max}}=10$</td>
<td>7800M</td>
<td>111 Tb</td>
<td>1.4 Tb</td>
</tr>
<tr>
<td>$^{16}\text{O}$</td>
<td>$N_{\text{max}}=6$</td>
<td>26 M</td>
<td>142 Gb</td>
<td>3.0 Gb</td>
</tr>
<tr>
<td>$^{16}\text{O}$</td>
<td>$N_{\text{max}}=8$</td>
<td>990 M</td>
<td>9700 Gb</td>
<td>130 Gb</td>
</tr>
</tbody>
</table>
Why “on-the-fly”?

Comparison of RAM requirements (3-body interactions) - Estimate

<table>
<thead>
<tr>
<th>Nuclide</th>
<th>Space</th>
<th>Basis dim</th>
<th>store</th>
<th>on the fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^7$Li</td>
<td>$N_{\text{max}}=12$</td>
<td>252 M</td>
<td>100 Tb</td>
<td>2.6 Tb</td>
</tr>
<tr>
<td>$^7$Li</td>
<td>$N_{\text{max}}=14$</td>
<td>1200 M</td>
<td>760 Tb</td>
<td>20 Tb</td>
</tr>
<tr>
<td>$^{12}$C</td>
<td>$N_{\text{max}}=6$</td>
<td>32M</td>
<td>4 Tb</td>
<td>0.07 Tb</td>
</tr>
<tr>
<td>$^{12}$C</td>
<td>$N_{\text{max}}=8$</td>
<td>590M</td>
<td>180 Tb</td>
<td>3 Tb</td>
</tr>
<tr>
<td>$^{12}$C</td>
<td>$N_{\text{max}}=10$</td>
<td>7800M</td>
<td>5000 Tb</td>
<td>86 Tb</td>
</tr>
</tbody>
</table>

1 Tb requires approximately 1,000 cores (depending on architecture)
Past year accomplishments:

- Completed modeling of new parallelization scheme (Krastev) \(\checkmark\)
- Fully distributed matrix-vector multiply (Krastev) \(\checkmark\)
- Major steps towards distributing Lanczos vectors (Krastev) a little behind
- Significant progress in 3-body implementation (Johnson & Ormand) behind schedule

Latest version (v6.2.5.) available at

/project/projectdirs/unedf/lcci/BIGSTICK/v625
Past year accomplishments:

- Completed modeling of new parallelization scheme (Krastev) ✓
- Fully distributed matrix-vector multiply (Krastev) ✓
- Major steps towards distributing Lanczos vectors (Krastev) a little behind
- Significant progress in 3-body implementation (Johnson & Ormand) behind schedule

Latest version (v6.2.5.) available at

/project/projectdirs/unedf/lcci/BIGSTICK/v625

In March we hosted the second annual meeting on Leadership Class CI codes at SDSU.

Attendees:

Johnson, Krastev
Vary, Maris, Liu
Navratil
Horoi
Ng, Yang, Calderon
Distribution of matrix-vector multiplication:

$^{52}\text{Fe}, 4000$ cores

$^{52}\text{Fe}, 8000$ cores
(Relative) speedup of matrix-vector multiplication:

\[ \text{relative speedup} \]

\( ^{52}\text{Fe}, 50 \text{ lanczos iterations} \)

10,000 cores

\[ \text{speedup} \]

\[ \text{# proc} \]

Ideal

Matvec

P. Krastev – SciDAC UNEDF meeting, Lansing MI, June 23, 2010
Distributing Lanczos vectors:

We know exactly which parts of Lanczos vectors are needed by each core.

Several cores usually need same sectors of Lanczos vector.

We employ hermiticity – a given core needs same parts from initial and final Lanczos vector, i.e. locally still $\dim(V_{\text{in}}) = \dim(V_{\text{out}})$.
**Distributing Lanczos vectors:**

We know exactly which parts of Lanczos vectors are needed by each core

Several cores usually need same sectors of Lanczos vector

We employ hermiticity – a given core needs same parts from initial and final Lanczos vector, i.e. locally still \( \text{dim}(V_{\text{in}}) = \text{dim}(V_{\text{out}}) \)

**Practical steps:**

✓ Define separate communication groups for each sector of Lanczos vector **DONE**

✓ Define communicator to handle dot products (each core needs only the updated scalar product) **DONE**

✓ Organize Lanczos diagonalization in terms of Lanczos sectors **DONE** (needs further optimization, specifically reorthogonalization and parallel I/O)

Collaborating with Yang and Ng (Berkeley Lab) we were able to improve significantly reorthogonalization and parallel I/O
Lanczos vector sectors:

\(^{22}\text{Ne}\) in \(sd\)-shell (with inert \(^{16}\text{O}\) core): Total \(J_z = 0\) (first 4 sectors are given)

<table>
<thead>
<tr>
<th>Proton sectors</th>
<th>Neutron sectors</th>
<th>Lanczos vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>(J^p_z = -4: 2\ SDs)</td>
<td>(J^n_z = +4: 24\ SDs)</td>
<td>1 Dim=48</td>
</tr>
<tr>
<td>(J^p_z = -3: 4\ SDs)</td>
<td>(J^n_z = +3: 39\ SDs)</td>
<td>2 Dim=156</td>
</tr>
<tr>
<td>(J^p_z = -2: 9\ SDs)</td>
<td>(J^n_z = +2: 60\ SDs)</td>
<td>3 Dim=540</td>
</tr>
<tr>
<td>(J^p_z = -1: 11\ SDs)</td>
<td>(J^n_z = +1: 72\ SDs)</td>
<td>4 Dim=792</td>
</tr>
</tbody>
</table>
Communication groups for Lanczos sectors:

Lanczos vector

1
2
3
4
5
...
N
**Communication groups for Lanczos sectors:**

<table>
<thead>
<tr>
<th>Lanczos vector</th>
<th>communicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>icomm1</td>
</tr>
<tr>
<td></td>
<td>rank0 rank2</td>
</tr>
<tr>
<td></td>
<td>rank5 rank13</td>
</tr>
<tr>
<td>2</td>
<td>icomm2</td>
</tr>
<tr>
<td></td>
<td>rank3 rank5</td>
</tr>
<tr>
<td></td>
<td>rank7 rank16</td>
</tr>
<tr>
<td></td>
<td>rank25</td>
</tr>
<tr>
<td>3</td>
<td>icomm3</td>
</tr>
<tr>
<td></td>
<td>rank10 rank25</td>
</tr>
<tr>
<td></td>
<td>rank30</td>
</tr>
<tr>
<td>4</td>
<td>icomm4</td>
</tr>
<tr>
<td></td>
<td>rank25 rank6</td>
</tr>
<tr>
<td></td>
<td>rank15</td>
</tr>
<tr>
<td>5</td>
<td>icomm5</td>
</tr>
<tr>
<td></td>
<td>rank1 rank3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>icommN</td>
</tr>
<tr>
<td></td>
<td>rank0 rank8</td>
</tr>
<tr>
<td></td>
<td>rank12 rank13</td>
</tr>
<tr>
<td></td>
<td>rank16 rank30</td>
</tr>
</tbody>
</table>

Ranks in global communicator MPI_COMM_WORLD
Dimension of Lanczos vector sectors - single shell:

\(^{52}\text{Fe}, \text{space – pf, basis dimension} = 109,954,620\)

\# Lanczos sectors = 27 ( = \# communication groups )
Dimension of Lanczos vector sectors - multi shell:

$^{12}\text{C}$, space – $N_{\text{max}} = 6$, basis dimension = 32,598,920

# Lanczos sectors = 75 ( = # communication groups )
Dot product in terms of Lanczos sectors:

(i) Each core needs global scalar product

(ii) Define communicator to handle dot products – consider only masters with unique global ranks – ROOT COMMUNICATOR

(iii) Calculate dot product over ROOT_COMM

(iv) Update dot product on all cores
Dot product in terms of Lanczos sectors:

(i) Each core needs global scalar product

(ii) Define communicator to handle dot products – consider only masters with unique global ranks – ROOT COMMUNICATOR

(iii) Calculate dot product over ROOT_COMM

(iv) Update dot product on all cores
Dot product, cont'd:

basis dimension = 1,963,461
# Lanczos sectors = 21
Dot product, cont'd:

Global ranks of roots for each Lanczos sector communicator

basis dimension = 1,963,461
# Lanczos sectors = 21

\(^{48}\text{Cr}, \text{nproc} = 200\)
Dot product, cont’d:

Global ranks of roots for each Lanczos sector communicator

```
basis dimension = 1,963,461
# Lanczos sectors = 21
```

```
48Cr, nproc = 200
```

```
ROOT_COMM

<table>
<thead>
<tr>
<th>Rank = 0</th>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank = 1</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</td>
</tr>
<tr>
<td>Rank = 2</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</td>
</tr>
<tr>
<td>Rank = 3</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</td>
</tr>
<tr>
<td>Rank = 5</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</td>
</tr>
<tr>
<td>Rank = 7</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21</td>
</tr>
</tbody>
</table>
```

P. Krastev – SciDAC UNEDF meeting, Lansing MI, June 23, 2010
Lanczos algebra:

(i) At each iteration, each core needs to have its Lanczos sectors updated

(ii) Multiple cores update the same Lanczos sectors

(iii) Overlapping communication groups (same global ranks belong to different groups) – cannot make simultaneous MPI calls
Lanczos algebra:

(i) At each iteration, each core needs to have its Lanczos sectors updated

(ii) Multiple cores update the same Lanczos sectors

(iii) Overlapping communication groups (same global ranks belong to different groups) – cannot make simultaneous MPI calls

```plaintext
... DO I = 1, # communication groups IF ( IPROC(I) >= 0 .AND. IPROC(I) <= NPROC(I) – 1 ) THEN CALL MPI_ALLREDUCE(........ICOMM(I),IERR) END IF END DO...
```
**Computational challenges:**

1. The larger the problem, the larger the number of overlapping communication groups, i.e. larger communication overhead. How can this be reduced?

2. **Reorthogonalization / Parallel I/O**

   Naive reorthogonalization scheme is not suitable – it assumes each core has a copy of the whole Lanczos vector.

   Preliminary reorthogonalization scheme working with Lanczos sectors needs optimization. **Open issue (consider various algorithms)**

3. **Include “memory cap” for Lanczos sectors and rebalance workload.** Challenging, but allows for greatest flexibility (enables runs on memory constrained machines, e.g., BlueGene – typically ~1GB/processor)

4. **Distribute 3-body Hamiltonian** (compute core assignments and deliver 3-body matrix elements)
**Computational challenges:**

**Reorthogonalization:**

- We reorthogonalize each new Lanczos vector against all old ones
- Lanczos vectors are stored on disk. **This requires fast I/O**

(For large number of processors consider storing Lanczos vectors on core – MPI calls less expensive than I/O)
Computational challenges:

Reorthogonalization:

- We reorthogonalize each new Lanczos vector against all old ones
- Lanczos vectors are stored on disk. This requires fast I/O

(For large number of processors consider storing Lanczos vectors on core – MPI calls less expensive than I/O)

Current MPI I/O scheme:

\(^{48}\text{Cr, space – pf, basis dimension = 1,963,461}\)

# Lanczos sectors = 21 ( = # communication groups)
Computational challenges:

Reorthogonalization:

- We reorthogonalize each new Lanczos vector against all old ones
- Lanczos vectors are stored on disk. This requires fast I/O

( For large number of processors consider storing Lanczos vectors on core – MPI calls less expensive than I/O )

Current MPI I/O scheme:

\[48\text{Cr, space – pf, basis dimension } = 1,963,461\]
# Lanczos sectors = 21 ( = # communication groups )

<table>
<thead>
<tr>
<th>ROOT_COMM (global ranks)</th>
<th>Rank = 0</th>
<th>Rank = 1</th>
<th>Rank = 2</th>
<th>Rank = 3</th>
<th>Rank = 5</th>
<th>Rank = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lanczos sectors</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>20</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local dimension of Lanczos vector for ROOT_COMM

\[1,323,399, 233,289, 307,970, 19,601, 39,601, 39,601\]
**Roadmap:**

**The Goal:**
Efficient, scalable, platform-independent application running on 1 – 100,000+ cores with 3-body forces

**Where are we at:**
- Fully distributed matrix-vector multiplication (2-body)
- Completing distributing Lanczos vectors *(finish by 9/2010)*
- Completing 3-body implementation *(finish by 9/2010)*

**The Map for Year 5:**
- Paper on factorization *(12/2010)*
- Store Lanczos vectors on core *(12/2010)*
- Include memory cap and rebalance *(12/2010)*
- Optimize reorthogonalization and I/O for efficient performance *(12/2010)*
- Parallelize 3-body *(3/2011)*
- Publish BIGSTICK *(6/2011)*