



# PROGRESS AND APPLICATIONS OF BIGSTICK CI CODE

Plamen Krastev,<sup>1,2</sup> Calvin Johnson<sup>1</sup> and Erich Ormand<sup>2</sup>

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

## Outline:

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

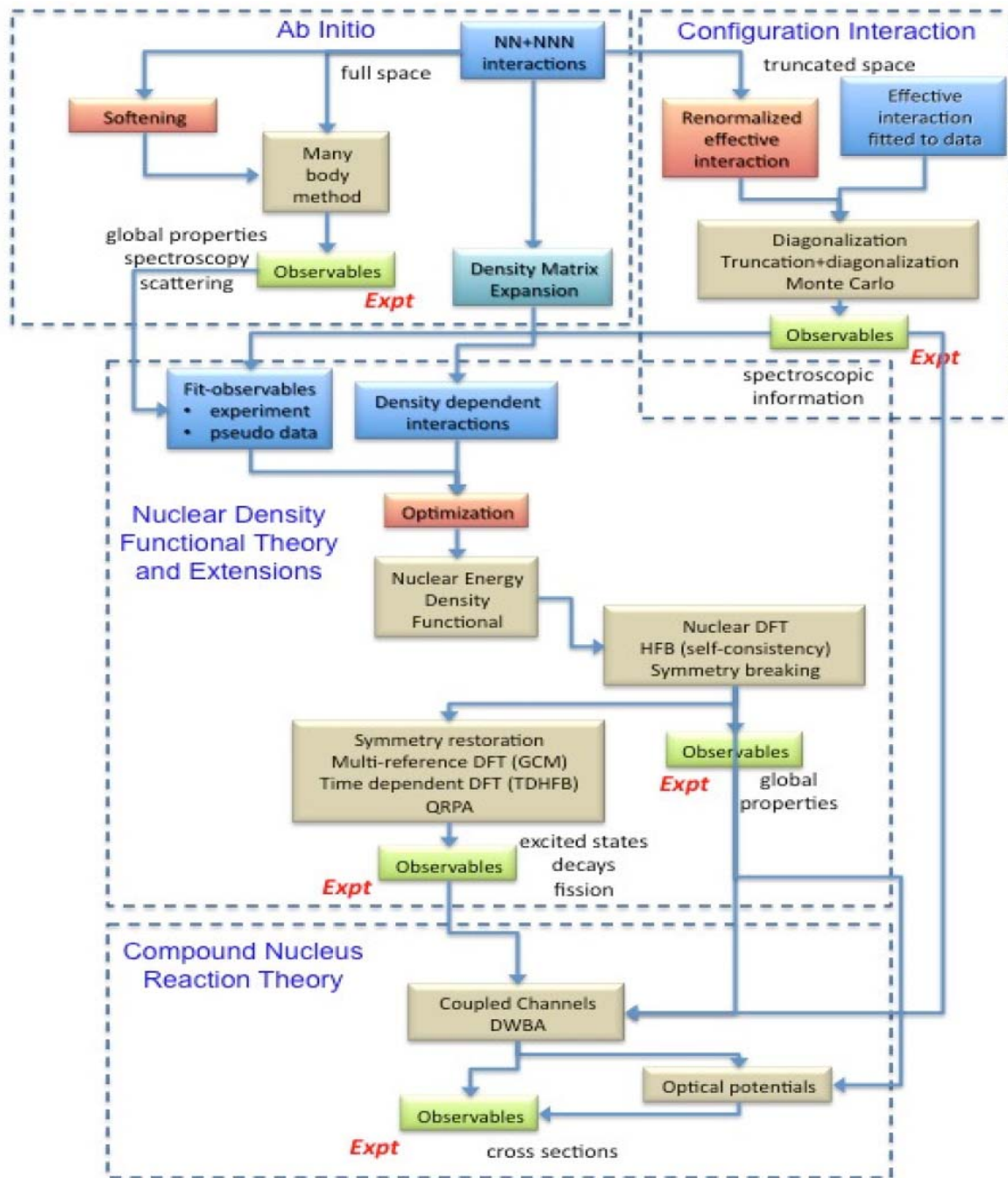
**SciDAC - UNEDF meeting, Lansing MI, June 2010**

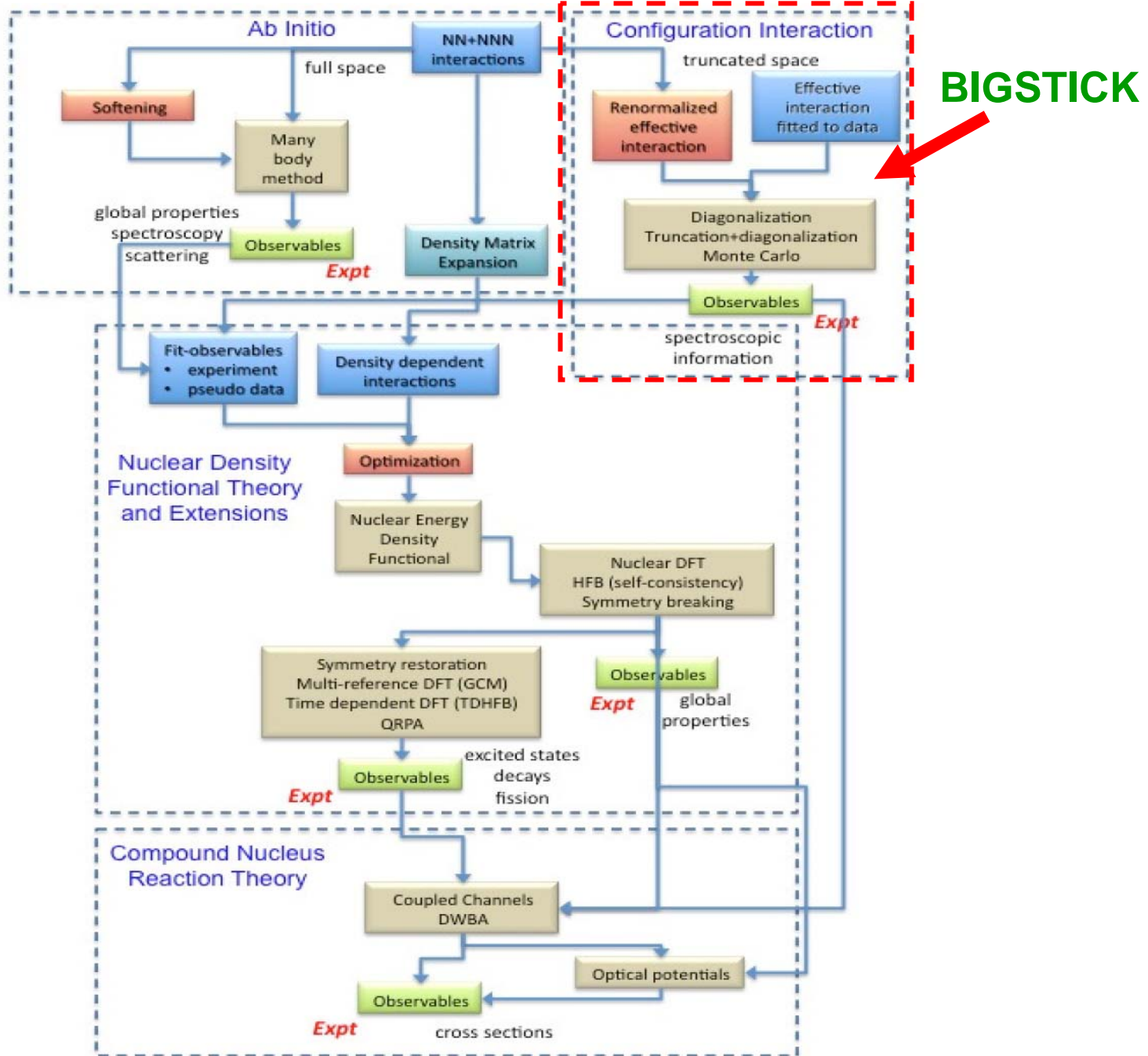


# BIGSTICK:

**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**



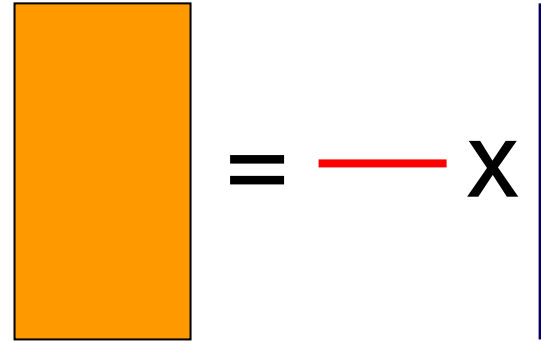


**BIGSTICK**

# Key idea for on-the-fly algorithms:

**Represent an area by its boundary**

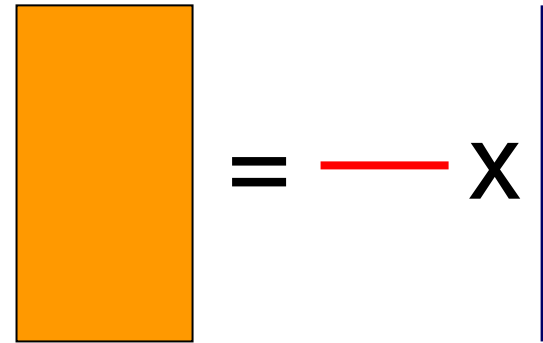
- ❑ **Factorization** of problem
- ❑ **Reduces dramatically memory load**



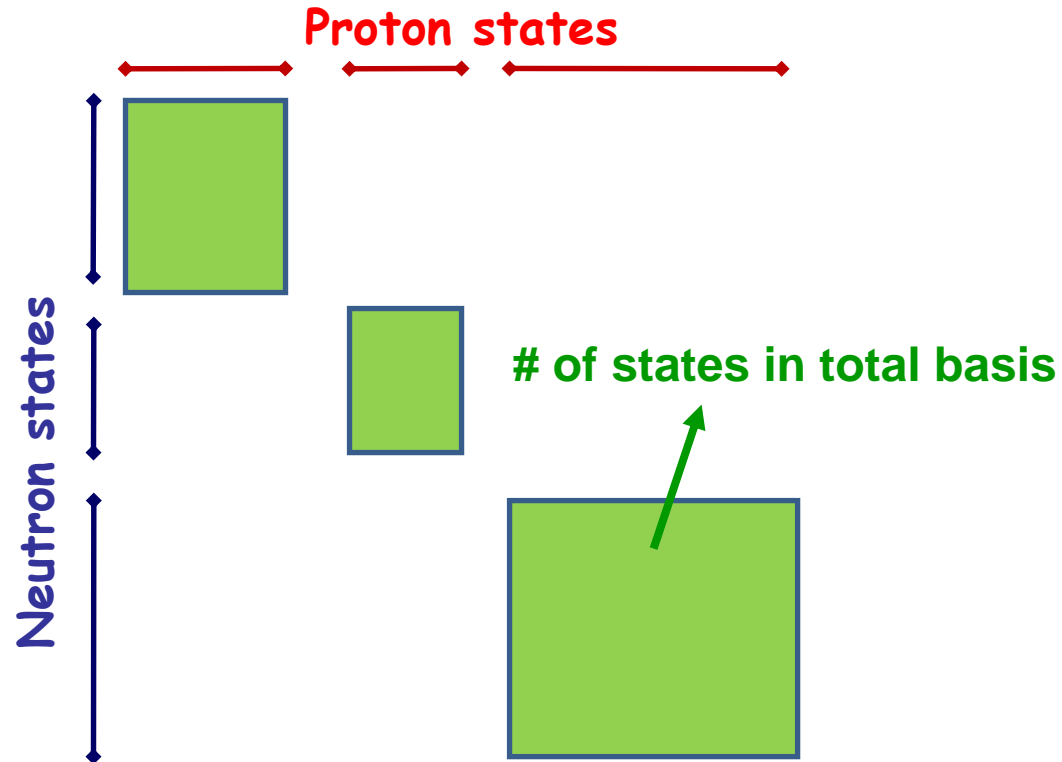
# Key idea for on-the-fly algorithms:

Represent an area by its boundary

- ❑ Factorization of problem
- ❑ Reduces dramatically memory load



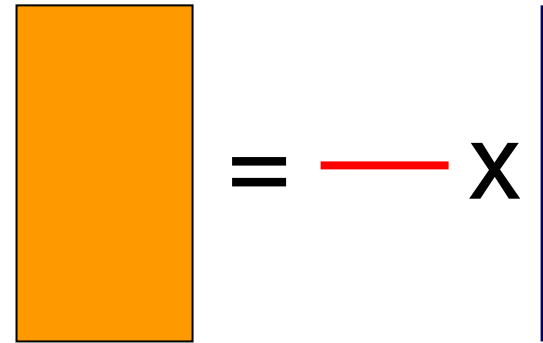
$$|\alpha\rangle = |\alpha_p\rangle \times |\alpha_n\rangle$$



# Key idea for on-the-fly algorithms:

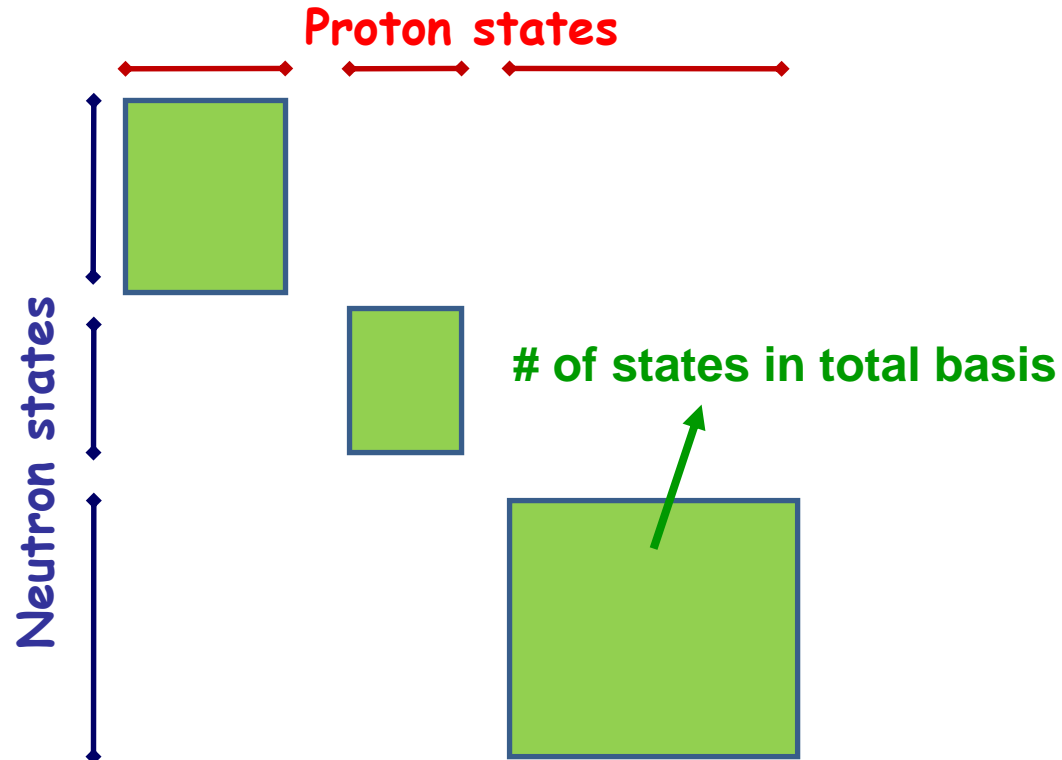
Represent an area by its boundary

- ❑ Factorization of problem
- ❑ Reduces dramatically memory load



$$|\alpha\rangle = |\alpha_p\rangle \times |\alpha_n\rangle$$

Hamiltonian can be factorized similarly



# 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

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



# Why "on-the-fly"?

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

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

**1 Tb requires approximately 1,000 cores (depending on architecture)**

## 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>**

# 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**

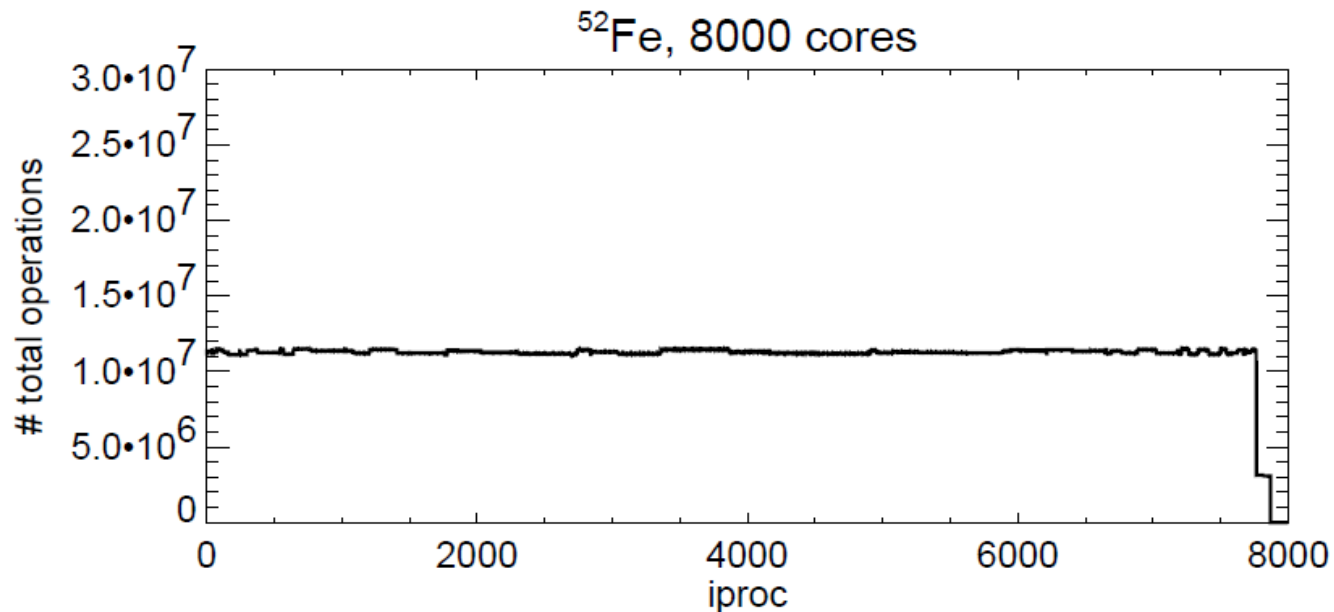
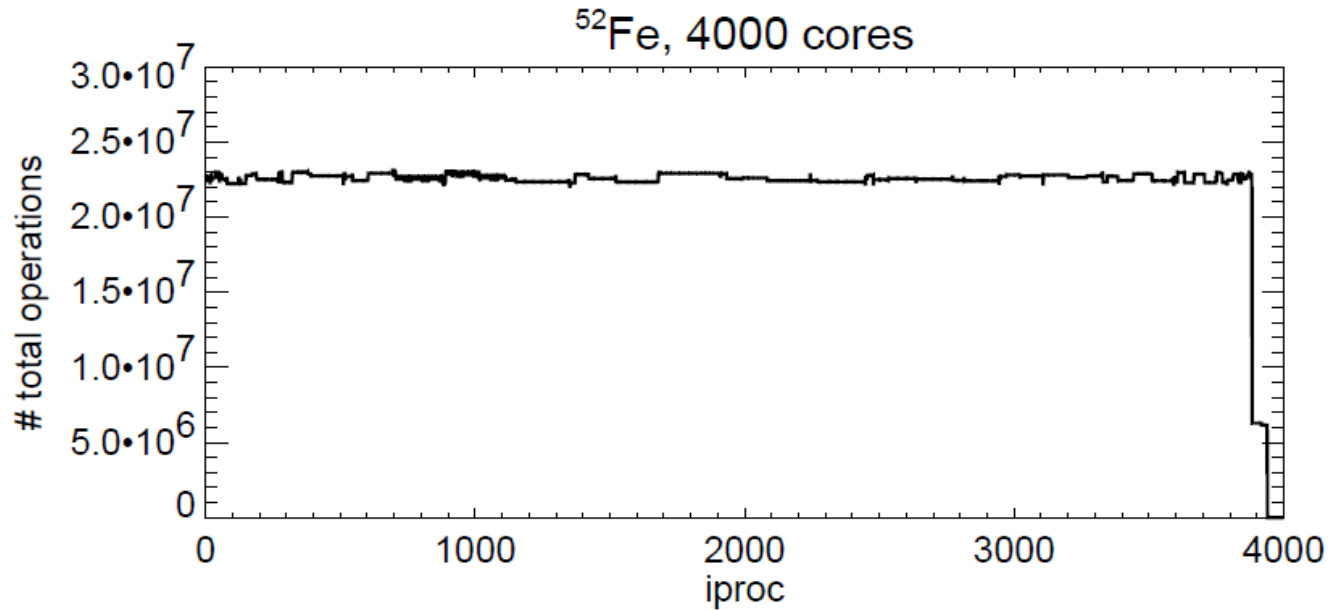
**Vary, Maris, Liu**

**Navratil**

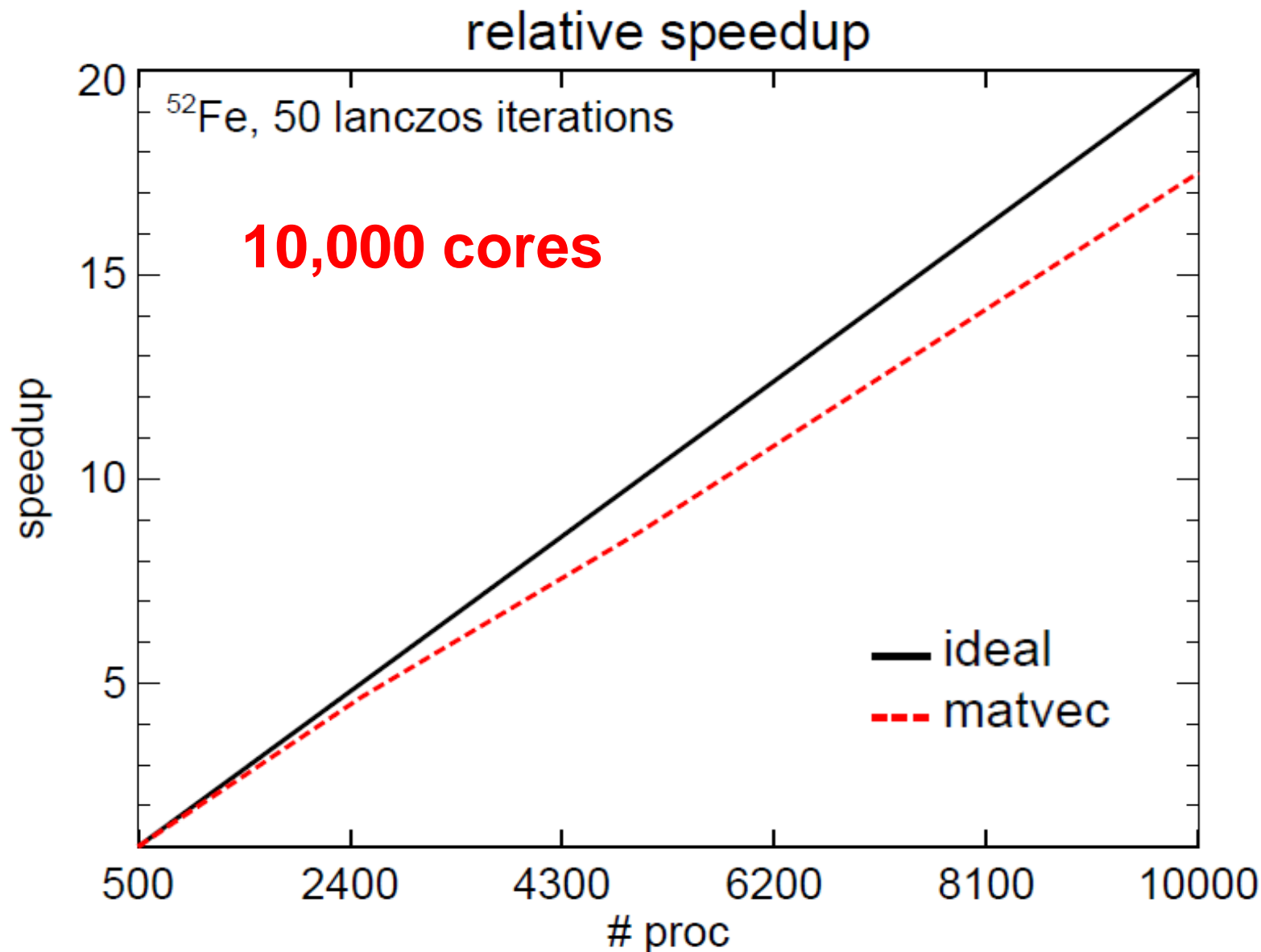
**Horoï**

**Ng, Yang, Calderon**

# Distribution of matrix-vector multiplication:



# (Relative) speedup of matrix-vector multiplication:



# 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  $\dim( V_{in} ) = \dim( V_{out} )$

# 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  $\dim( V_{in} ) = \dim( V_{out} )$

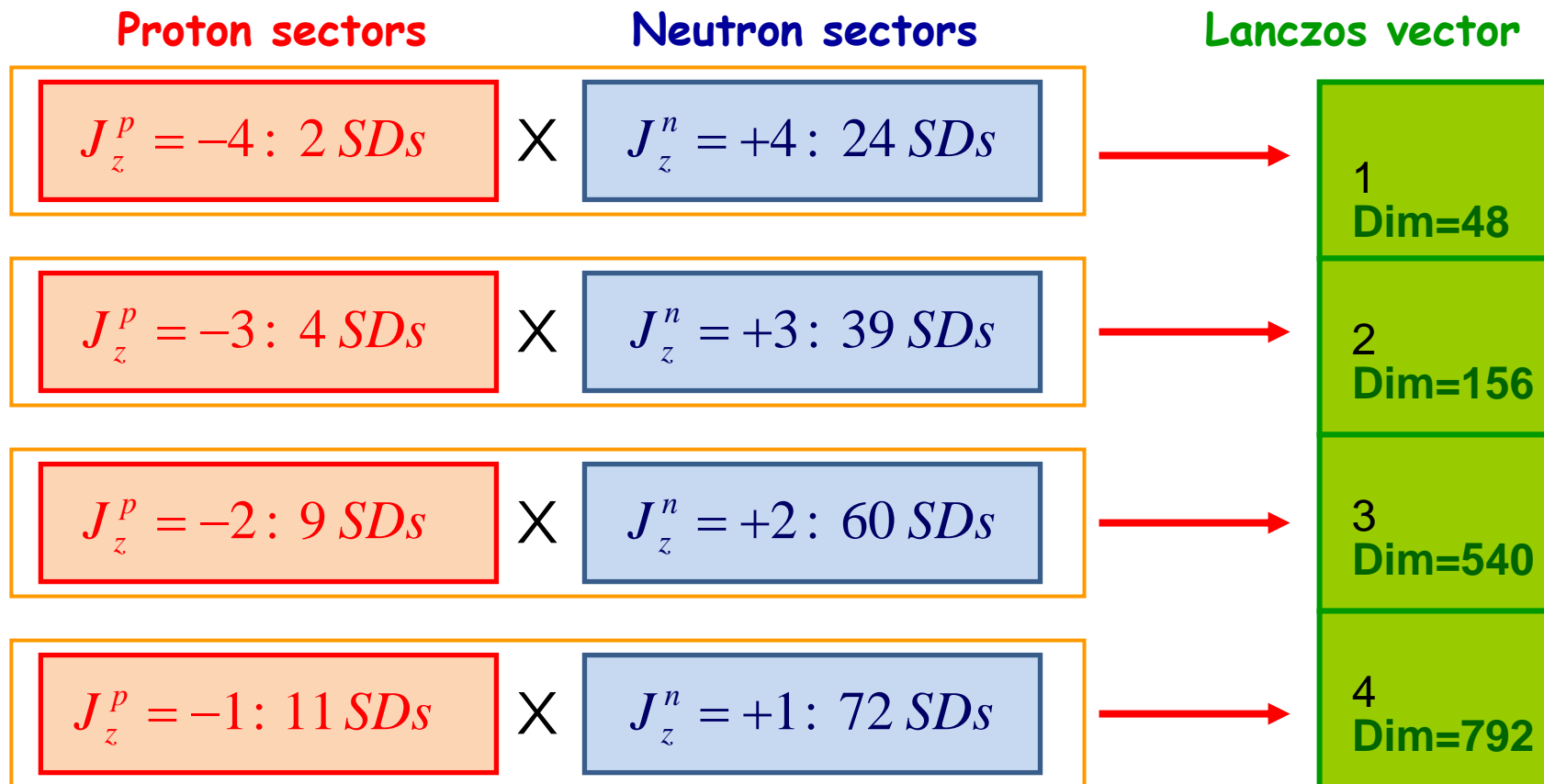
## Practical steps:

- ✓ 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:

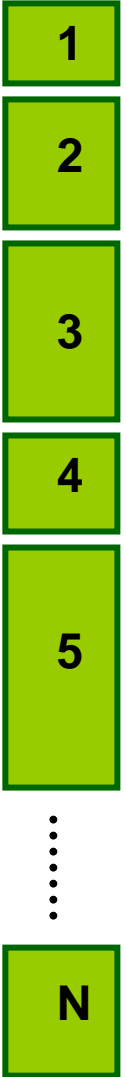
$^{22}\text{Ne}$  in  $sd$ -shell (with inert  $^{16}\text{O}$  core): Total  $J_z = 0$  (first 4 sectors are given)





# Communication groups for Lanczos sectors:

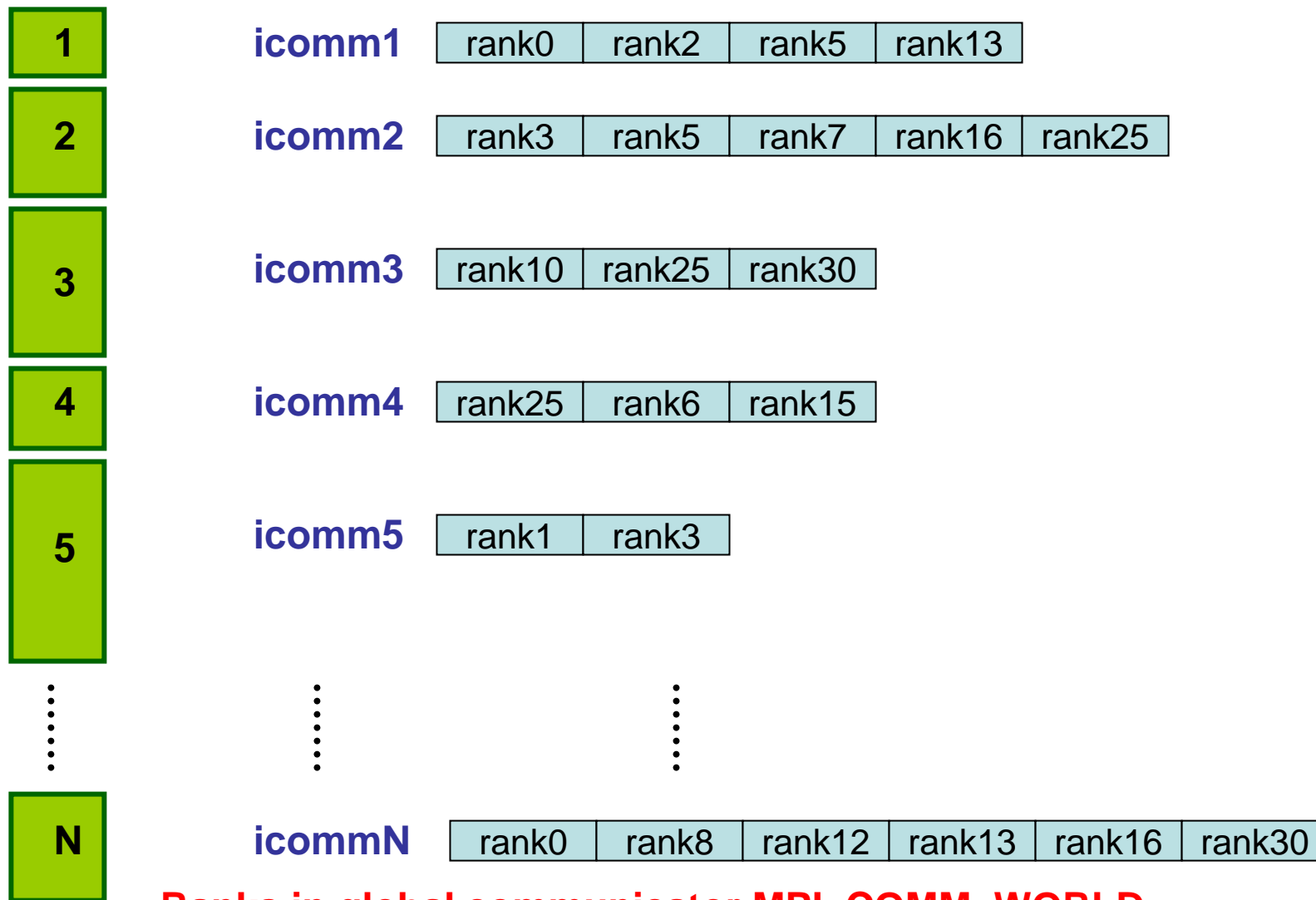
Lanczos vector



# Communication groups for Lanczos sectors:

Lanczos vector

communicator

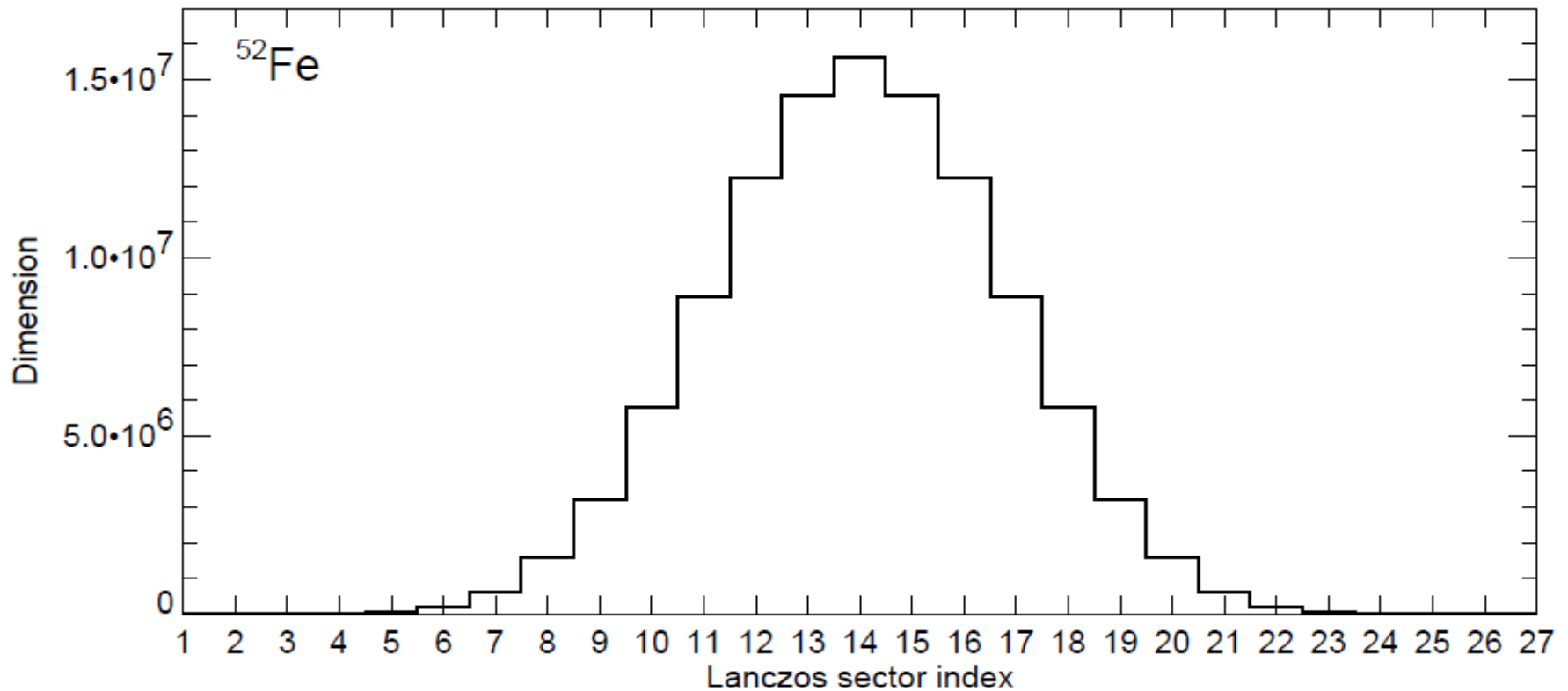


**Ranks in global communicator MPI\_COMM\_WORLD**

## Dimension of Lanczos vector sectors - single shell:

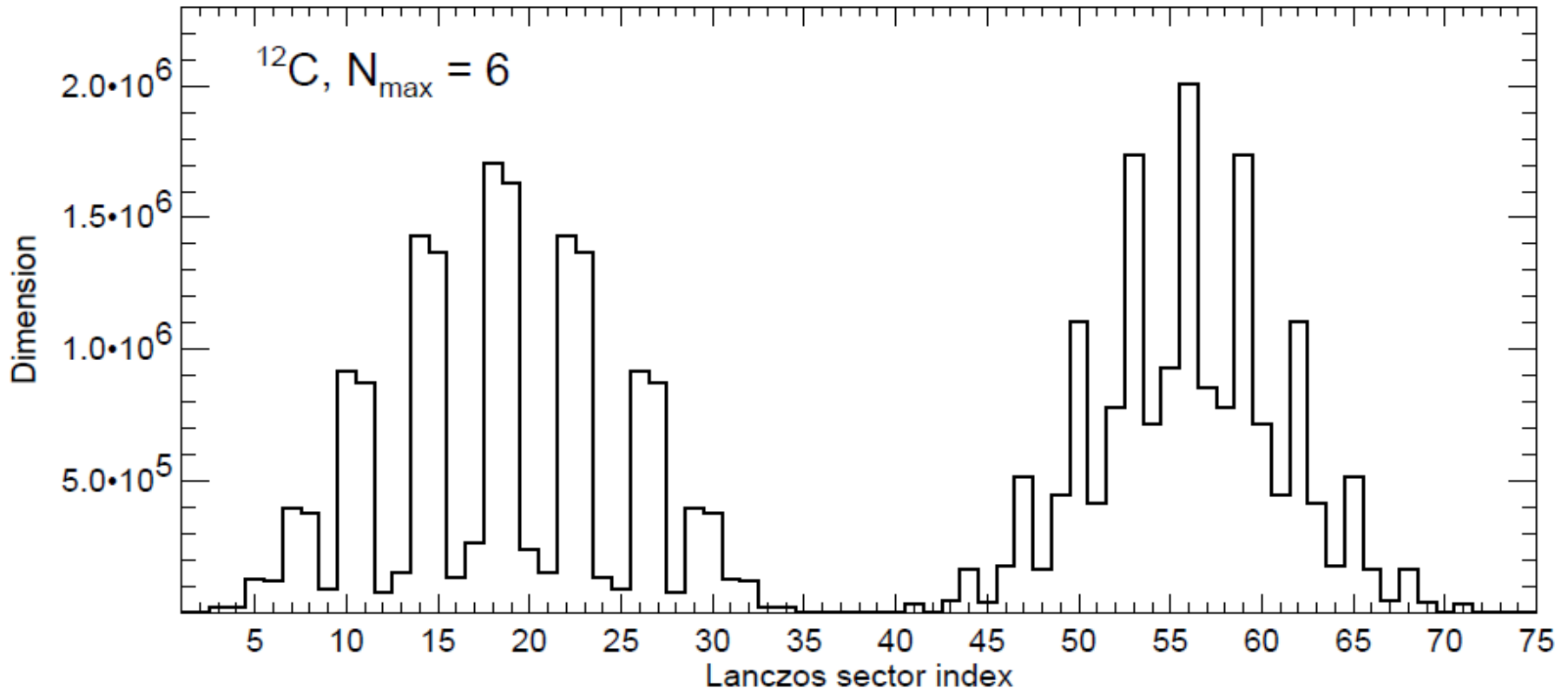
**$^{52}\text{Fe}$ , space – pf, basis dimension = 109,954,620**

**# 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)**

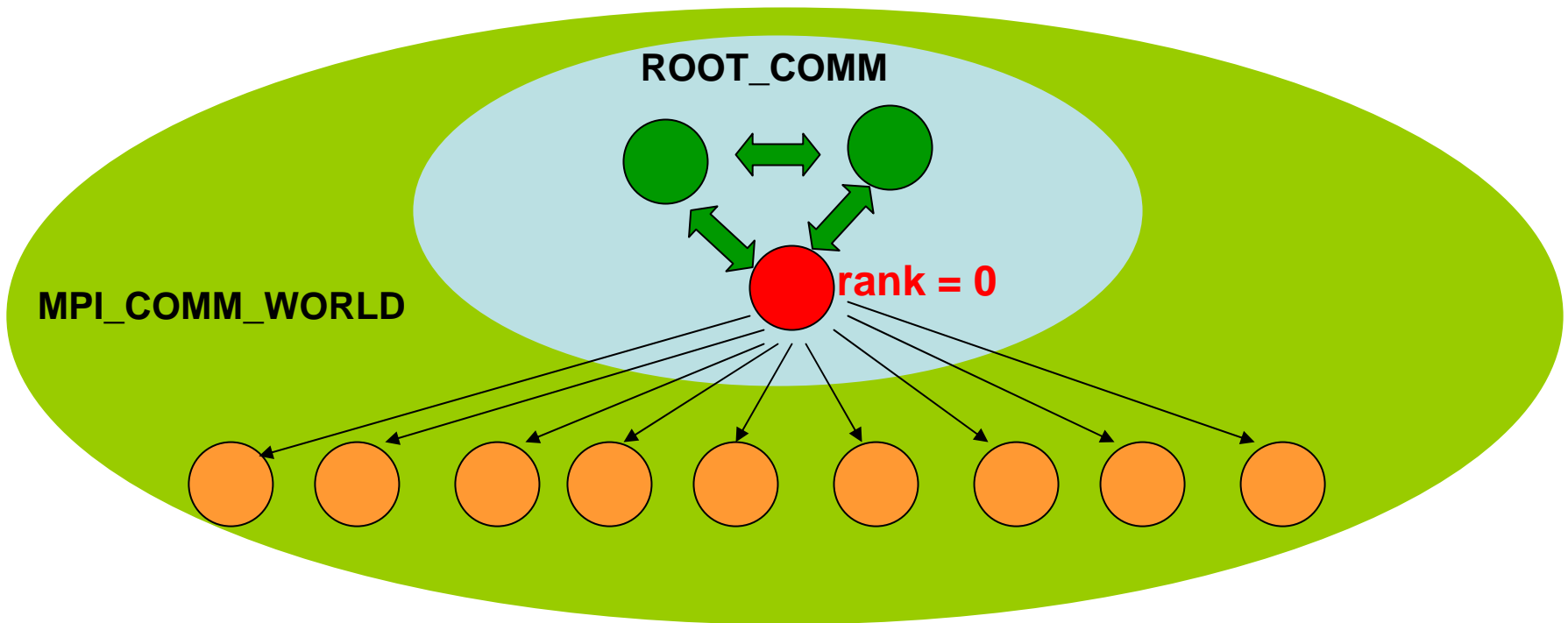


# Dot product in terms of Lanczos sectors:

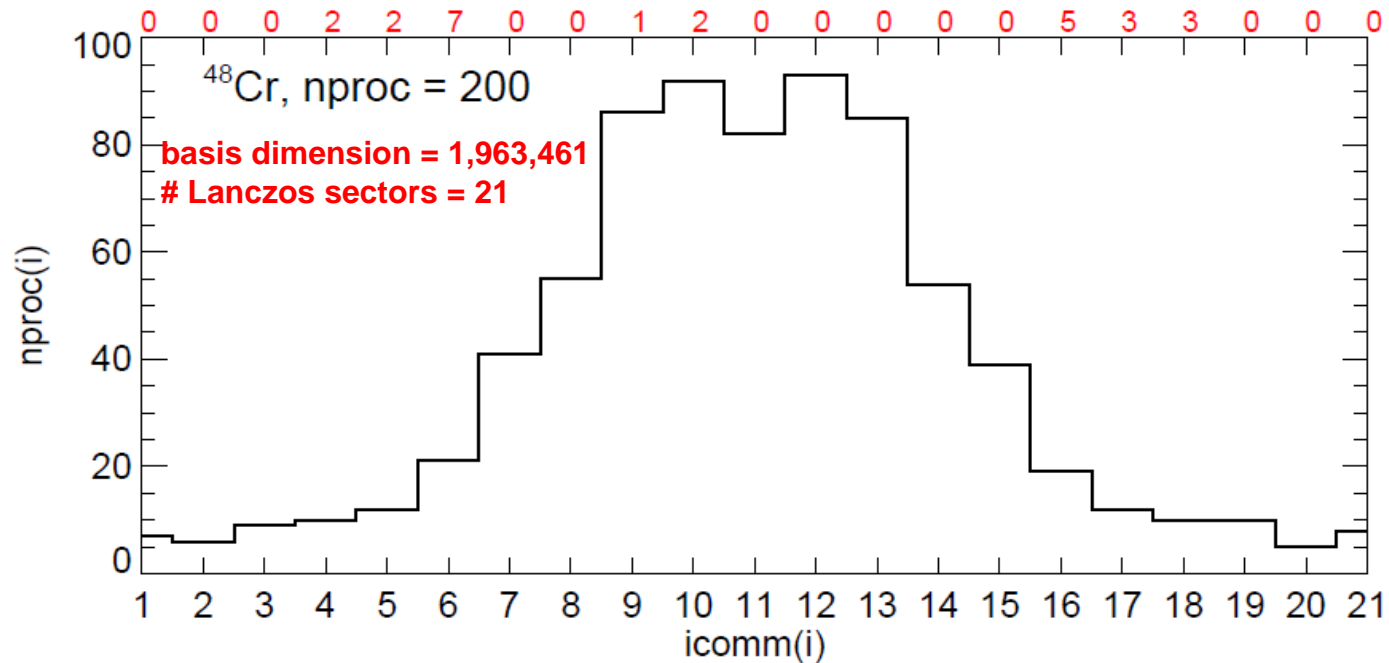
- (i) Each core needs global scalar product
- (ii) Define communicator to handle dot products – consider only masters with unique global ranks – ROOT COMMUNICATOR
- (iii) Calculate dot product over ROOT\_COMM
- (iv) Update dot product on all cores

# 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

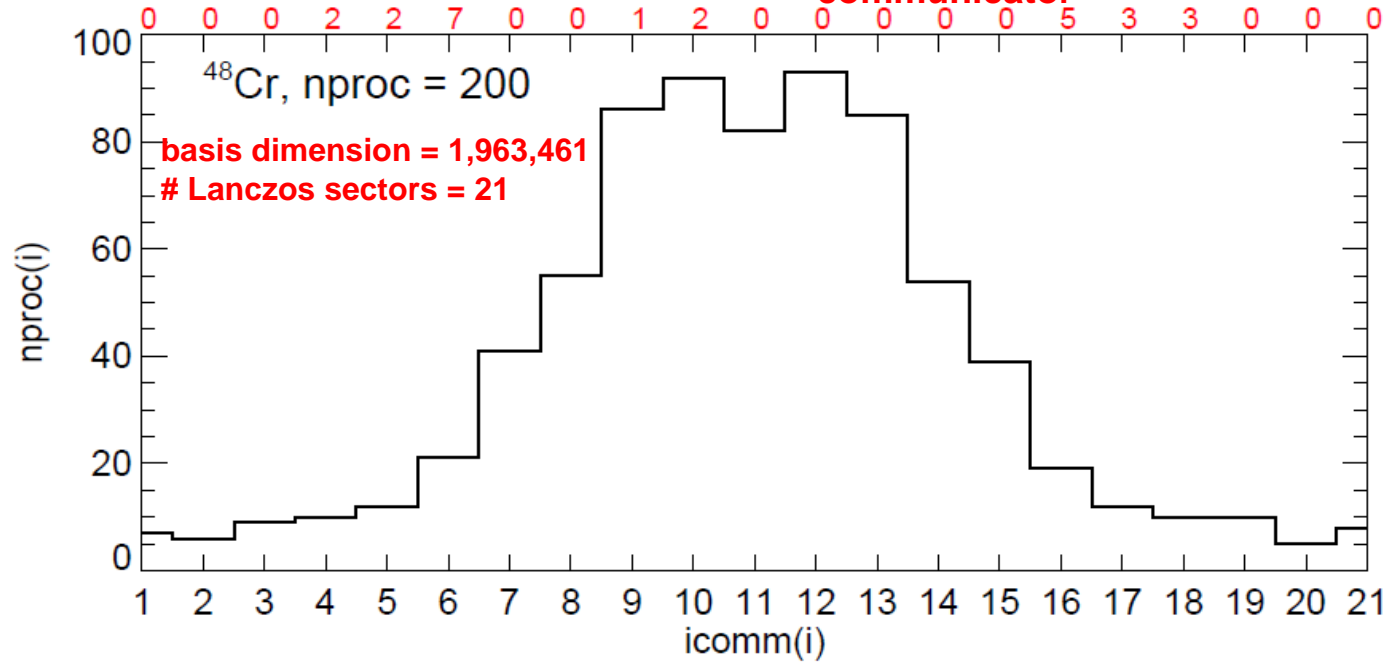


# Dot product, cont'd:



# Dot product, cont'd:

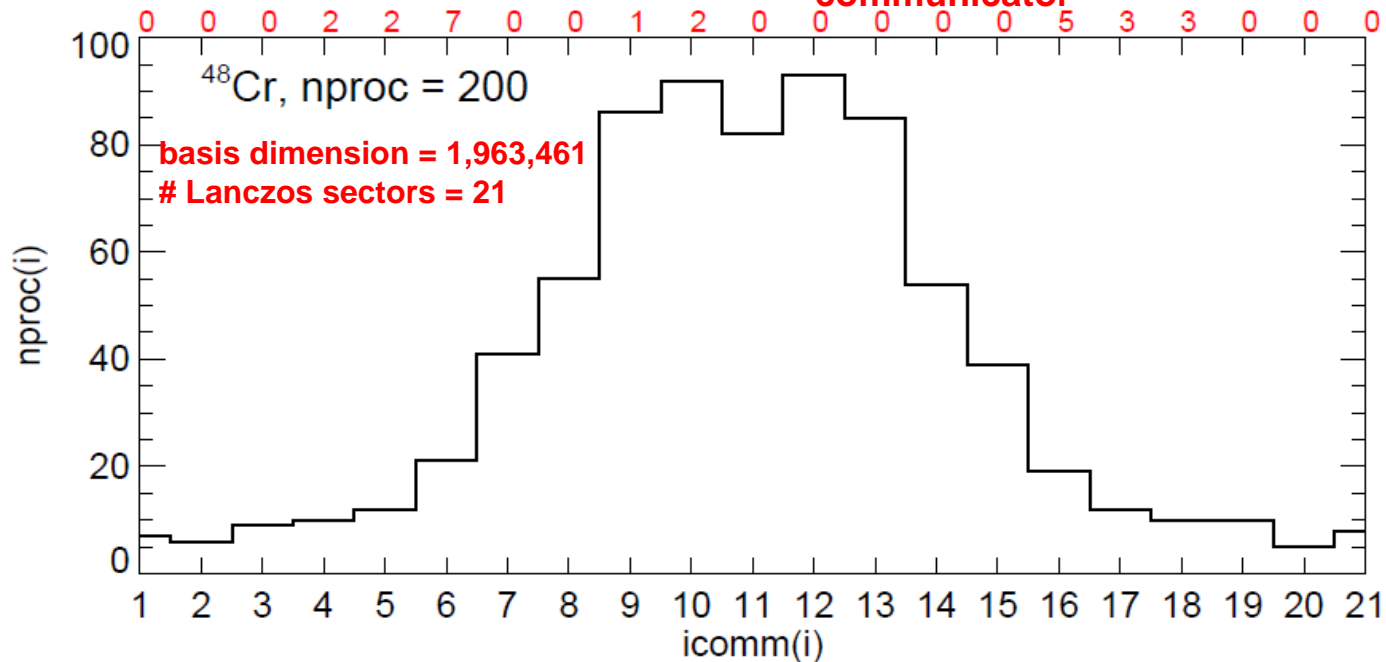
Global ranks of roots for each Lanczos sector communicator





# Dot product, cont'd:

Global ranks of roots for each Lanczos sector communicator



## ROOT\_COMM

## LANCZOS SECTORS

Rank = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank = 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank = 2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank = 3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank = 5	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank = 7	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

# 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**

```
...  
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:

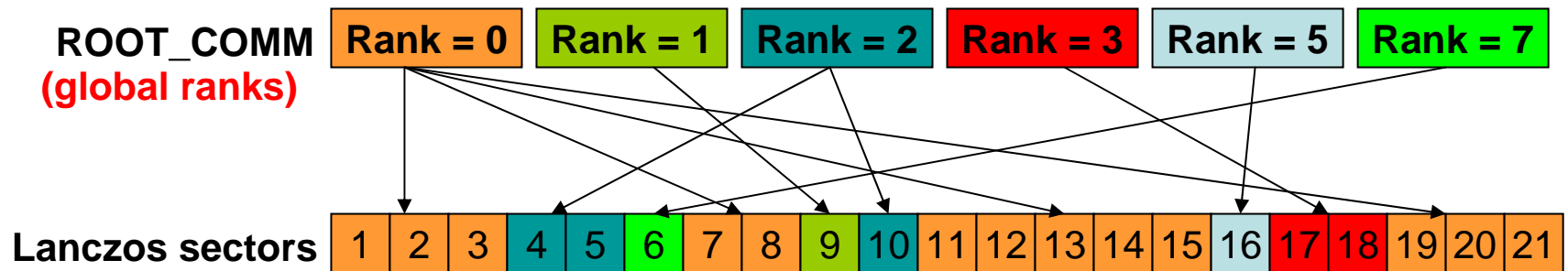
- ❑ We reorthogonalize each new Lanczos vector against all old ones
- ❑ Lanczos vectors are stored on disk. **This requires fast I/O**

( 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:

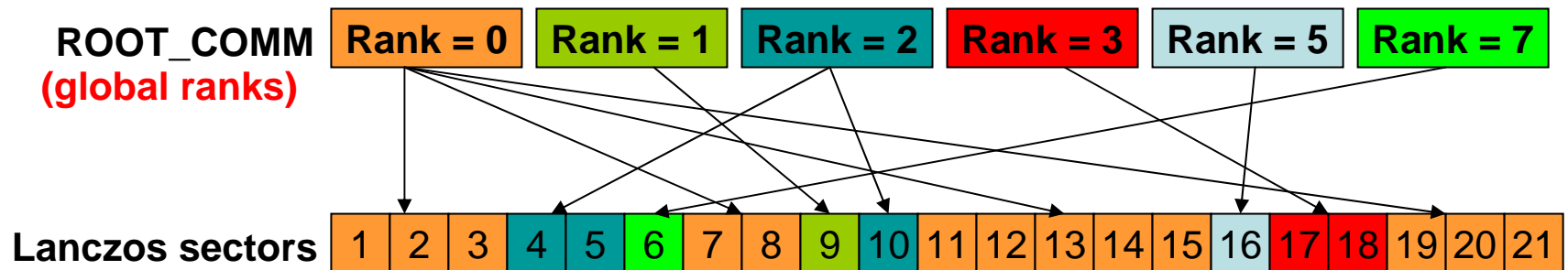
## Reorthogonalization:

- ❑ 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 )**



## Local dimension of Lanczos vector for ROOT\_COMM



# 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)