NESAP code optimizations for ab initio nuclear structure calculations

Pieter Maris

Dept. of Physics and Astronomy
Iowa State University
Ames, IA 50011

Nersc User Group meeting, July 2019, Rockville MD

Ab initio nuclear strucuture calculations

Given a Hamiltonian operator

$$\hat{\mathbf{H}} \;=\; \sum_{i<j} \frac{(\vec{p}_i - \vec{p}_j)^2}{2\,m\,A} + \sum_{i<j} V_{ij} + \sum_{i<j<k} V_{ijk} + \ldots$$

solve the eigenvalue problem for wave function of A nucleons

$$\hat{\mathbf{H}}\,\Psi(r_1, \ldots, r_A) \;=\; \lambda\,\Psi(r_1, \ldots, r_A)$$

- Eigenvalues λ discrete (quantized) energy levels
  - total energies: $E_\Psi \;=\; \langle\Psi|\hat{\mathbf{H}}|\Psi\rangle \;=\; -E_\Psi^{\text{binding}}$
  - excitation energies: $E_{\mathrm{exc}} \;=\; E_\Psi - E_{\mathrm{gs}}$
- Eigenvectors: representation of A-body wave function

Challenges

- Self-bound quantum many-body problem,
  with $3A$ degrees of freedom in coordinate (or momentum) space
- Not only 2-body interactions, but also intrinsic 3-body interactions
  and possibly 4- and higher N-body interactions
- Strong interactions, with both short-range and long-range pieces

No-Core Configuration Interaction approach

Barrett, Navrátil, Vary, *Ab initio no-core shell model*, PPNP69, 131 (2013)

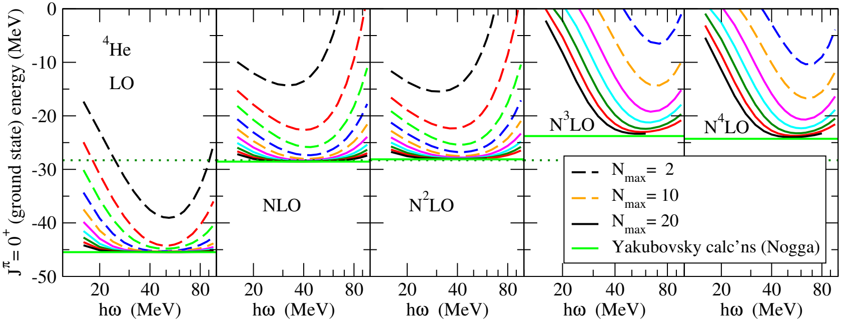
- Expand wavefunction in basis states $|\Psi\rangle = \sum a_i |\Phi_i\rangle$
- Express Hamiltonian in basis $\langle\Phi_j|\hat{\mathbf{H}}|\Phi_i\rangle = H_{ij}$
- Diagonalize Hamiltonian matrix H_{ij}
- No-Core: all A nucleons are treated the same
- Complete basis $\longrightarrow$ exact result
  - caveat: complete basis is infinite dimensional

- In practice
  - truncate basis
  - study behavior of observables as function of truncation

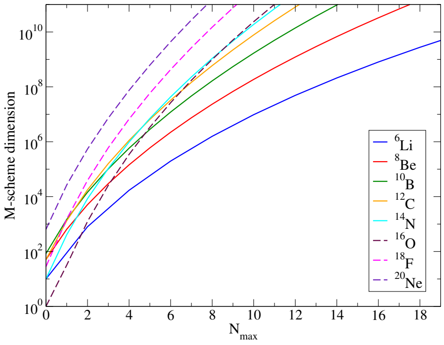
- Computational challenge
  - construct large ($10^{10} \times 10^{10}$) sparse symmetric matrix H_{ij}
  - obtain lowest eigenvalues & -vectors corresponding to low-lying spectrum and eigenstates

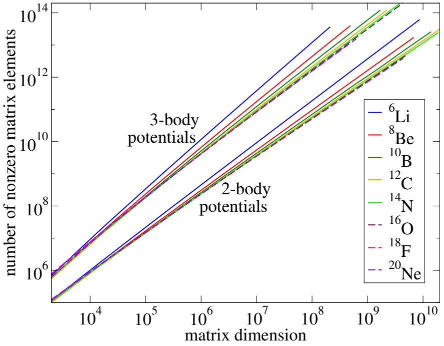
Convergence

- Variational: for any finite truncation of the basis space, eigenvalue is an upper bound for the ground state energy
- Smooth approach to asymptotic value with increasing basis space
- Convergence: independence of both $N_{\max}$ and H.O. basis $\hbar\omega$
  - different methods using the same interaction should give same results within (statistical plus systematic) numerical uncertainties



Computational Challenge





- Increase of basis space dimension with increasing A and N_{max}
  - need calculations up to at least $N_{\text{max}} = 8$, preferably $N_{\text{max}} = 10$ for meaningful extrapolation and numerical error estimates
- More relevant measure for computational needs
  - number of nonzero matrix elements
  - current limit 10^{14} (Cori, Theta)

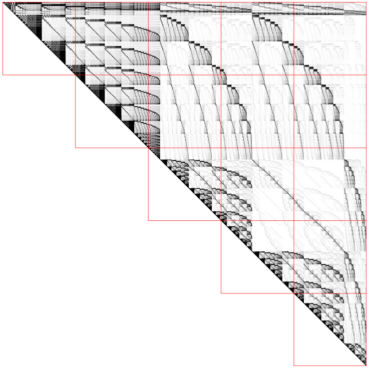
Many-Fermion Dynamics for nuclear structure

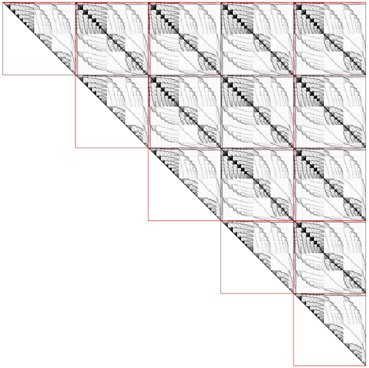
MFDn: No-Core Configuration Interaction code
for nuclear structure calculations

- Fortran legacy code
  - initial version dates back to early 90s (F77)
  - distributed memory parallel code using MPI
  - in use as production code at NERSC since early 2000s
  - since late 2000s also used at OLCF and ALCF
- Ongoing algorithm development and code optimization for current and next-generation HPC platforms
  - SciDAC-2 (UNEDF), SciDAC-3, SciDAC-4 (NUCLEI)
  - Jaguar Early Science, NESAP-KNL, NESAP-Perlmutter
- Hybrid OpenMP / MPI, Fortran 90 – 2003
  - construct many-body matrix H_{ij} from input TBMEs (plus 3NFs)
  - obtain lowest eigenpairs using LOBPCG or Lanczos algorithm
  - use eigenvectors to calculate observables

Distributed symmetric matrix

- Matrix is symmetric, so we only need half the matrix
- Load-balancing
  - 2-dimensional distribution of matrix over MPI ranks
  - local load determined by number of nonzero matrix elements
  - can be achieved by even distribution of many-body (n, l, j) orbitals

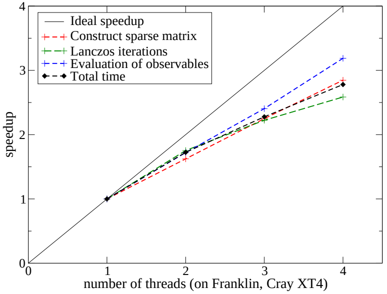


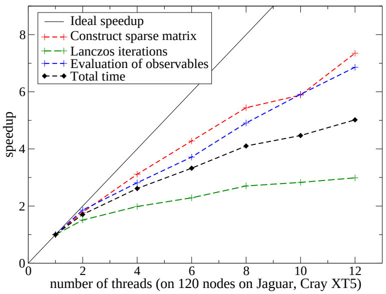


Jaguar Early Science Project (2008-2009)

- Cray XT5 Jaguar at OLCF
- ^{14}C to ^{14}N β-decay with chiral EFT NN + 3NF

  P. Maris, J.P. Vary, P. Navratil, W.E. Ormand, H. Nam, and D.J. Dean, *Origin of the anomalous long lifetime of ^{14}C*, Phys. Rev. Lett. 106, 202502 (2011)
- First hybrid OpenMP / MPI version of MFDn



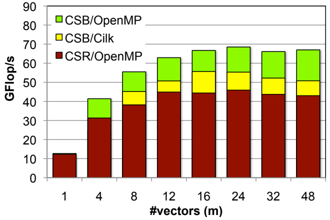


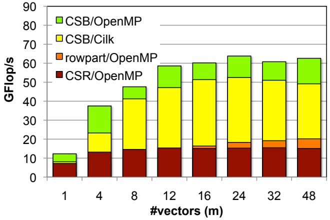
P. Maris, M. Sosonkina, J.P. Vary, E.G. Ng, C Yang, *Scaling of ab-initio nuclear physics calculations on multicore computer architectures*, Procedia Computer Science 1, 97 (ICCS 2010)

Symmetric SpMV & SpMV^T

MFDn is memory bound, so we store only half of the symmetric matrix, and perform SpMV and SpMV^T with the same data structures

- Compressed sparse row (CSR)
  - need private output vectors for SpMV^T to avoid race conditions
  - prohibitively expensive on many-core architectures
- Compressed sparse block (CSB)
  - improves data locality and cache performance
  - allows for efficient OpenMP parallelization for SpMV and SpMV^T

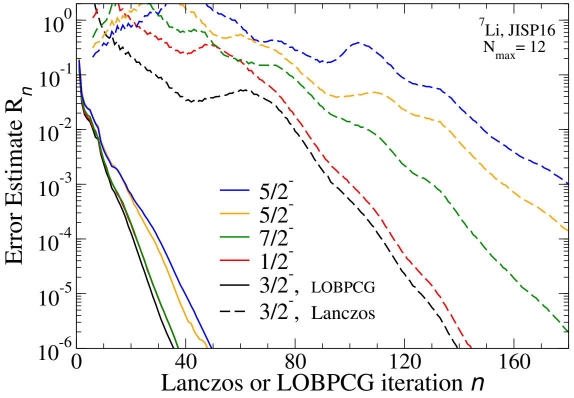




Aktulga, Afibuzzaman, Williams, Buluç, Shao, Yang, Ng, Maris, Vary, *IEEE Transactions on Parallel and Distributed Systems*, DOI 10.1109/TPDS.2016.2630699 (2016)

Lanczos Algorithm vs. LOBPCG solver

Locally Optimal Block Preconditioned Conjugate Gradient:
SpMV acting on block of vectors, which improves cache performance, allows for vectorization, and, with a good preconditioner, needs significantly less iterations compared to Lanczos algorithm

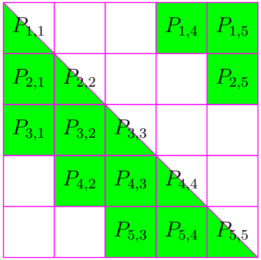


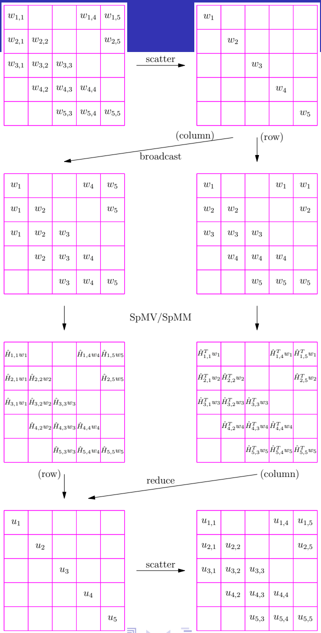
Despite doing approximately 1.6 times more work in SpMV/SpMM,
LOBPCG factor of 2 faster than Lanczos

Shao, Aktulga, Yang, Ng, Maris, and Vary, Comp. Phys. Comm. 222, 1 (2018)

Efficient distributed SpMV

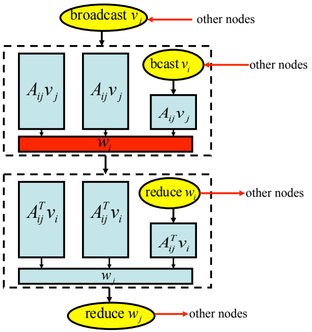
- Communication needs to be load-balanced as well
- Vectors distributed over all processors for orthogonalization

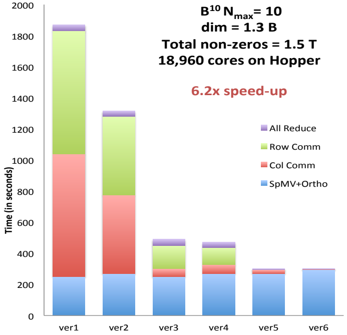




Efficient distributed SpMV – MPI communication

Aktulga, Yang, Ng, PM, Vary, Concurr. Comput. 26 (2014), doi:10.1002/cpe.3129





- Overlap communication with computation
- Optimize mapping onto network topology for non-overlapping communication

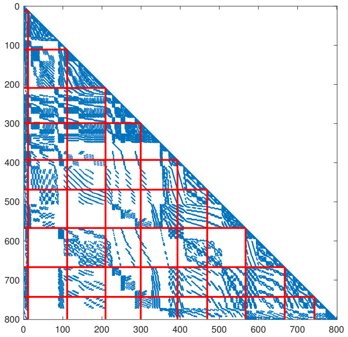
see also Oryspayev, PhD thesis 2016, ISU

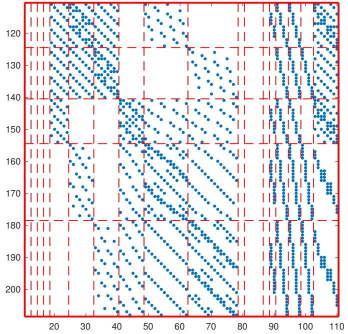
Tuning single-node performance on KNL

- Single-node performace using MFDn proxy
  - local workload of one node out of 5,000 nodes production run
    - construction of local matrix with dimension of about 118×10^6 and 7.5×10^9 nonzero matrix elements
    - local SpMV/SpMM and transpose SpMV/SpMM
    - no communication, no orthonormalization, no 'LOBPCG magic'
- Explore MPI and OpenMP scaling within node
  - near-perfect MPI and OMP scaling up to 64 (68) ranks $\times$ threads
  - OMP shared memory within node minimizes memory footprint
- Optimize memory placement
  - quad-flat with vectors in MCDRAM and matrix in DDR4 gives best performance but gain is offset by extra reboot time
- Vectorization
  - use compiler report to see which loops vectorize automatically
  - use OpenMP4 SIMD directives for manual vectorization
  - split complicated innerloops into smaller and simpler subloops

B. Cook, P. Maris, M. Shao, N. Wichmann, M. Wagner, J. O'Neill, T. Phung and G. Bansal,
High performance optimizations for nuclear physics code MFDn on KNL, LNCS 9945, 366 (2016)

Matrix sparsity structure





- *A*-body problem with *a*-body interaction: nonzero matrix elements iff at least $(A - a)$ particles are in identical single-particle states
- Nonzero tiles of varying size (dashed lines)
- Tiles are combined to form (approximately) square CSB blocks

Matrix construction

Compare pairs of many-body states to determine sparsity structure

- count nonzero tiles
- within nonzero tiles count nonzero matrix elements

Construct nonzero matrix element

- store in CSB format (row, column, value) using 16-bit integers for row and column indices within CSB block

Calculation of observables after obtaining eigenvectors

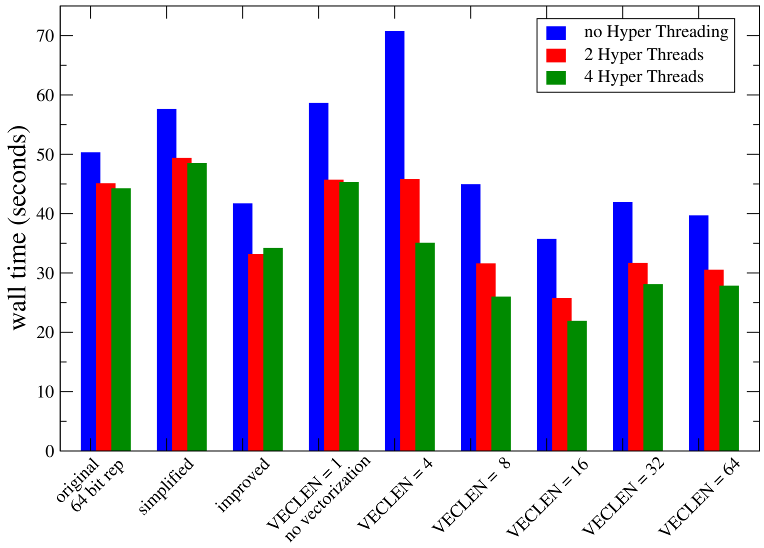
```
!$OMP DO SCHEDULE(dynamic)
    do i = 1, ncolumns
      nnonzero = 0
      do j = 1, nrows
!   compare truncated bitrepesentations
        xor = IEOR(col_bitrep(i), row_bitrep(j))
        ndiffs = popcnt(xor)
        if (ndiffs .gt. 2*hrank) cycle
!   compare many-body states phi
        call MBstate_differences(
   $           nparticles,
   $           colstatelist(1:nparticles, i)
   $           rowstatelist(1:nparticles, j),
   $           ndiffs)
        if (ndiffs .le. hrank) then
          nnonzero = nnonzero + 1
          ...
          call constructME(...)
          ...
        endif
      enddo
    enddo
!$OMP END DO
```

Performance improvements matrix construction

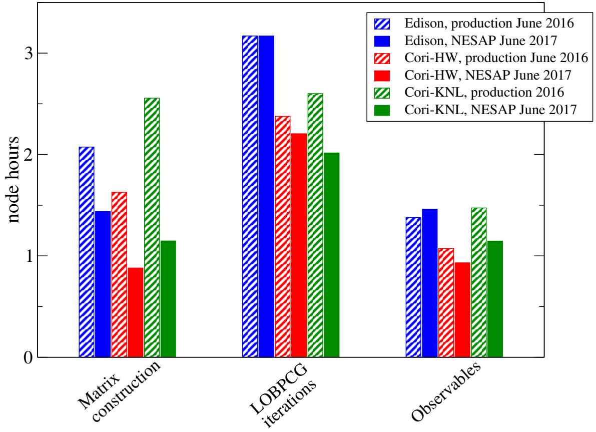
- Intel compiler opimization report:
  inner loops do not vectorize
  - no vector instruction for fortran function popcnt
  - subroutine MBstate_difference contains lots of branching, cycle, and early exit statements
- Without vectorization, performance is poor
- Strategy to improve performance
  1. simplify MBstate_difference from about 120 lines to 20 lines
     - not dealing with exceptions, which increases work-load slightly
     - remove cycle and early exits (may need to pad several arrays)
     - naively, significantly larger work-load (more comparisons executed), but in practice only slightly slower
  2. split inner loop to improve cache performance
  3. split inner loops into subloops of appropriate length for vectorization

presented at IXPUG 2016, Sept. 2016, Argonne IL

Improved matrix construction – performance



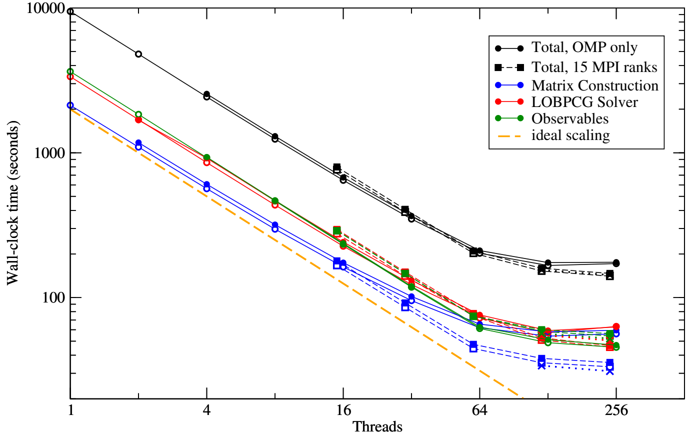
Comparison Edison vs. Cori-Haswell vs. Cori-KNL



- dimension 252 million, with 400 billion nonzero matrix elements
- 124 nodes on Edison
  62 nodes on Cori
  using 496 MPI ranks

- Tuning for KNL also improves performance Cori-HW and Edison

Single-node scaling on KNL (Cori, Theta)

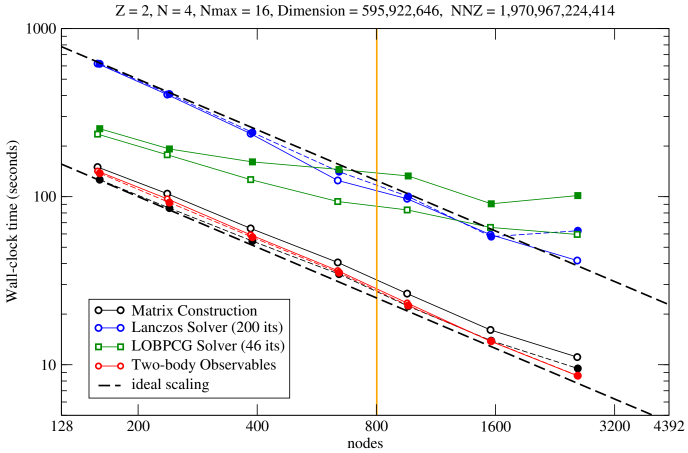


- Good scaling up to number of cores available on both Cori (open symbols) and Theta (closed symbols)

Communication issues

- Communication time fluctuates wildly between different runs
  - depends on placement of job on hardware
  - solution: restrict job to subset of available switches
- Useful to tune some of the many MPI settings
  - most important: module load craype-hugepages2M
- One MPI rank per node: communication by only one core
  - one core cannot saturate communication bandwidth
  - MPI standard allows more threads to perform MPI communication however, MPI standard only guarantees correctness, not efficiency
  - in practice collective MPI calls by multiple threads get serialized . . .
  - solution: use 4 or 8 MPI ranks per node, even though overall memory footprint and communication volume increase
- Reduction operations take significant amount of time
  - executed by a single thread only
  - solution: use user-defined multithreaded reduction operator
- Communication volume for LOBPCG implementation is 8 to 16 times larger than for Lanczos
  - Bcast and Reduce of sets of vectors, instead of single vectors

Strong Scaling on Theta (4 and 16 MPI ranks/node)



- Lanczos scales well up to (almost) the entire machine, but communication becomes a bottleneck for LOBPCG solver
- With 3-body forces scaling of solver is significantly better

NESAP for Perlmutter (2019-2020)

- Benchmark runs on Edison
- Use OpenMP with PGI compiler for GPU offload
  - benchmark source code + test cases available
  - stand-alone version of LOBPCG solver + test case
- Revisit using CUDA for GPU offload of matrix construction
  - initial version was developed for Titan around 2012-2014, but only for matrix construction with 3-body forces, and not recently updated due to lack of manpower . . .
- Roofline analysis of determination sparsity structure and matrix construction
  - extract single-node 'simulator' plus representative input data
- MPI communication
  - extract MPI communication motif during iterative solver
  - translate into 'Ember' and use Structural Simulation Toolkit (SST) to simulate the MPI communication
  - make available to Cray for simulations for Slingshot network

MPI Communication Skeleton Simulations

SpMV and SpMV^T

call MPI_AllGatherV(..., col_com)

call MPI_Bcast(..., row_com)

call MPI_Reduce(..., row_com)

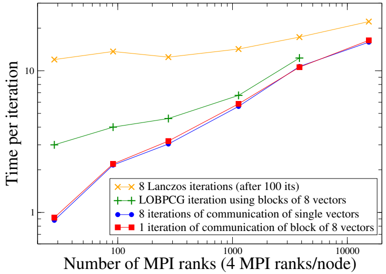
call MPI_Reduce_Scatter(..., col_com)

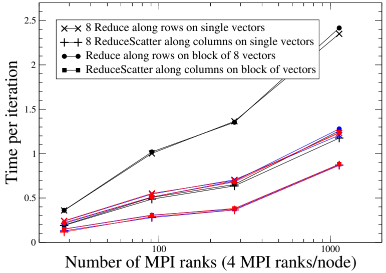
Orthogonalization, LOBPCG

call MPI_AllReduce(..., MPI_COMM_WORLD)

Repeat

- communication dominates
  SpMM on $>$ 1,000 ranks
  SpMV on $>$ 20,000 ranks
- using more cores for reduction:
  naive OpenMP loop and MKL saxpy



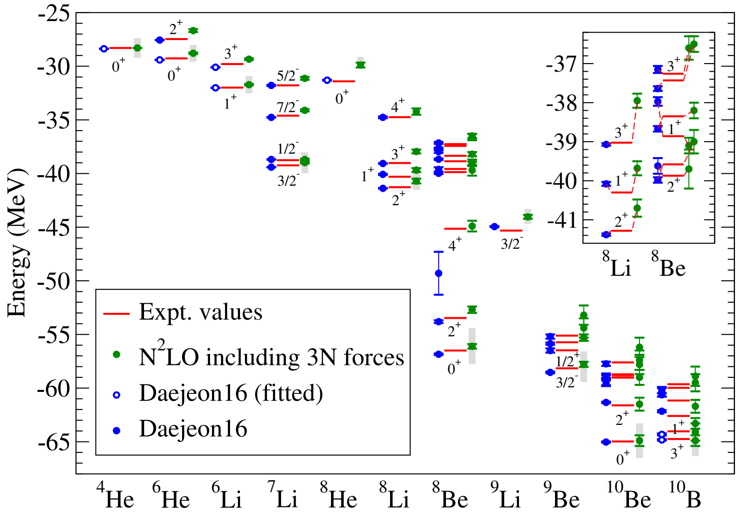


Concluding remarks

- NESAP for Cori was essential for us
  in order to get acceptable performance on KNL
  - without vectorization, KNL does not perform as well as Haswell
  - without NESAP, we would not have vectorization
    in the matrix construction, nor in the evaluation of observables
  - dungeon session (April 2016) was essential
    to get this effort jump-started
- NESAP for Cori was useful for other systems
  - Theta at ALCF
  - also improved performance on Edison and Cori-Haswell
- Excited about NESAP for Perlmutter
  - frequent interaction with Brandon Cook
  - extraction of communication motif alread gives us better
    understanding of current performance on Cori-KNL

NERSC staff is available to help – make use of it

Ground state energies of light nuclei

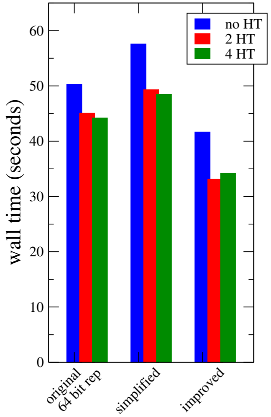


P. Maris, I.J. Shin, and J.P. Vary, in preparation (2019)

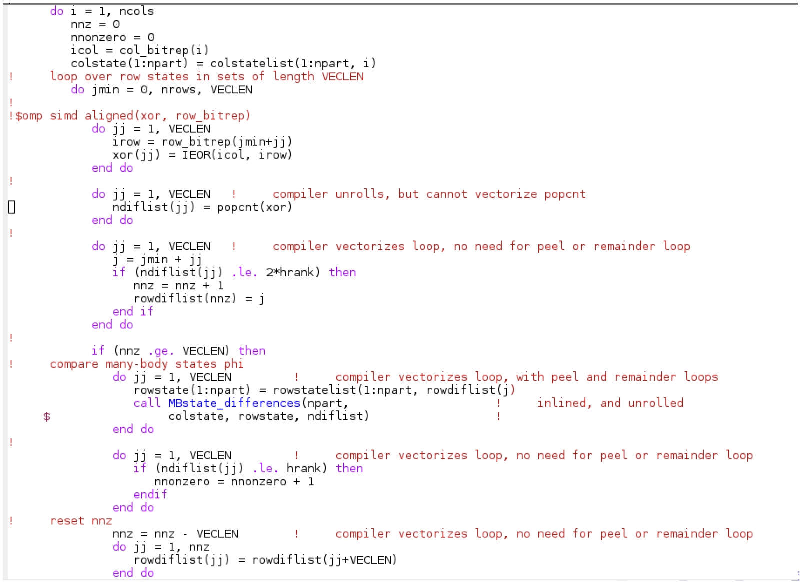
Performance improvements

```
!$OMP DO SCHEDULE(dynamic)
      do i = 1, ncols
         nnz = 0
!     compare truncated bitrepesentations
         do j = 1, nrows
            xor = IEOR(col_bitrep(i), row_bitrep(j))
            ndiffs = popcnt(xor)
            if (ndiffs .le. 2*hrank) then
               nnz = nnz + 1
               rowdiflist(nnz) = j
            endif
         enddo
!
!     compare many-body states phi
         colstate(1:nparticles) =
     $        colstatelist(1:nparticles, i)
         do j = 1, nnz
            rowstate(1:nparticles) =
     $           rowstatelist(1:nparticles, rowdiflist(j)
            call MBstate_differences(nparticles,
     $           colstate, rowstate, ndiflist)
            if (ndiffs .le. hrank) then
               nnonzero = nnonzero + 1
            endif
         enddo

!     collect nonzeros
         nnonzero = 0
!     to be vectorized
         do j = 1, nnz
            if (ndiflist(j) .le. hrank) then
               nnonzero = nnonzero + 1
               ...
`            endif
         enddo
      enddo
!$OMP END DO
```



Performance improvements



```
      do i = 1, ncols
         nnz = 0
         nnonzero = 0
         icol = col_bitrep(i)
         colstate(1:npart) = colstatelist(1:npart, i)
!     loop over row states in sets of length VECLEN
         do jmin = 0, nrows, VECLEN
!
!$omp simd aligned(xor, row_bitrep)
            do jj = 1, VECLEN
               irow = row_bitrep(jmin+jj)
               xor(jj) = IEOR(icol, irow)
            end do
!
            do jj = 1, VECLEN   !     compiler unrolls, but cannot vectorize popcnt
□              ndiflist(jj) = popcnt(xor)
            end do
!
            do jj = 1, VECLEN   !     compiler vectorizes loop, no need for peel or remainder loop
               j = jmin + jj
               if (ndiflist(jj) .le. 2*hrank) then
                  nnz = nnz + 1
                  rowdiflist(nnz) = j
               end if
            end do
!
            if (nnz .ge. VECLEN) then
!     compare many-body states phi
               do jj = 1, VECLEN         !     compiler vectorizes loop, with peel and remainder loops
                  rowstate(1:npart) = rowstatelist(1:npart, rowdiflist(j)
                  call MBstate_differences(npart,                     !     inlined, and unrolled
     $                 colstate, rowstate, ndiflist)                  !
               end do
!
               do jj = 1, VECLEN         !     compiler vectorizes loop, no need for peel or remainder loop
                  if (ndiflist(jj) .le. hrank) then
                     nnonzero = nnonzero + 1
                  endif
               end do
!     reset nnz
               nnz = nnz - VECLEN        !     compiler vectorizes loop, no need for peel or remainder loop
               do jj = 1, nnz
                  rowdiflist(jj) = rowdiflist(jj+VECLEN)
               end do
```