Algorithms for a Nonlinear Hermitian Eigenproblem

Grady Schofield[1], James R. Chelikowsky[1], Yousef Saad[2]

Spectrum Slicing

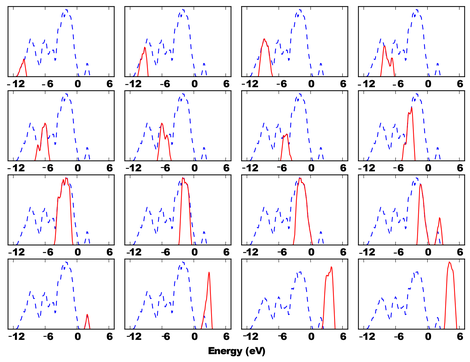
The Kohn Sham equation is a nonlinear eigenproblem where the number of eigenpairs sought corresponds roughly to the number of electrons in the system. Solving the equation involves solving the eigenproblem iteratively, looking for a fixed point in terms of the density. Since the number of eigenpairs sought scales with the number of electrons in the system, analyzing larger systems means increasing the number of eigenpairs computed. The cost of generating the vectors for an approximation space scales quadratically in the number of states, but operations associated with turning the approximation space into approximate eigenvectors, orthogonalization and a Rayleigh-Ritz procedure, scale cubically in the number of states. We developed a spectrum slicing method to address this issue. In this method, the spectrum is partitioned into disjoint slices and the eigenpairs are computed in smaller subspaces associated with each slice as depicted below. The method also functions as a parallelization scheme since separate groups of processors can work on each slice.

The Kohn-Sham equation

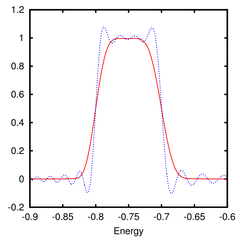
$$-\nabla^2\psi + (V_{\text{ion}} + V_{\text{H}} + V_{\text{xc}}[\rho])\psi_i = \lambda_i\psi_i$$

$$\rho = \sum_i f(\lambda_i)\psi_i^2$$

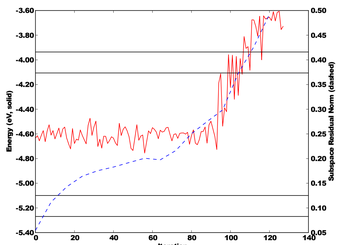
Boundary conditions are either zero Dirichlet or periodic in 1, 2, or 3 dimensions for wires, slabs or solids respectively.



The occupied states of a silicon cluster with 525 atoms are shown in blue. The red slices were computed with the spectrum slicing code, extending into the unoccupied or virtual states in the lower left corner. Each slice was computed independently on a separate set of processors. Each set of processors deals with a smaller approximation subspace, reducing the cost of the cubic scaling subspace operations.

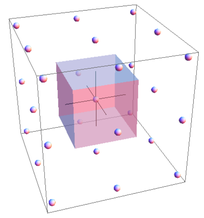


To enhance a portion of the spectrum we use a Chebyshev-Jackson polynomial approximation of a step function as a filter. This is a classical Chebyshev approximation with the high order terms attenuated by a smooth function. The Chebyshev-Jackson filter is shown in red. This smoothness prevents convergence of eigenpairs whose eigenvalue lays in the local extremum of the ripples in the blue line, allowing us to use the convergence heuristic described below.



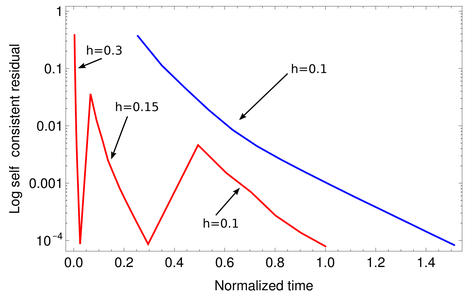
The red line shows the Ritz value (approximate eigenvalue) of the most recent vector iterate. The inner horizontal lines are the bounds of the slice and the filter satisfies some precision criteria at the outer horizontal lines. We monitor this red line for it drifting out of the upper slice bounds and use this as a convergence heuristic. Once the red line leaves the energy range of the slice, the method will be finding eigenpairs outside the slice. We avoid repeating a Rayleigh-Ritz procedure this way.

High Order Hamiltonian and Postprocessing

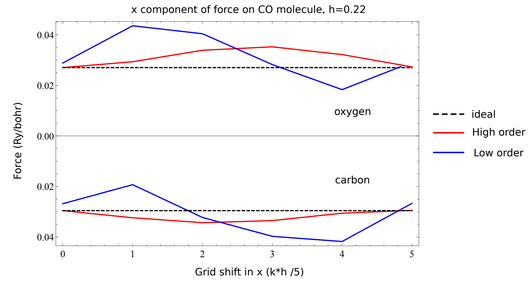


The key to the high order methods we are implementing is a Taylor series approximation in a cube around each gridpoint. By using numerical derivatives for high order postprocessing or their symbolic counterpart in terms of unknown grid data for Hamiltonian operators, we get the accuracy of a finer grid on a coarse grid.

A typical finite difference discretization for the Kohn-Sham problem involves a stencil for the Laplacian and pointwise evaluation of the potentials. The eigenvalues and total energies converge faster than many other quantities of interest, such as forces, because computing the these quantities requires the eigenvectors which converge more slowly than the eigenvalues. We are developing a method that uses high order Taylor series at each grid point to give a representation of the pontetials and high order approximations of the solution for postprocessing. Using modern C++11 features we are able to use the same code for high order numerical calculations as well as symbolic manipulations necessary to derive the high order functionals that represent potentials in the Hamiltonian.

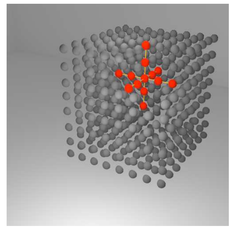


We can project solutions from coarser grids onto finer grids using the Taylor series approximation between grid points. After reaching self consistency on the coarse grid (finding the fixed point), a new iteration can be started on the finer grid with a better guess than would otherwise be possible. The first step on the finer grid shows a jump in the self consistent residual error (red line). We are current working on understanding this jump and reducing it. Even with relatively large jumps, a savings of 35% is possible for a transition from 0.3 to 0.15 to 0.1 versus doing the entire computation at 0.1.



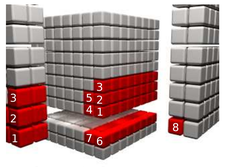
The interatomic forces computed on two atoms show the effect of discretization error as the grid is translated. The ideal result would be a horizontal line, translational independence of the result. Using high order operators for part of the ionic potential and high order Taylor series approximations of the solution in the postprocessing, the variation around the horizontal line is reduced by a factor of two. To get closer to the ideal solution, we are looking at high order operators for the remaining part of the ionic potential as well as the Hartree and exchange-correlation potentials.

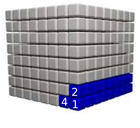
High Performance Compute Kernels

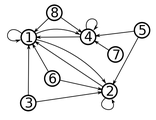


The most elementary implementation of the high order finite differences Laplacian operator uses an index array to sample the points needed in the stencil around the point of interest, shown to the left. The code for this is not easily vectorized, and if the compiler does not unroll loops to the point that it is handling multiple stencils at once, then pipelining is harder to acheive in the floating point unit. We are working on a method to generate optimized code at runtime, exploiting knowledge of the domain's shape, to get better vectorization, cache reuse and pipelining. The programmer writes a metaprogram in C++ that compiles to machine language at runtime where characterisitics of the CPU such as depth of the floating point pipeline, cache sizes, etc. are input parameters. A preliminary implementation of this idea showed the Laplacian stencil running at around 50% of peak performance for a realistic problem and scaling linearly on Hopper compute nodes, which have 24 cores.

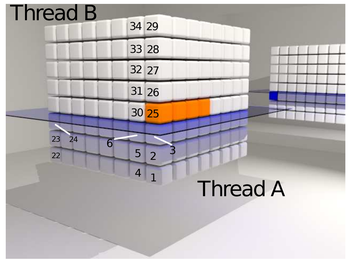
A high order treatment of the Laplacian term in the Hamiltonian uses a stencil that samples points around a grid point. This results in a sparse matrix structure. but the rectilinear structure of the stencil makes it possible to implement the operations with a vectorized code.







The grid nature of the discretization scheme allows us to collect nearby rows of grid points into groups and process the rows as vectors. If we consider these arrays componentwise, the fact that we've taken a group of them at once means we have multiple stencils whose operations are independent of each other. Interlacing operations from the various stencils improves pipelining in the code. In the figure above the blue rows are output arrays, and we would have three stencils being computed at once. Not all of the necessary input arrays for the stencil are shown in red.



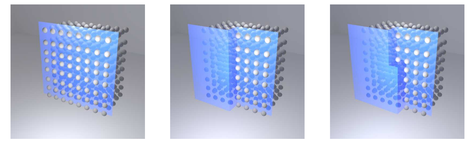
One goal of this project is moving toward a thread based architecture with fewer MPI processes running on the compute node. Each thread uses shared data, synchronizing with simple latches in most cases, where the data is arranged to minimize false sharing in the CPU cache. In the figure above, given the ordering of grid point arrays, false sharing can only occur between the data in the last array handled by thread A and the first array handled by thread B. We have observed the latch based synchronization used in the matrix vector product scaling all the way up to 60 threads on a Xeon Phi coprocessor.

Scalable RDMA Network Code

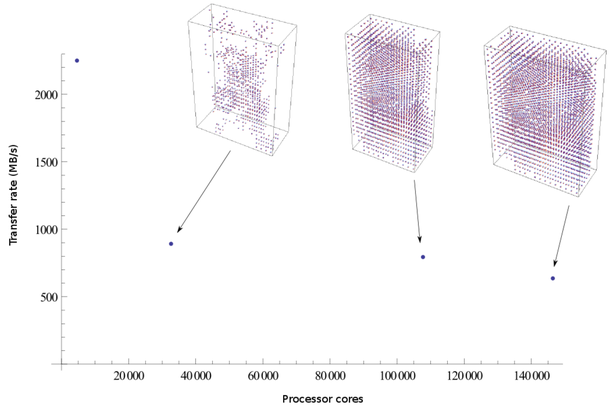
The most costly step in the algorithm for solving the eigenvalue problem is the filtering operation that produces the basis for the approximation subspace. This filtering step consists of matrix-vector products. For a finite difference discretization of the Kohn-Sham Hamiltonian, communication dominates the cost of doing these products. We have developed a partitioning scheme (and associated algorithms) that attains these goals.

1. We want to minimize copying of data into send buffers. Instead we want to do communication directly from the data buffers where data already resides.
2. We want to communicate with fewer large messages as opposed to more small messages. This is because there is a steep drop in peek transfer rates for smaller messages.
3. We want to communicate with as few neighboring compute nodes as possible, this works toward the previous goal of fewer large messages.
4. We want to use remote direct memory access capabilities of the network hardware because this is the normal mode of operation for supercomputer networks and stands the best chance of getting the highest performance.

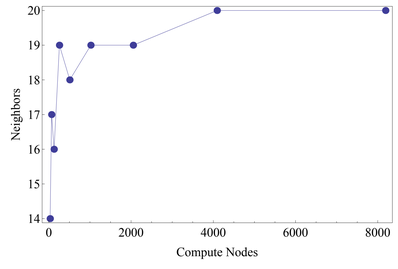
We have developed a code that achieves these goals using RDMA capabilities with either Infiniband verbs or Cray's Generic Network Interface (GNI) library. A large test was performed on a DNA system, see bellow, using NERSC's Hopper computer.



The new network code achieves better load balancing by partitioning domains to even out the number of grid points per compute node. Allowing partition planes with notches, as depicted above can prevent load imbalances of as much as 20%. This type of partition scheme complicates the situtaion when trying to reduce the copying of data into send buffers.



This is the average send/receive rate for our matrix-vector multiplication operations. The boxes show the compute nodes used on Hopper at NERSC, which has a 3D network topology. The system tested was a DNA system where the grid spacing was reduced to keep the number of grid points per core constant in the last three data points. Further improvements in the transfer rate could be made by reordering the nodes.



Part of obtaining better scalability in the matrix-vector product comes from minimizing the number of neighboring compute nodes that must converse with a given node during the calculation. The partition scheme we have implemented keeps the number to a minimum even as the number of compute nodes (not cores) grows large, as shown in the figure above.

1. Institute for Computational Engineering and Sciences, University of Texas at Austin
2. Department of Computer Science and Engineering, University of Minnesota