Solving Large-scale Eigenvalue Problems in Nuclear Structure Calculation

Chao Yang, LBNL

in collaboration with

LBNL

Hasan Metin Aktulga

Esmond Ng

IBM ILOG

Philip Sternberg

Iowa State University

James Vary

Pieter Maris

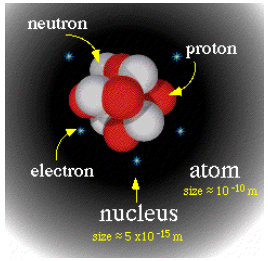
Feb 16, 2011

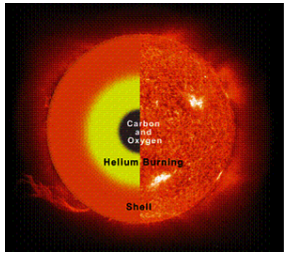
Outline

- Overview of nuclear structure calculation using configuration interaction (CI)
  - Background
  - Numerical methods for solving large-scale eigenvalue problems
  - Parallel implementation
    - Constructing the matrix
    - Data distribution
    - Load balancing
    - Parallel sparse matrix vector multiplication
- Total-J calculation
  - Large-scale null space calculations
- Challenges

Nuclear Structure Calculation

- Strong interactions among protons and neutrons, origin of the ^{12}C formation in stars, foundation for nuclear reaction theory ...





- Quantum many-body problem

$$\mathcal{H}\Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_k) = \lambda\Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_k).$$

  - $\mathcal{H}$ – nuclear Hamiltonian describes kinetic energy and 2-body (NN), 3-body (NNN) potential;
  - Ψ – nuclear wavefunction, $|\Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_k)|^2$ probability density of finding nucleons $1, 2, ..., k$ at $\mathbf{r}_1$, $\mathbf{r}_2$, ..., $\mathbf{r}_k$;
  - λ – quantized energy level. Often interested in the ground state (λ_1) and a few (10-100) low excited states;

- Solving the many-body problem directly is not feasible except for small k;

Nuclear Configuration Interaction

- Basis expansion $\Psi = \sum_a \alpha_a \Phi_a(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_k)$, where $a \equiv (a_1, a_2, ..., a_k)$, $a_i \in [1, i_{\max}]$.
- Many-body (MB) state (Slater determinant)

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

- Single-particle state: ϕ_{a_i} is an eigenfunction of a harmonic oscillator, associated with a set of quantum numbers $|n\ell j m_j\rangle_{a_i}$;
- The size of the expansion (N) depends on $i_{\max}$, k and several constraints
  - $\sum_{a_i \in a} 2n_{a_i} + \ell_{a_i} \leq N_0 + N_{max}$;
  - $\sum_{a_i \in a} m_{j\,a_i} = M_0$;
  - parity constraint;

Finite-dimensional Eigenvalue Problem

- $\widehat{H}x = \lambda x$, where

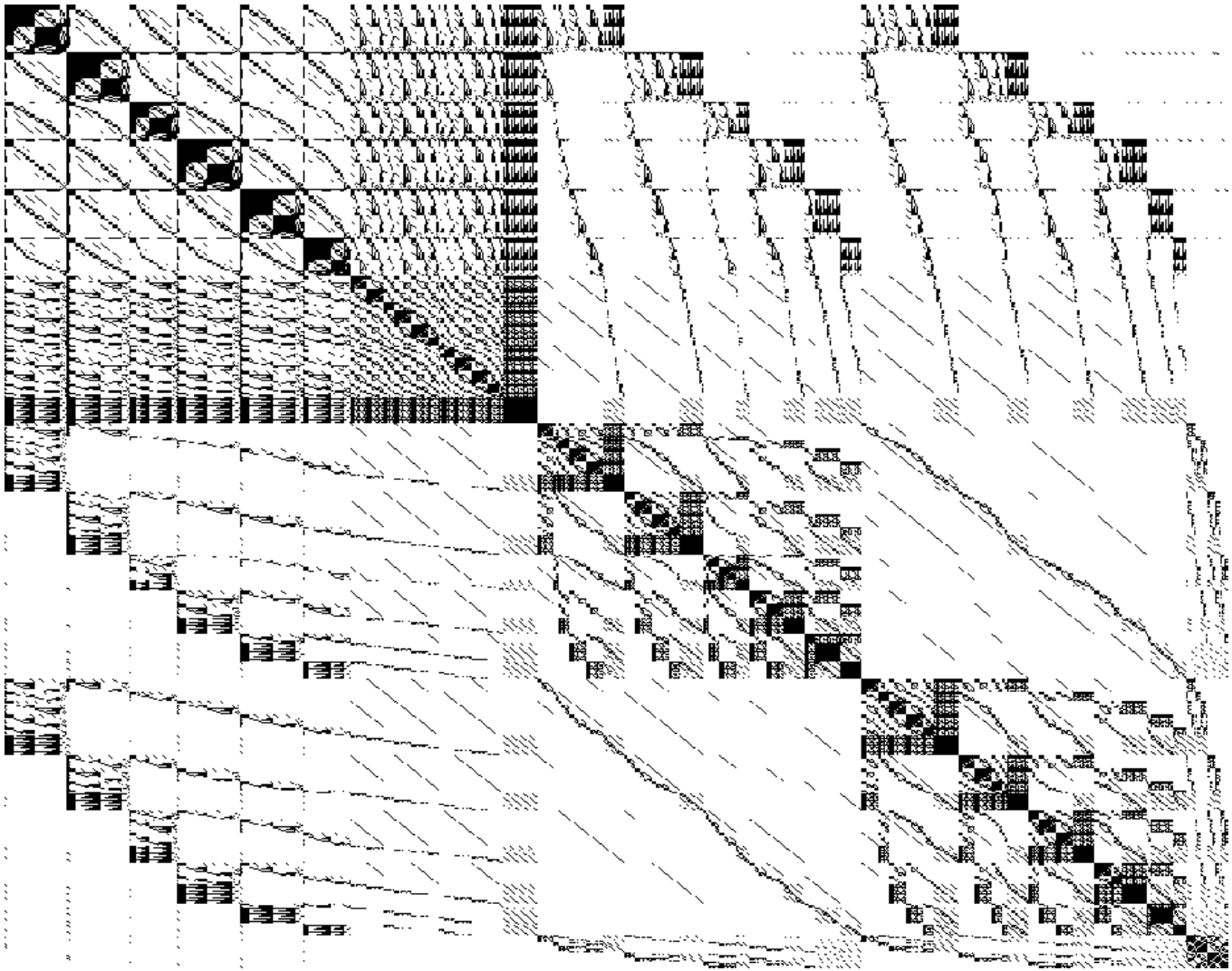
$$\widehat{H}_{a,b} = \int_{\Omega} (\Phi_a^* \mathcal{H} \Phi_b) d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_k, \quad x = (\alpha_1, \alpha_2, \cdots, \alpha_N)^T,$$

  and $a = (a_1, a_2, ..., a_k)$, $b = (b_1, b_2, ..., b_k)$.
- Dimension of $\widehat{H}$ can be quite large
- $\widehat{H}$ is quite sparse.
  - Sparsity follows from the orthornormality of ϕ's, and the 2 or 3-body interacting potential in $\mathcal{H}$:

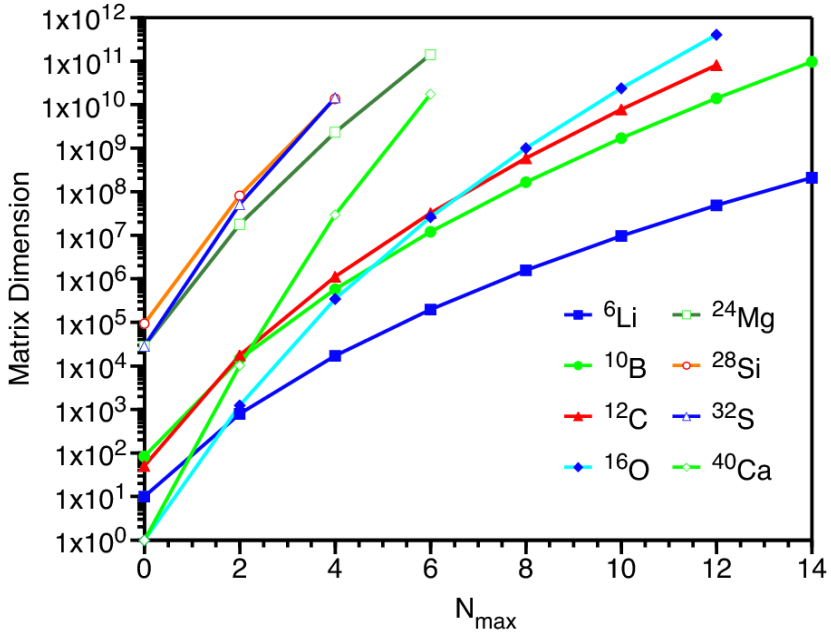
    If a and b are many-body states that differ by more than 2 (or 3) single-particle states, the matrix element indexed by a and b is exactly zero.

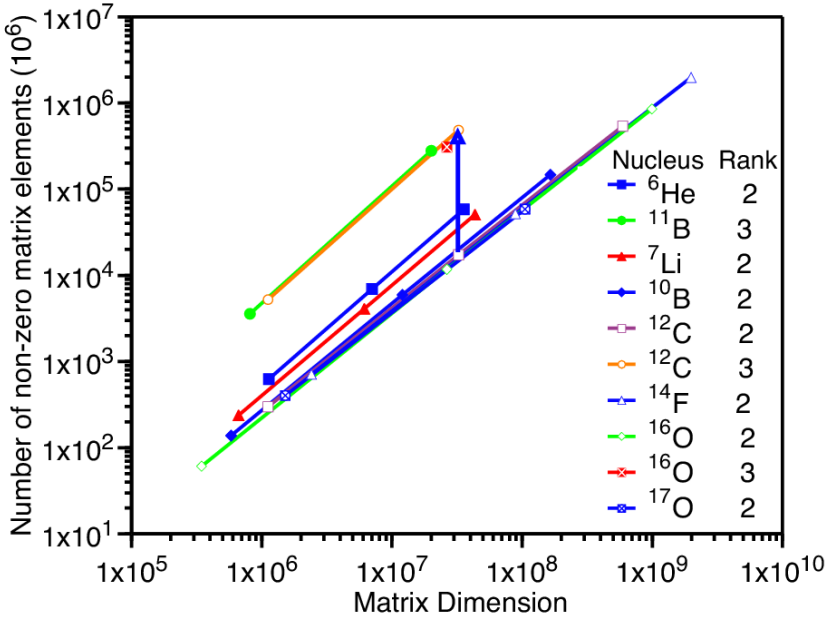
  - No "nice" pattern (e.g., banded structure)

Sparsity Structure for ^{6}Li



Matrix size and sparsity





Rule of thumb: number of nonzeros $\sim O(N^{1.5})$

Dimensions and sparsity of matrices

- Estimates of aggregate memory needed for storage of sparse symmetric Hamiltonian matrix in compressed column format

(does not include memory for vectors)

nucleus	$N_{\max}$	dimension	2-body	3-body	4-body
^{6}Li	12	$4.9 \cdot 10^{7}$	0.6 TB	33 TB	590 TB
^{12}C	8	$6.0 \cdot 10^{8}$	4 TB	180 TB	4 PB
^{12}C	10	$7.8 \cdot 10^{9}$	80 TB	5 PB	140 PB
^{16}O	8	$9.9 \cdot 10^{8}$	5 TB	300 TB	5 PB
^{16}O	10	$2.4 \cdot 10^{10}$	230 TB	12 PB	350 PB
^{8}He	12	$4.3 \cdot 10^{8}$	7 TB	300 TB	7 PB
^{11}Li	10	$9.3 \cdot 10^{8}$	11 TB	390 TB	10 PB
^{14}Be	8	$2.8 \cdot 10^{9}$	24 TB	1100 TB	28 PB
^{20}C	8	$2 \cdot 10^{11}$	2 PB	150 PB	6 EB
^{28}O	8	$1 \cdot 10^{11}$	1 PB	56 PB	2 EB

(presented at *Extreme Scale Computing Workshop – nuclear physics* Washington DC Jan 2009)

- Need high-performance computing on large-memory platforms

Types of Calculations

- In many cases, we are interested in the ground state of $\widehat{H}$ and a few low excited states,i.e., we compute 10-20 smallest eigenvalues of $\widehat{H}$
- In some applications, we are interested in a large number of low energy states with a prescribed total angular momentum **J** (Total-J calculation)
  - Compute a large number of eigenvalues, then pick out the ones with the desired **J**
  - Use the fact that $[\widehat{H}, \widehat{J^2}] = \widehat{H}\widehat{J^2} - \widehat{J^2}\widehat{H} = 0$ to simultaneously diagonalize $\widehat{H}$ and $\widehat{J^2}$
    1. Compute an invariant subspace Z of $\widehat{J^2}$ associated with a prescribed **J** (null space calculation)
    2. Project $\widehat{H}$ into Z, i.e. $G = Z^T \widehat{H} Z$
    3. Compute desired eigenvalues and eigenvectors of G
    4. Back transformation

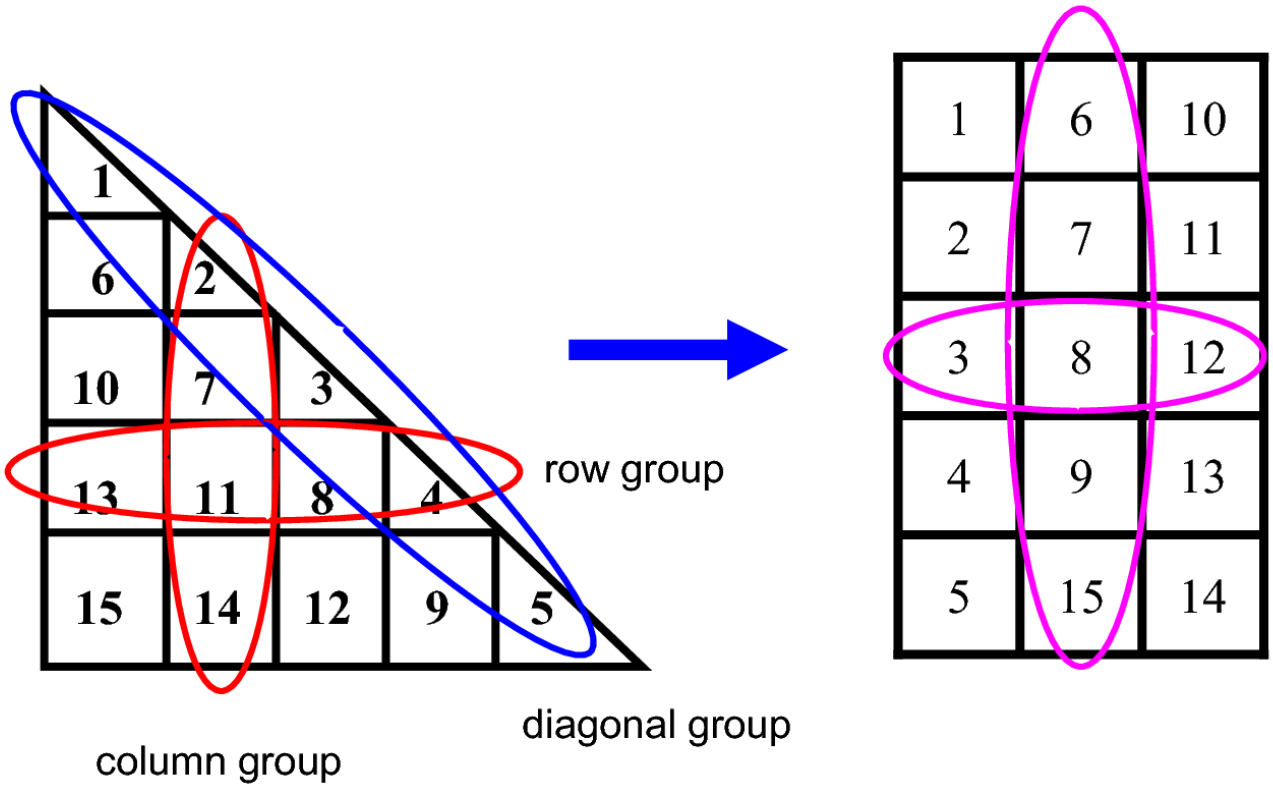
Basic Steps of MFDn

- Generate (enumerate) and distribute MB states (to achieve load balance) (MB states viewed as column and row indices of $\widehat{H}$)
- Matrix Hamiltonian construction
  - Figure out where the nonzeros are before evaluating and storing them
  - Efficient data structure
  - Numerical evaluation
- (Compute desired invariant subspace of $\widehat{J}^2$)
- Solve large sparse matrix eigenvalue problem by Lanczos
  - Efficient and scalable matrix-vector (MATVEC) multiplication
  - Efficient and scalable orthogonalization
- Evaluate observables

Hamiltonian construction	$\sim$ 1500 wall clock seconds
Lanczos	$\sim$ 2500 wall clock seconds

^{16}O, $N_{\max} = 8$, $N \sim 10^9$, 12,090 cores on Franklin

Processor Grid and Communication Groups

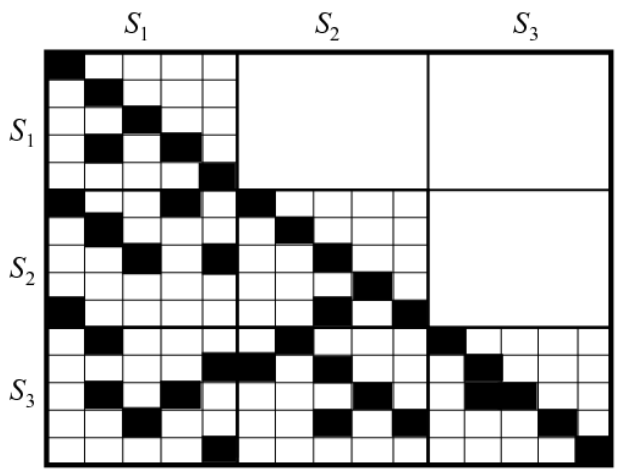


MB State Generation and Distribution

- Enumerate by lexigraphical order: let $a = (a_1, a_2, ..., a_k)$ and $b = (b_1, b_2, ..., b_k)$, where $a_i, b_i \in [1, i_{\max}]$

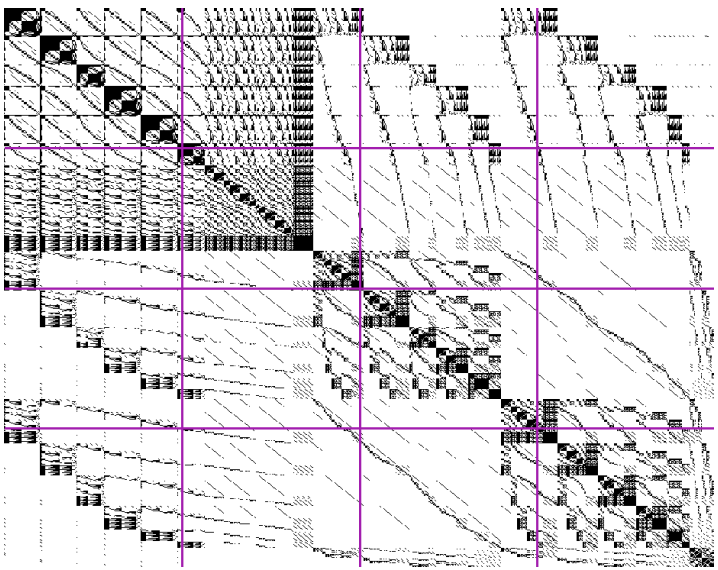
$$a < b \quad \text{iff} \quad \exists j \quad \text{such that} \quad a_j < b_j \quad \text{and} \quad a_i = b_i \quad \forall i < j.$$

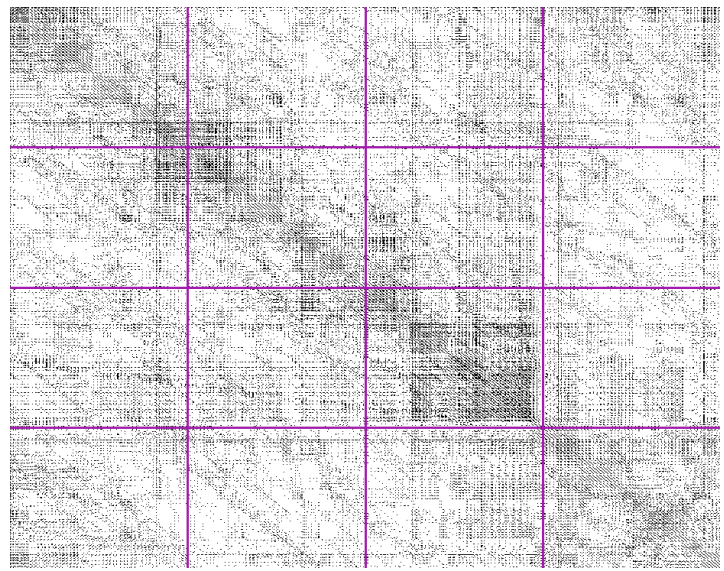
- Validity check
- MB state distribution objectives:
  - Partition **valid** MB states into groups S_1, S_2,..., S_{n_g} of approximately equal sizes;
  - The number of nonzeros $\widehat{H}_{a,b}$ in each (S_i, S_j) block is approximately the same;
  - Efficient and scalable;



Parallel MB State Generation & Cyclic Distribution

- The ith processor increment the smallest possible MB state $i-1$ times;
- Each processor performs n_g-fold increment simultaneously;
- Discard MB state if it is not valid;





Hamiltonian Matrix Construction

Rows/columns indexed by **many-body states**

$$a = \overbrace{(a_1, a_2, \ldots, a_k)}^{\textbf{many-body state}} \qquad : \qquad a_i < a_{i+1}$$

a_i's are **single-particle states**

Physics excludes most of the $\binom{i_{\max}}{k}$ many-body states

If a and b are many-body states that differ by more than 2 (or 3) single-particle states, the matrix element indexed by a and b is exactly zero.

If not, we call a and b an **interacting pair**.

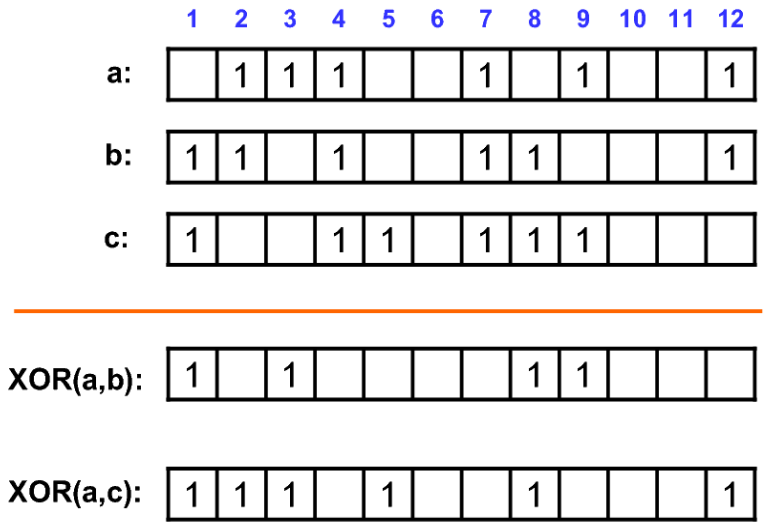
Example

- If a 2-body potential is used in $\mathcal{H}$,

$$\begin{aligned} a &= (2, 3, 4, 7, 9, 12) \\ b &= (1, 2, 4, 7, 8, 12) \\ c &= (1, 4, 5, 7, 8, 9) \end{aligned}$$

  are many-body states, then (a, c) is not an interacting pair, but (a, b) and (b, c) are interacting pairs.

- Implementation: bitwise operation



The Need for Blocking

- Exhaustive pairwise comparison is prohibitively expensive
- Would like to identify large zero blocks without performing pairwise comparisons
- Group MB states into clusters, create a cluster identifier for each cluster, compare cluster id's
- Partition the single-particle states into bins, count how many single-particle states are in each bin.

E.g., using the partition $\{[1\text{-}4],[5\text{-}8],[9\text{-}12]\}$, we have

many-body states	**cluster identifiers**
(2,3,4,7,9,12)	(3,1,2)
(1,2,4,7,9,12)	(3,1,2)
(1,4,5,7,8,9)	(2,3,1)
(1,2,9,10,11,12)	(2,0,4)

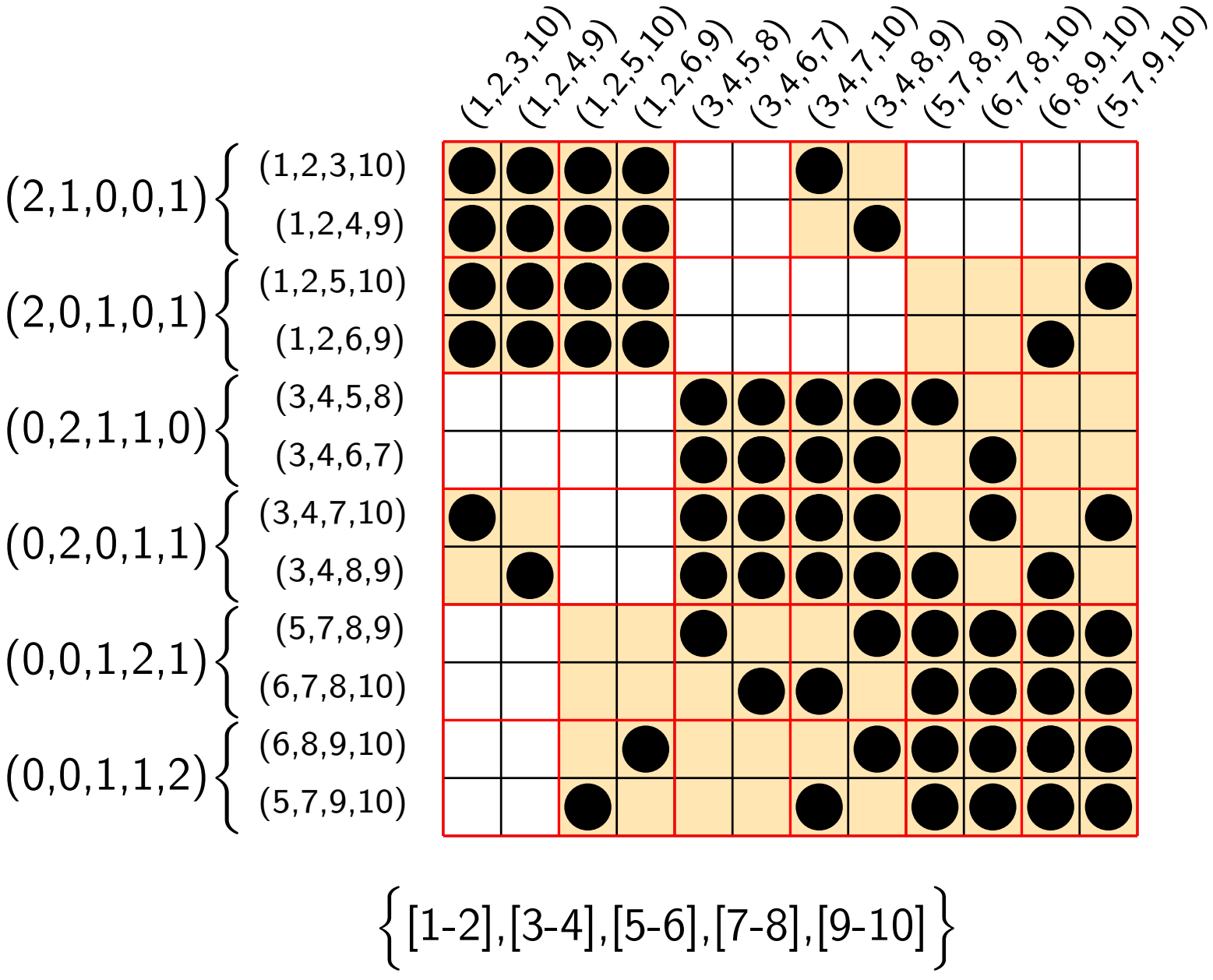
The Need for Blocking (Continued)

E.g., using the partition $\{[1\text{-}4],[5\text{-}8],[9\text{-}12]\}$, we have

many-body states	**cluster identifiers**
(2,3,4,7,9,12)	(3,1,2)
(1,2,4,7,9,12)	(3,1,2)
(1,4,5,7,8,9)	(2,3,1)
(1,2,9,10,11,12)	(2,0,4)

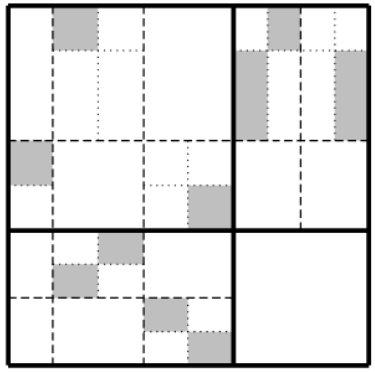
Claim: Let S and T be cluster identifiers with $\|S - T\|_1 > 4$. Then $\mathcal{H}_{\{S,T\}} = 0$.

Tiny Example with Blocking



$$\{[1\text{-}2],[3\text{-}4],[5\text{-}6],[7\text{-}8],[9\text{-}10]\}$$

Performance Gain from Multi-level Blocking



- Nucleus: ^{16}O
- Configuration space: $N_{\max} = 8$, $N = 10^9$
- Number of processors: 12,090

Number of levels	time (seconds)	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}

Parallel Eigenvalue Computation

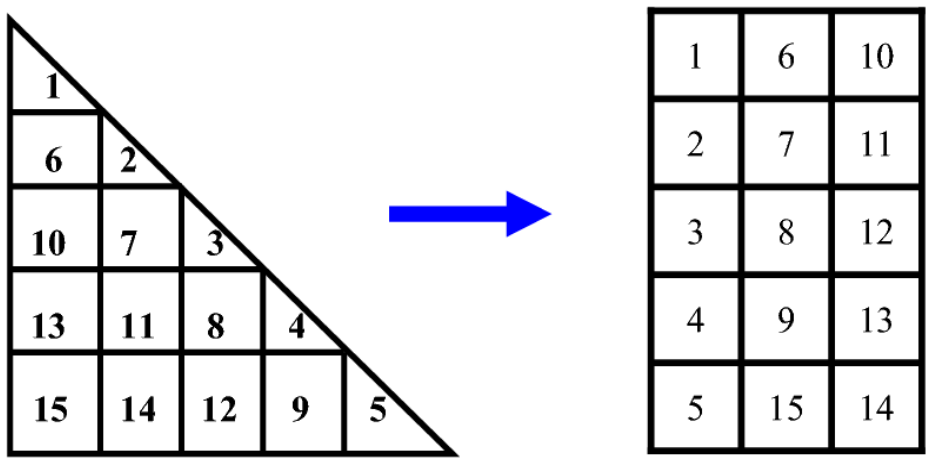
- Solved by Lanczos iteration (with implicit restart)

$$\widehat{H}V = VT + fe_m^T, \quad V^T V = I_m$$

- Perform $y \leftarrow \widehat{H}x$ many times
- Maintain $V^T V = I$
- Memory bound: ^{16}O Hamiltonian uses 6 terabytes
- Store lower half of matrix, distributed across:

d **diagonal** processors

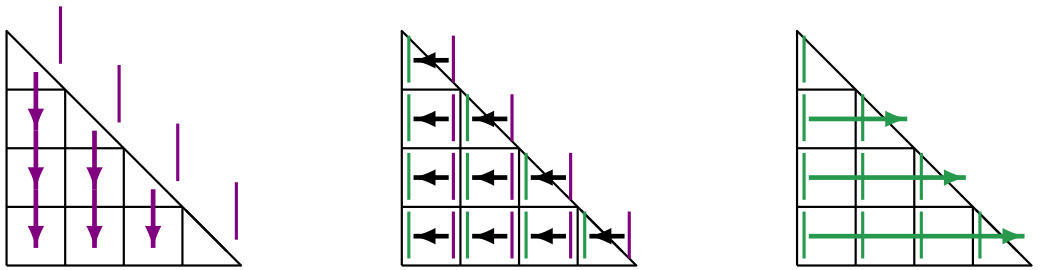
$d(d+1)/2$ total processors



Matrix-Vector Multiply

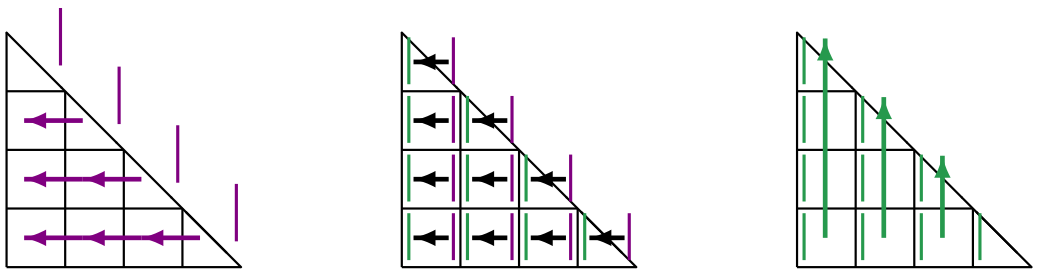
Steps for MATVEC: input (x) and output (y) vectors are stored on diagonal processors

1.



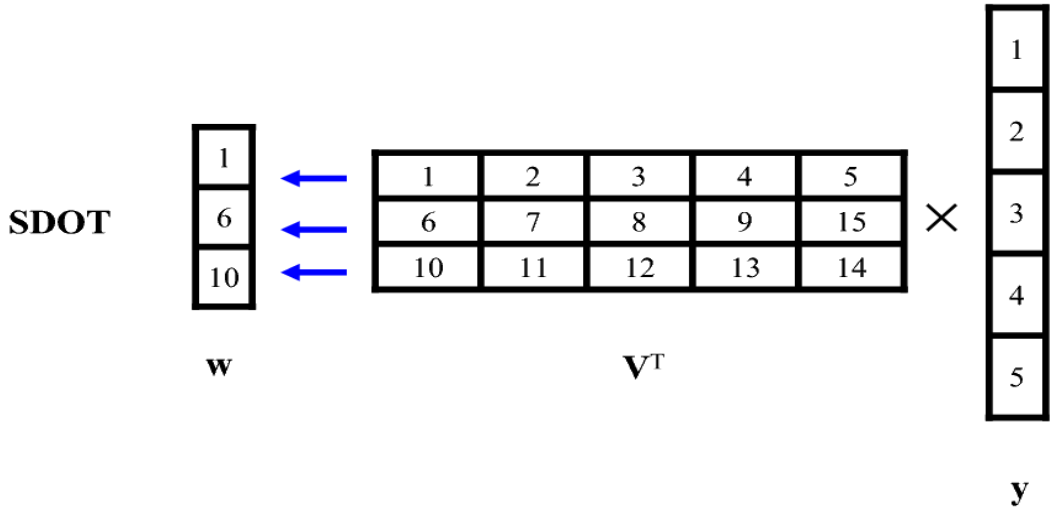
BCast(x) $y \leftarrow Ax$ Reduce(y)

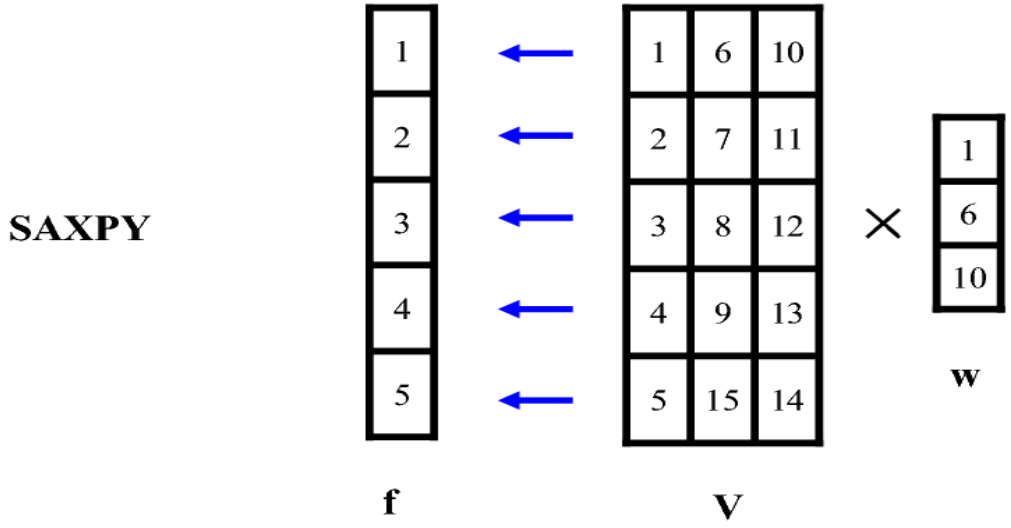
2.



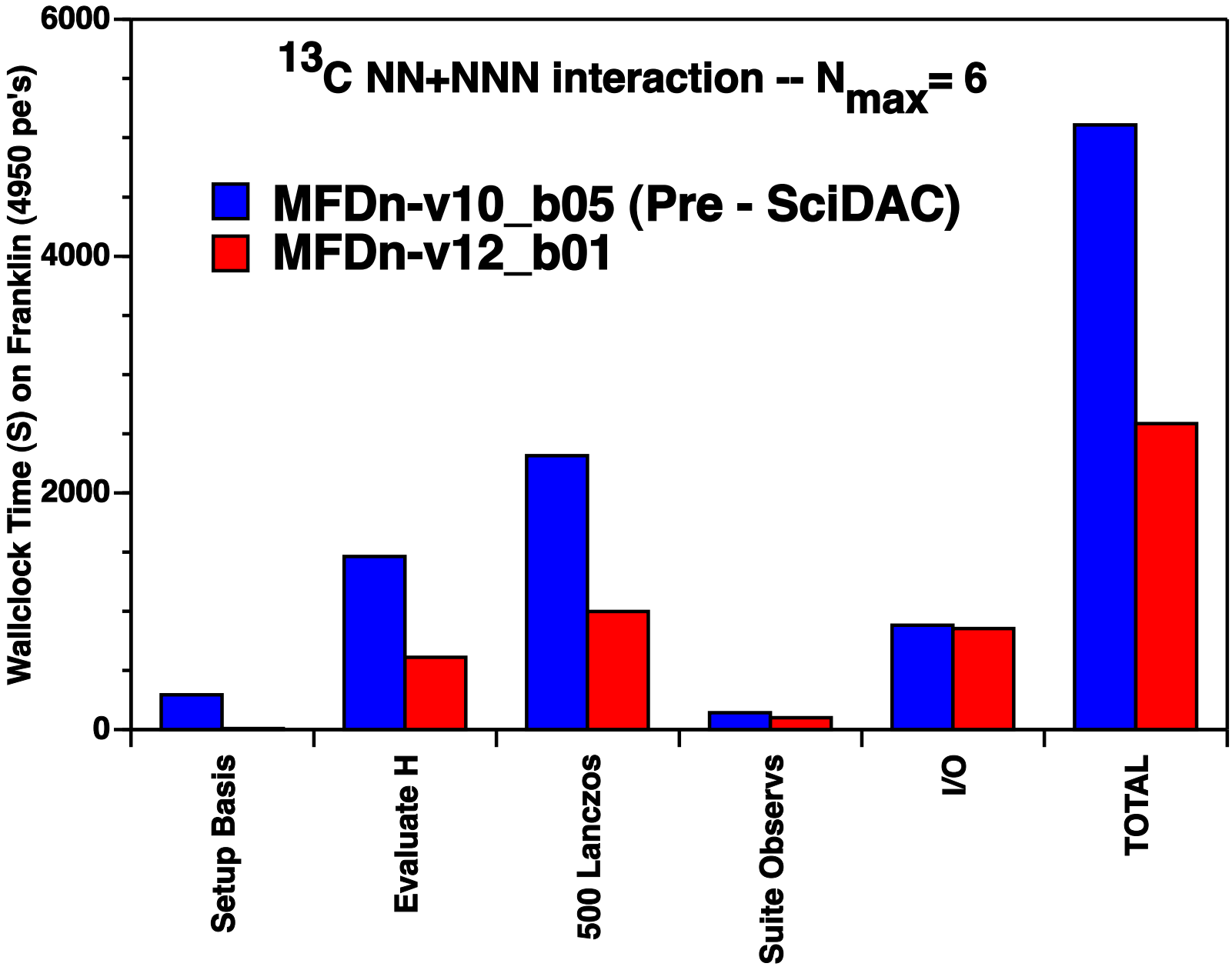
BCast(x) $y \leftarrow A^T x$ Reduce(y)

Parallel Orthogonalization $f \rightarrow V(V^T y)$

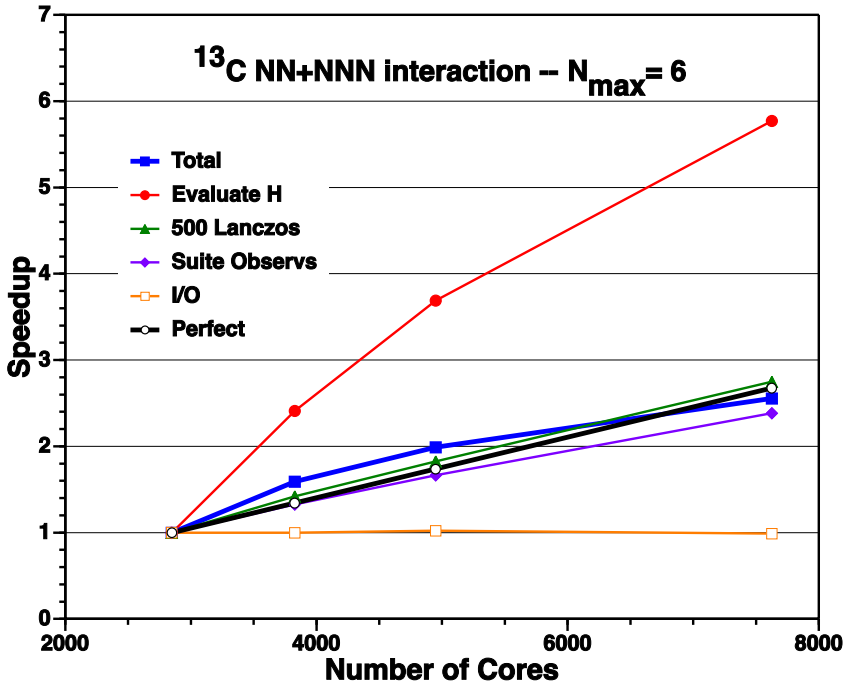


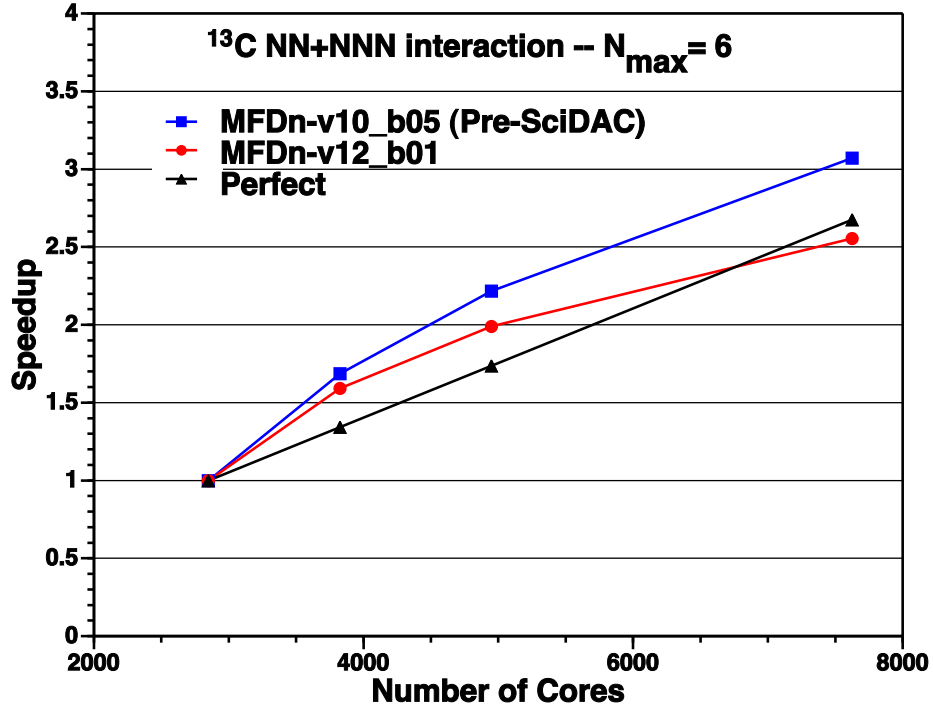


Overall Performance of MFDn



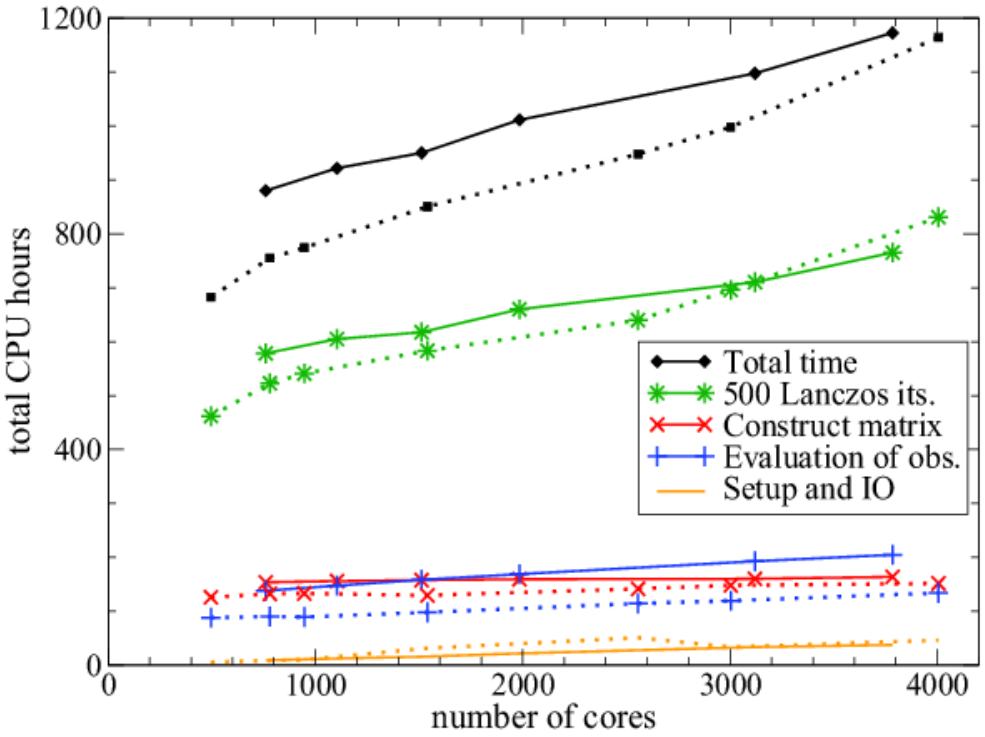
Scalability of MFDn





Total CPU time of MFDn on Cray XT4 with hybrid MPI/OpenMP

- For application scientist, time to completion, or CPU resource units used, is more important than speedup



^{6}Li, $N_{\max} = 12$,
2-body interactions
on Franklin (NERSC)

solid: hybrid MPI/OMP
1 MPI PE per node
with 4 threads

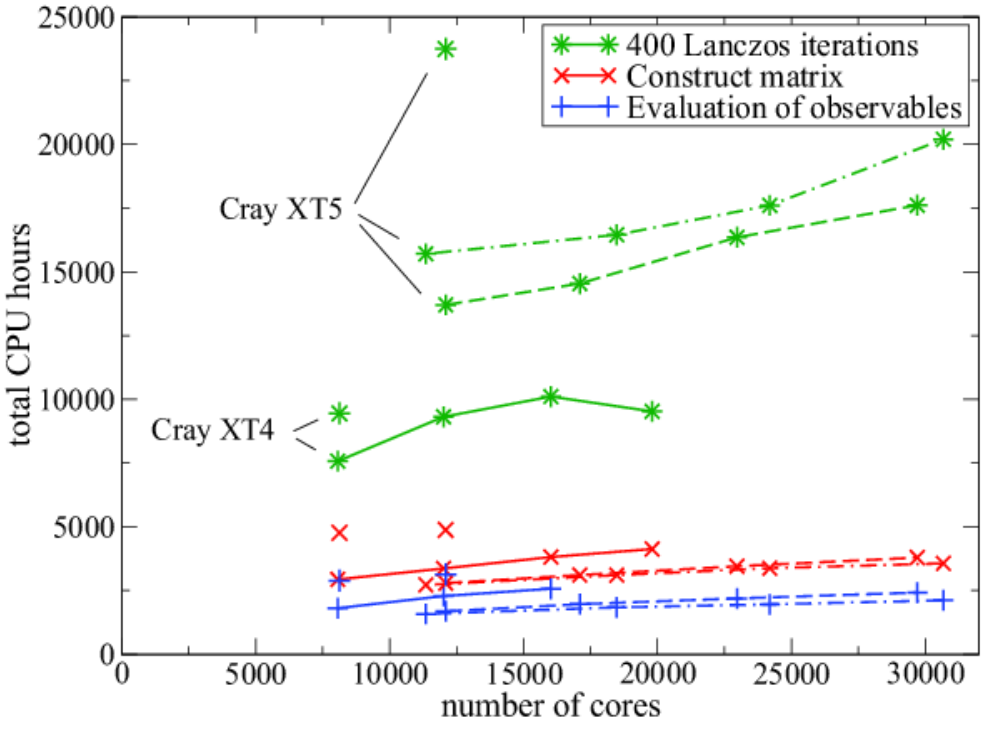
dotted: pure MPI
4 MPI PE's per node

dimension $49 \cdot 10^6$
nonzero m.e. $74 \cdot 10^9$
memory for matrix: 600 GB

- Pure MPI more efficient than hybrid MPI/OpenMP for this case

Hybrid MPI/OpenMP more efficient as problem size grows

^{14}N, $N_{\max} = 8$, 2-body interactions, on Franklin (XT4) and Jaguar (XT5)



solid: hybrid OMP/MPI
1 MPI PE per node
with 4 threads (XT4)

dashed: hybrid OMP/MPI
1 MPI PE per NUMA node
with 6 threads (XT5)

dot-dashed: hybrid OMP/MPI
1 MPI PE per compute node
with 12 threads (XT5)

dimension $1.1 \cdot 10^9$
nonzero m.e. $1 \cdot 10^{12}$
memory for matrix: 8 TB

For comparison: symbols at 8,128 (XT4) and at 12,090 (XT5) cores
pure MPI with 1 MPI PE per core

Total-J Calculation

- Want to compute low energy state of $\widehat{H}$ with a prescribed total angular momentum
- When there is no external field, $[\widehat{H}, \widehat{J}^2] = 0$. Thus $\widehat{H}$ and $\widehat{J}^2$ are simultaneously diagonalizable
- Find Z such that

$$\widehat{J}^2 Z = Z\Omega,$$

where

$$Z^T Z = I_m, \quad \texttt{eig}(\Omega) = \lambda$$

with a known λ (J-basis or null space calculation)

- Form $S = Z^T \widehat{H} Z$
- Solve $SG = G\Lambda$ iteratively
- Form $Y = ZG$

J-basis (null space) Calculation

- Enumerate many-body states (MBS) in groups according to reduced set of quantum numbers associated with single particles states
- MBS within each group is invariant under $\widehat{J}^2$.

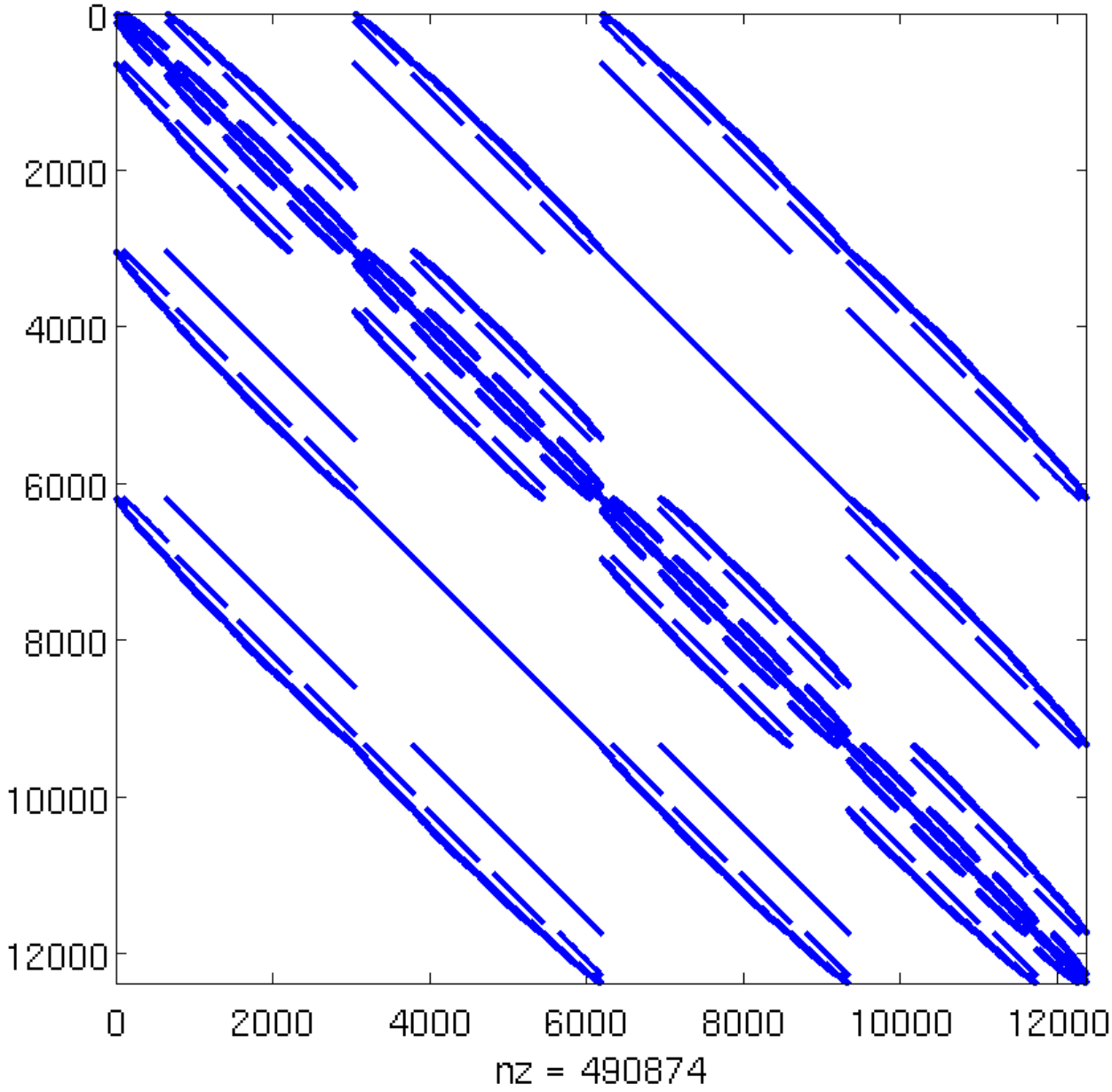
$$\widehat{J}^2 = \begin{pmatrix} \widehat{J}_1^2 & & & \\ & \widehat{J}_2^2 & & \\ & & \ddots & \\ & & & \widehat{J}_{n_g}^2 \end{pmatrix}$$

- Problem reduces to computing

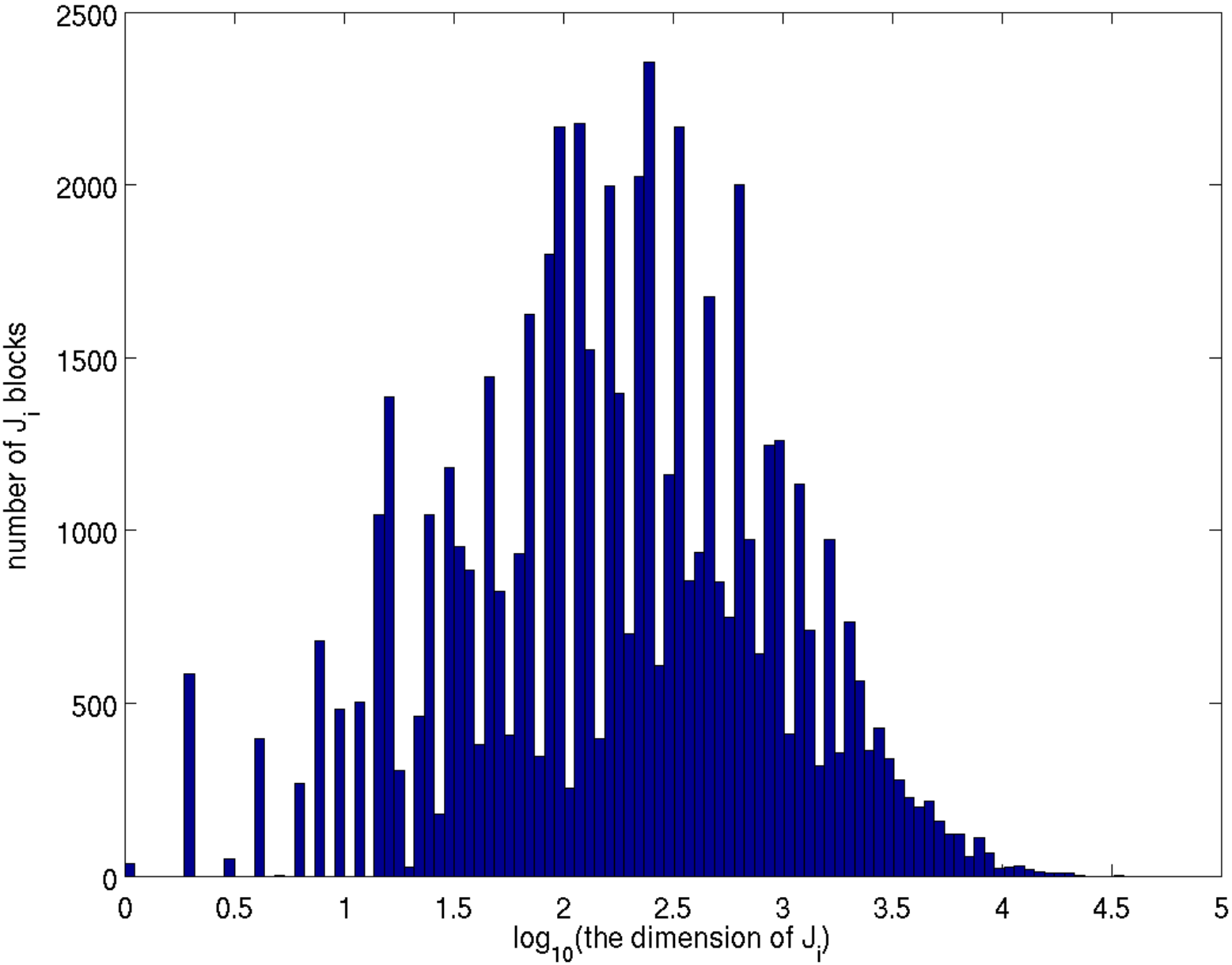
$$\widehat{J}_i^2 Z_i = Z_i \Omega_i, \quad \texttt{eig}(\Omega_i) = \lambda$$

- $\widehat{J}_i^2$ is very sparse
- The dimension of $\widehat{J}_i^2$ is known, but can vary significantly from one i to another (1 to tens of thousands)
- rank(Z_i) depends on λ (10% $\sim$ 30% of the dimension of $\widehat{J}_i^2$)

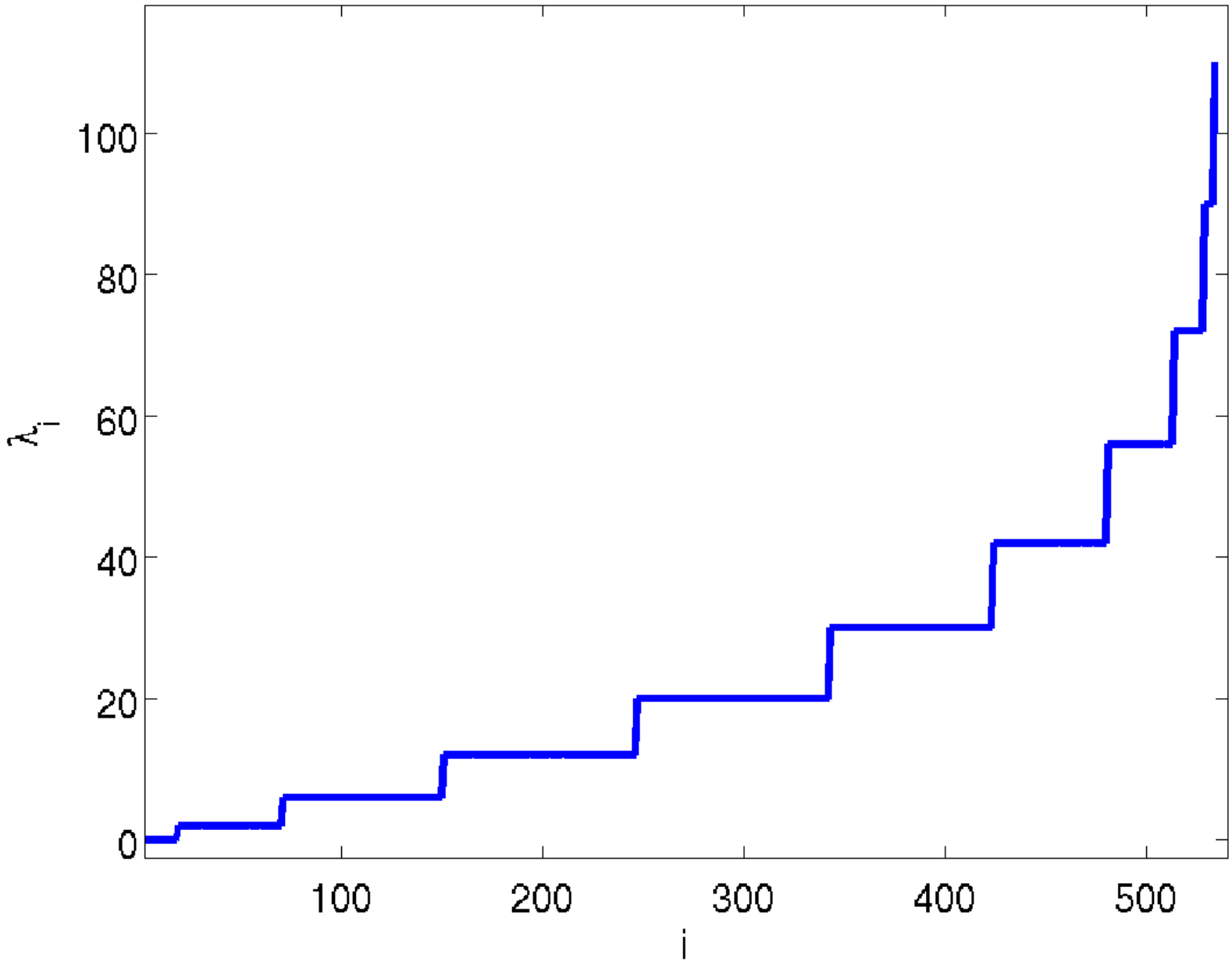
Sparsity of $\widehat{J}_i^2$



The dimensions of $\widehat{J}_i^2$'s



The spectrum of a $\widehat{J}_i^2$



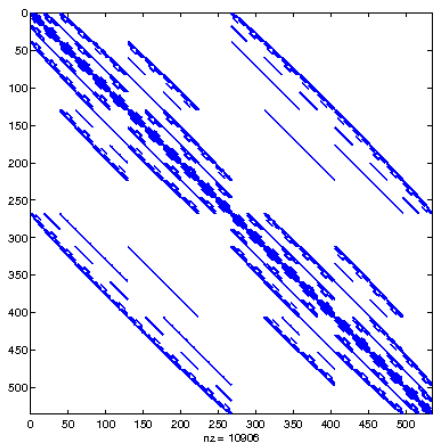
Methods for Computing Z_i

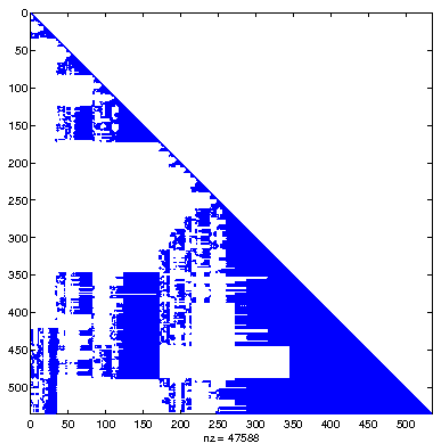
- Rank-revealing QR

$$(\widehat{J}_i^2 - \lambda I)P = QR$$

  can use randomized algorithms (does not require pivoting)

- Shift-invert Lanczos. Apply Lanczos (or subspace iteration) to $(\widehat{J}_i^2 - \sigma I)^{-1}$, where σ is close to λ.





- Polynomial accelerated subspace iteration (PASI)

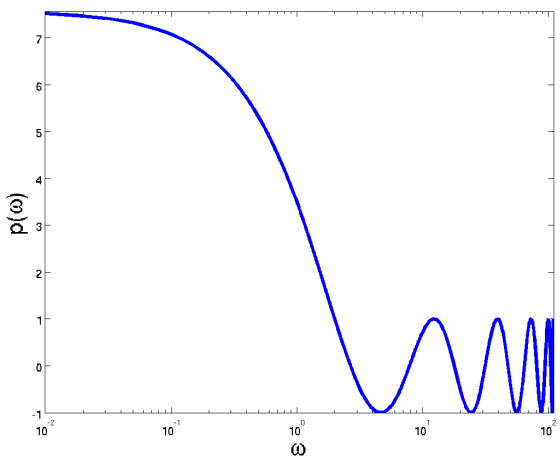
PASI

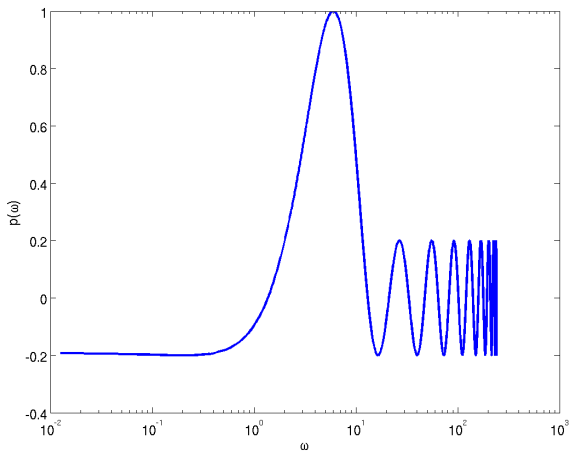
Apply subspace iteration to $p(\widehat{H})$

1. Pick an initial guess to Z_i (V such that $V^T V = I$);
2. $W \leftarrow p(\widehat{H})V$;
3. $[V, R] = qr(W)$;
4. go back to Step 2 if convergence not reached

The choices of polynomials:

- Chebyshev if λ is the smallest eigenvalue of $\widehat{J}_i^2$.
- Bandpass polynomial otherwise

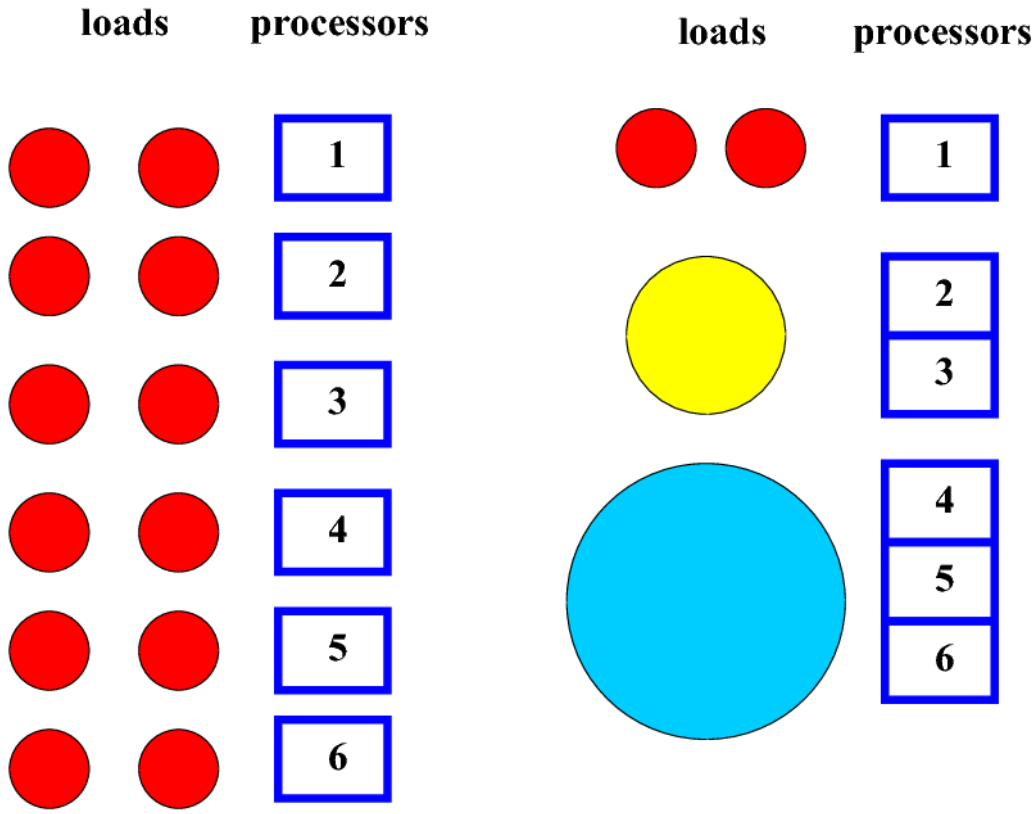




Parallelization

Two inherently conflicting objectives:

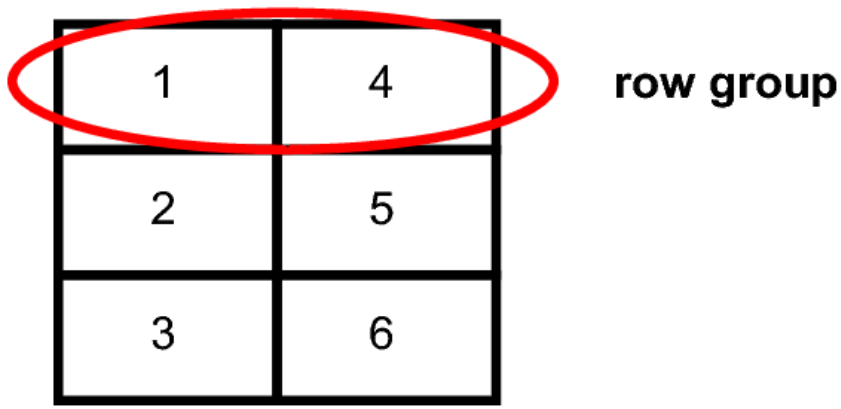
- Limit the granularity of the parallelism.
- Limit the amount of communication overhead.



Heuristic

Classify $\widehat{J}_i^2$ into small, medium, large blocks based on dimension, load estimation, ratio of flops over communication volume

- Small blocks are assigned to single processors. An sequential algorithm is used to find the desired invariant subspace. (No communication)
- Medium size blocks are mapped to a row group. The invariant subspace (Moderate amount of communication) is computed in parallel by processors within the same group
- Large size blocks ("outliers") are tackled by all processors simultaneously. (Lots of communication)

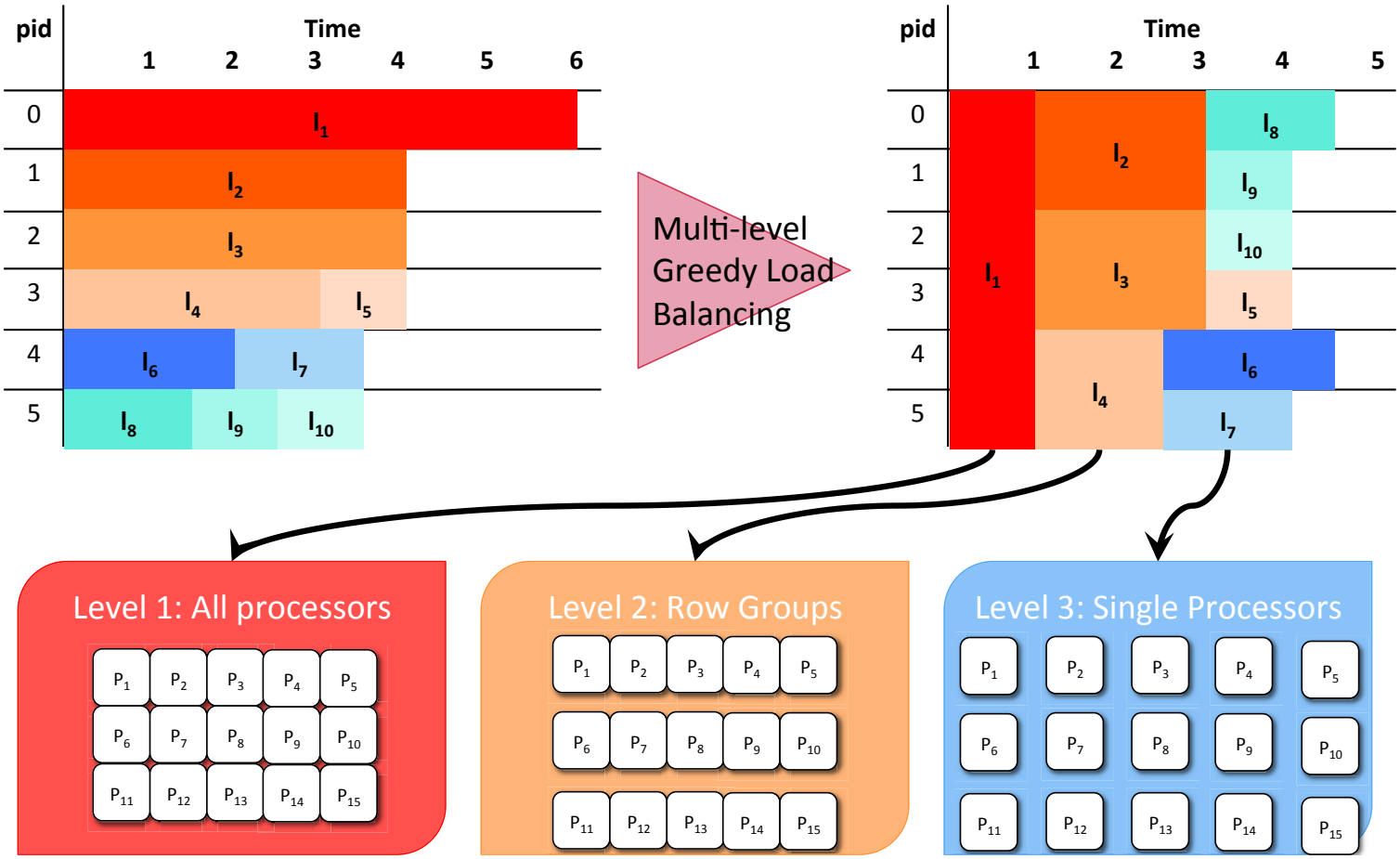


Greedy Load Balance

Once $\widehat{J}_i^2$'s have been classified, the total amount of load (including communication cost) is fixed.

1. Compute the idea average load w per row group for medium-sized blocks;
2. Distribute medium-sized blocks (sorted in descending order in terms of their loads) in a cyclic fashion over n_r groups. If assigning a particular $\widehat{J}_i^2$ to a processor group p_r results in load overflow, skip p_r and try to assign $\widehat{J}_i^2$ to the next available group without exceeding the w limit. If $\widehat{J}_i^2$ cannot be assigned to any row group, set it aside for later assignment;
3. If there exits some medium-size $\widehat{J}_i^2$ blocks that cannot be assigned to any of the row groups
   - raise w slightly and repeat step 2;
   - or, assign $\widehat{J}_i^2$ with the largest load to the processor group with the least amount of filled load ...
4. Distribute the small blocks to reduce load variation.

Load Balancing Null Space Computations



Performance on Real Problems

nucleus	$(N_{\max}, J)$	n_g	k	n
^{6}Li	(10,0)	7.8×10^4	3.3×10^5	9.7×10^6
	(10,1)	7.8×10^4	9.4×10^5	9.7×10^6
	(12,0)	2.5×10^5	1.4×10^6	4.9×10^7
	(12,1)	2.5×10^5	3.9×10^6	4.9×10^7
	(12,12)	2.5×10^5	2.8×10^5	4.9×10^7
^{12}C	(4,0)	5.8×10^3	5.5×10^4	1.1×10^6
	(6,0)	5.6×10^4	1.3×10^6	3.3×10^7
	(6,2)	5.6×10^4	3.5×10^6	3.3×10^7
	(6,12)	5.6×10^4	3.1×10^4	3.3×10^7

Load Balance Performance

Table: The minimum, average and maximum wall clock time consumed by PASI when the greedy load balancing algorithm is used.

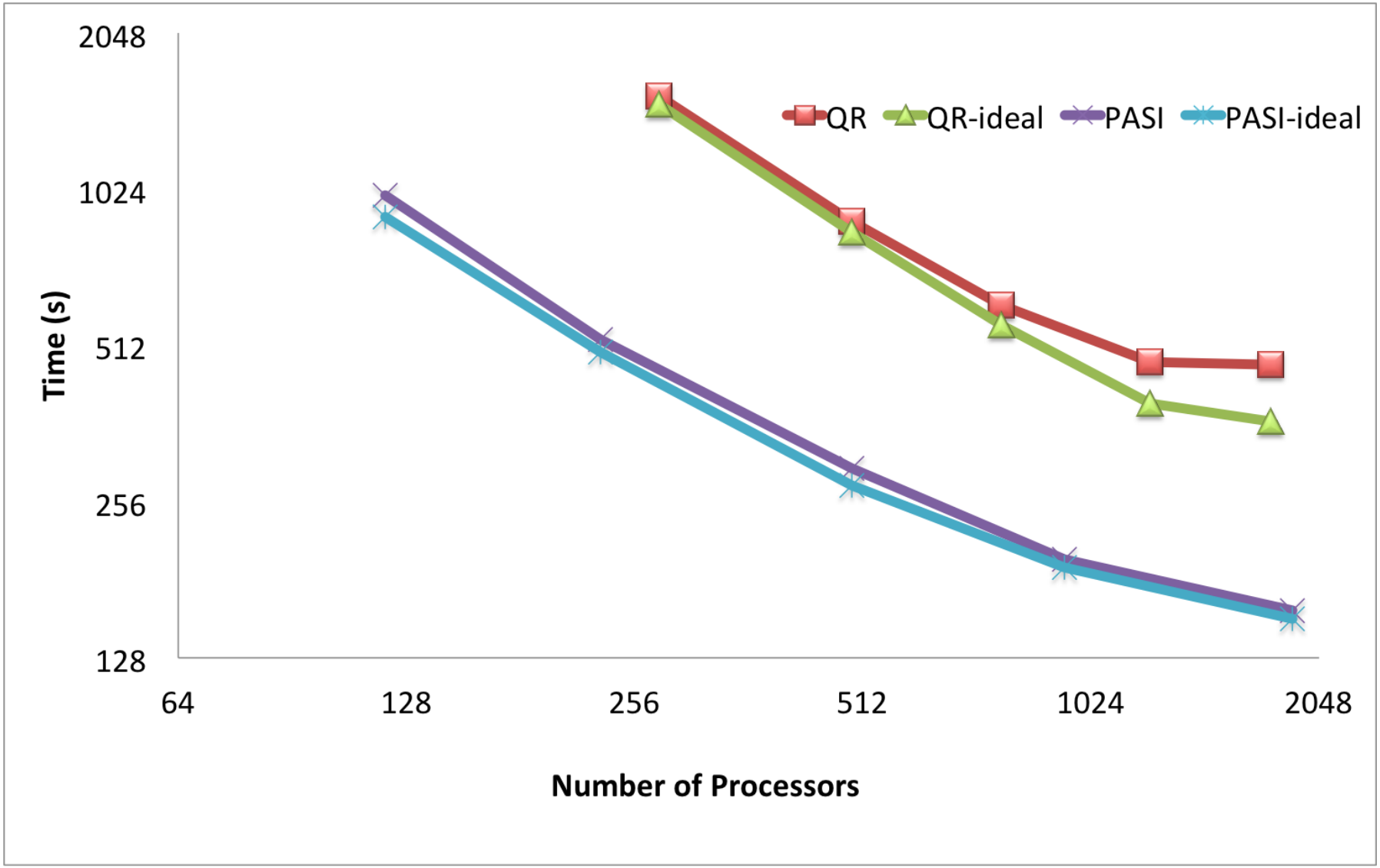
nucleus	$N_{\max}$	wt_{min}	wt_{avg}	wt_{max}	%comm
^{6}Li	8	0.95	1.10	1.35	17%
	10	9.5	12.0	13.0	21%
	12	126	129	132	39%
^{12}C	4	3.3	4.0	5.2	20%
	6	848	902	995	20%

$$wt_{avg} = \left[\sum_{j=1}^{n_p} \sum_{i=1}^{n_g} wt_j(\widehat{J}_i^2) \right] / n_p$$

Performance Improvement Over Previous Implementation

nucleus	$N_{\max}$	alg	n_p	cyclic	greedy	ideal
^{6}Li	10	PASI	120	12.9	13.0	12.0
^{6}Li	12	PASI	120	131	132	129
^{12}C	4	PASI	120	6.1	5.2	4.0
^{12}C	6	PASI	120	1015	995	902
^{12}C	6	PASI	496	608	295	275
^{6}Li	10	QR	120	24.1	17.8	14.7
^{6}Li	12	QR	120	233	193	176
^{12}C	4	QR	120	18.7	17.0	15.5
^{12}C	6	QR	496	1220	900	860

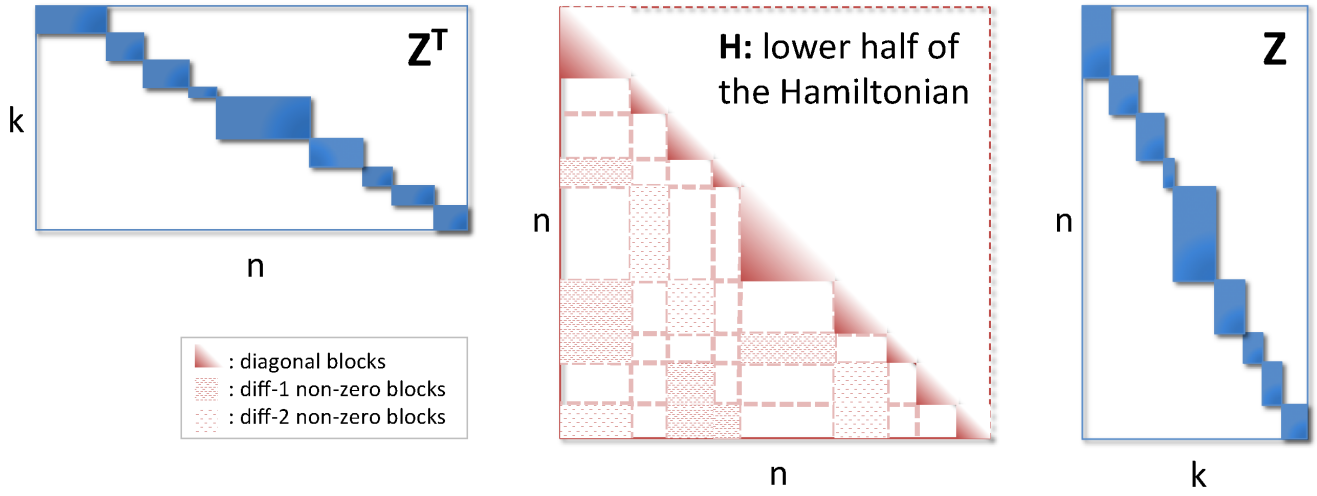
Parallel Scalability (Strong scaling)



Comparison of QR and PASI

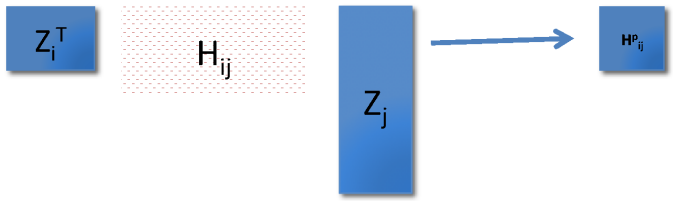
nucleus	$(N_{\max}, J)$	QR	PASI	n_p
^{6}Li	(10, 0)	17.8	13.0	120
^{6}Li	(10, 1)	17.8	34.9	120
^{6}Li	(12, 0)	193	132	120
^{6}Li	(12, 1)	195	464	120
^{6}Li	(12, 12)	140	95	496
^{12}C	(6, 0)	900	295	496
^{12}C	(6, 1)	890	> 1800	496
^{12}C	(6, 12)	840	105	496

Subspace Projection of the Hamiltonian

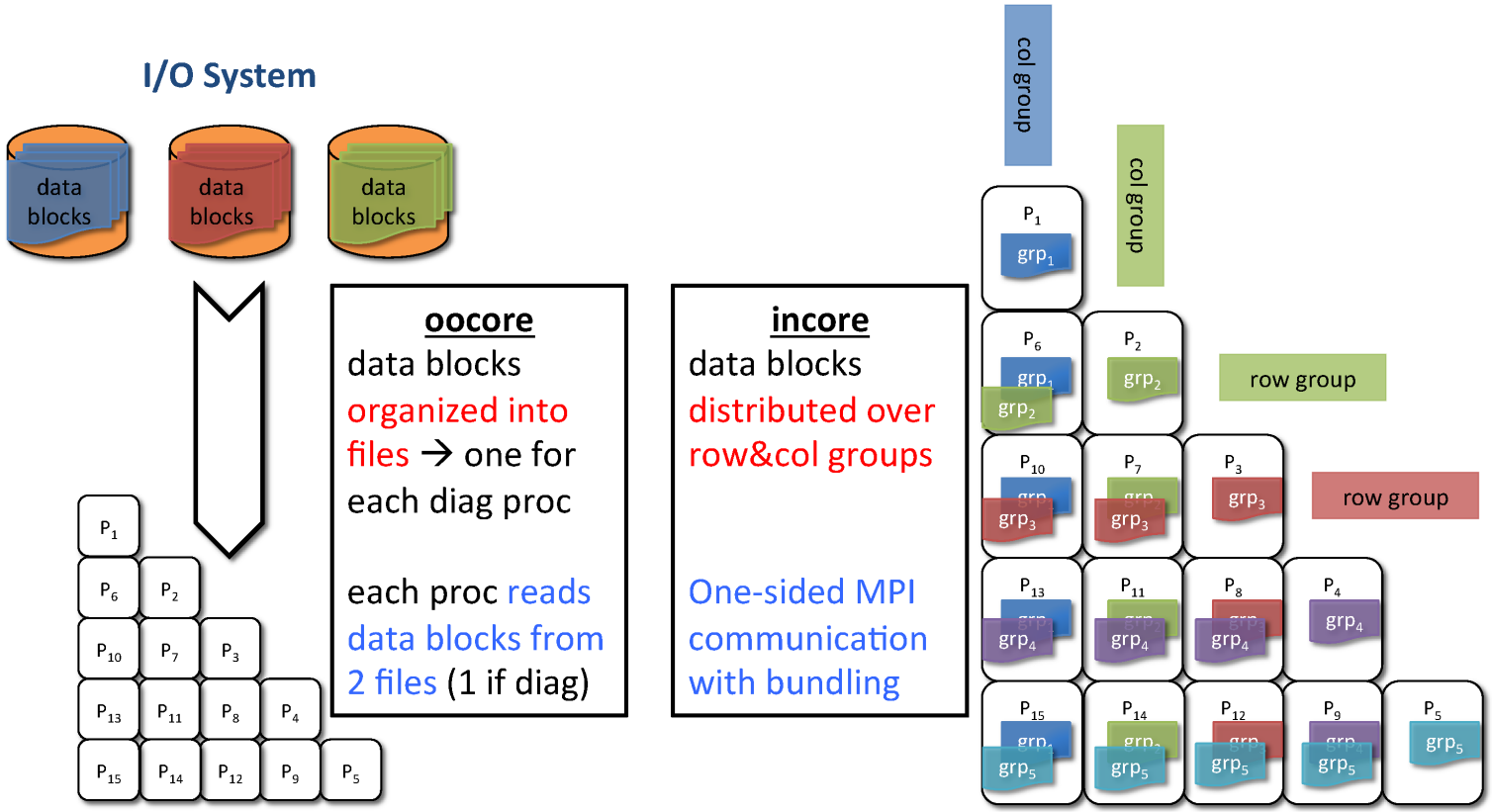


Each non-zero H_{ij} block defines a task:

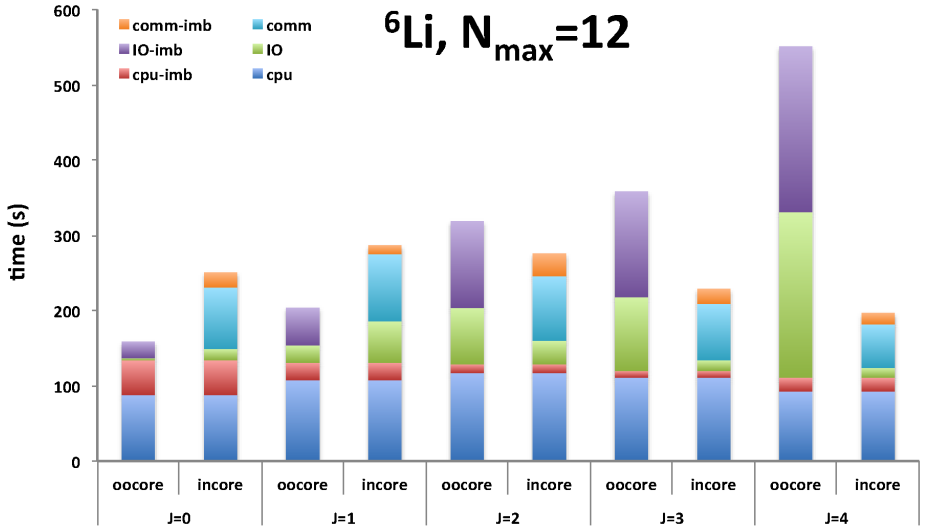
1. construct H_{ij}
2. bring the data blocks, Z_i and Z_j
3. project block by block: $Z_i^T (H_{ij} Z_j)$



Out-of-core vs. In-Core Approaches

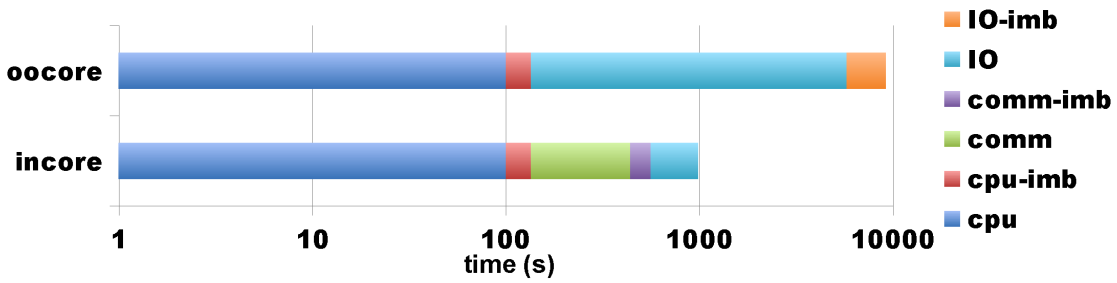


Out-of-core vs. In-Core Performance



^{6}Li, N_{max}=14
J=3

Much larger problem, bigger gain!



Challenges

- Numerical method for solving large-scale eigenvalue problem is a well studies subject. But large-scale parallel implementation for nuclear CI calculation is not trivial.
- Optimizing the performance of individual pieces of the code (SpMV, orthogonalization etc.) is important. Optimizing the global performance of the code is even more important and difficult. A decision (data structure, data distribution, load balance) made for one part of the code often affects the performance of another part of the code.
- Things will become more complicated for many core machines with hybrid OpenMP/MPI implementation. How do we address this additional level of complexity?
- The current implementation is contrained by memory usage. Alternatives:
  - Out-of-core
  - Recompute matrix elements on the fly (when a MATVEC is performed)

References

1. P. Sternberg, C. Yang, E. G. Ng, P. Maris, J. P. Vary, M. Sosonkina, and H. V. Le. Accelerating Configuration Interaction Calculations for Nuclear Structure. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Austin, Texas, November 15 - 21, 2008).
2. J. P. Vary, P. Maris, E. Ng, C. Yang and M. Sosonkina. Ab initio nuclear structure – the large sparse matrix eigenvalue problem. *Journal of Physics: Conference Series*, 180:012083, 2009.
3. P. Maris, M. Sosonkina, J. P. Vary, E. G. Ng and C. Yang. Scaling of ab-initio nuclear physics calculations on multicore computer architectures. *International Conference on Computer Science, ICCS 2010, Procedia Computer Science,* 1, 97 (2010).
4. H. M. Aktulga, C. Yang, E. Ng, P. Maris and J. Vary. Large-scale parallel null space calculation for nuclear configuration interation *To appear in HPCS2011 proceedings*, 2011
5. H. M. Aktulga, C. Yang, E. Ng, P. Maris and J. Vary. On Reducing I/O Overheads in Large-scale Invariant Subspace Projection *Submitted to HPSS2011*, 2011