Distributed Memory Implementation of Coupled Cluster

Justin Gage Lietz, Peter Ahrens, Stephanie Lauber

Introduction

Accurate calculations of neutron star equation of state require realistic nuclear forces combined with ab initio many-body methods to solve the Schrödinger equation. The coupled cluster approximation to the Schrödinger equation scales polynomially with respect to problem size, yet still maintains a high level of accuracy for ground state energies. This an improvement over methods that scale exponentially, such as configuration interaction.

The balance between scalability and accuracy of the solution makes coupled cluster desirable in a range of applications including nuclear physics, quantum chemistry and condensed matter physics.

However, to solve larger systems with greater accuracy, the memory requirements of coupled cluster requires a distributed computing environment. Implementing an efficient distributed memory algorithm for coupled cluster requires non-trivial load balancing and cross-node communication.

Coupled Cluster Doubles Equation

$$
\begin{aligned}
0 &= \langle ab||ij\rangle + \hat{P}(ab)\sum_{c} f_{bc}t_{ij}^{ac} - \hat{P}(ij)\sum_{k} f_{kj}t_{ik}^{ab} \\
&+ \tfrac{1}{2}\sum_{cd}\langle ab||cd\rangle t_{ij}^{cd} + \tfrac{1}{2}\sum_{kl}\langle kl||ij\rangle t_{kl}^{ab} + \hat{P}(ij|ab)\sum_{kc}\langle kb||cj\rangle t_{ik}^{ac} \\
&+ \tfrac{1}{4}\sum_{klcd}\langle kl||cd\rangle t_{ij}^{cd}t_{kl}^{ab} + \hat{P}(ij)\sum_{klcd}\langle kl||cd\rangle t_{ik}^{ac}t_{jl}^{bd} \\
&- \tfrac{1}{2}\hat{P}(ij)\sum_{klcd}\langle kl||cd\rangle t_{ik}^{dc}t_{lj}^{ab} - \tfrac{1}{2}\hat{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{lk}^{ac}t_{ij}^{db}
\end{aligned}
$$

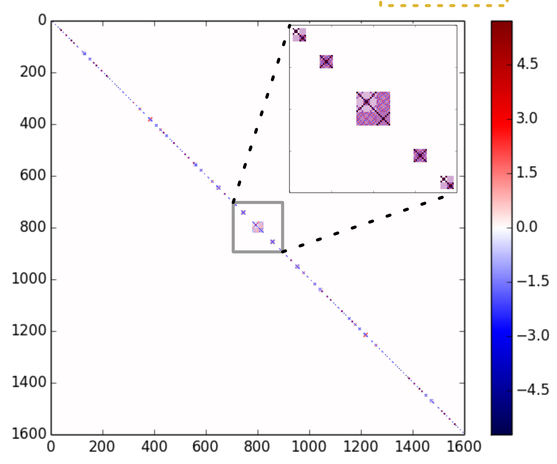
The coupled cluster equations iteratively solve for the cluster amplitudes, representing the state of the particles.

Currently, coupled cluster computations are done by maintaining an individual copy of the smaller amplitude matrix, represented by t_{ij}^{ab}, on each node and dynamically distributing the large, ***sparse*** block-diagonal interaction matrix, requiring a global communication at each iteration.

By taking advantage of physical symmetries in the system, we implement a static load balancing scheme that allows for the distributed storage and computation of the large interaction matrix, reducing individual process wait time.

Sparse Block-Diagonal Matrix Structure

$$\vec{P}_a + \vec{P}_b \neq \vec{P}_c + \vec{P}_d \implies \langle ab||cd\rangle = 0$$

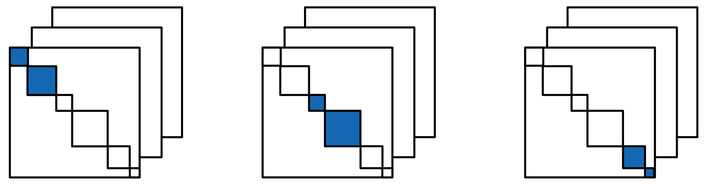


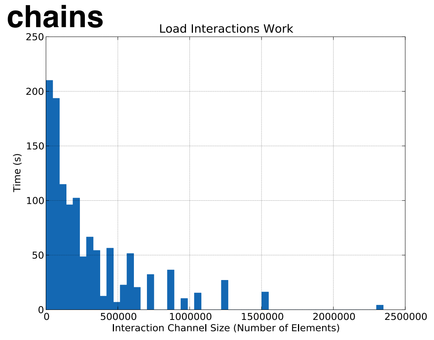
The sparse block diagonal structure of the matrix is determined by the symmetries of the interaction. Conservation laws restrict which matrix elements can be non-zero.

Distributed Algorithm and Load Balancing

Step 1

Initialization and memory balancing of interaction chains



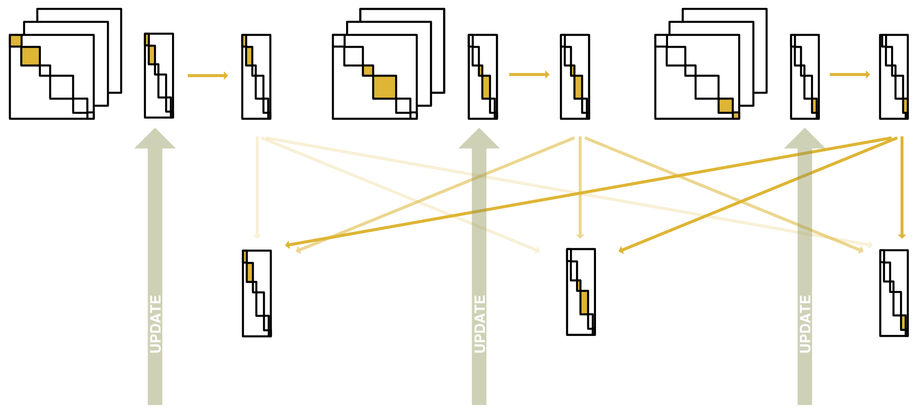


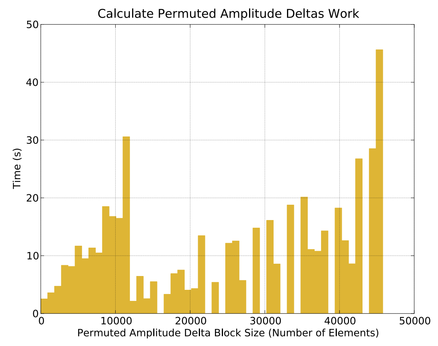
The dimensions of the blocks are pre-computed; the load balancer takes in these dimensions and evenly distributes the work load. Each processor constructs a local piece of the total interaction matrix.

There are many small blocks and few outliers, making it easy to balance.

Step 2

Calculate and distribute permuted local amplitudes



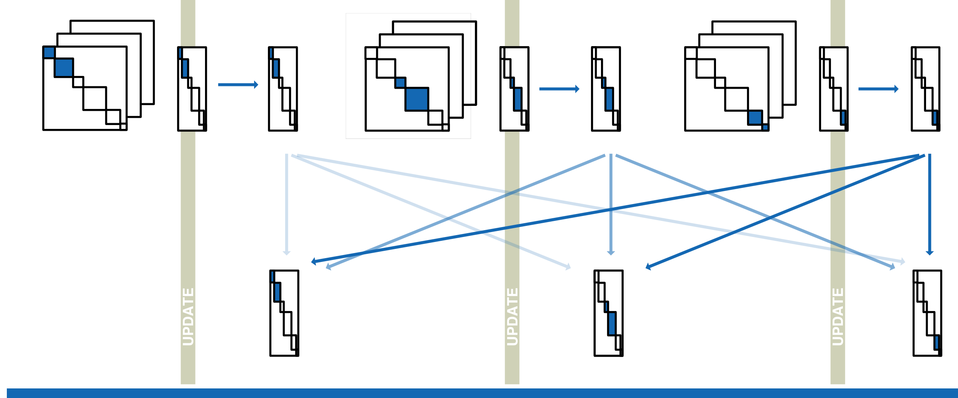


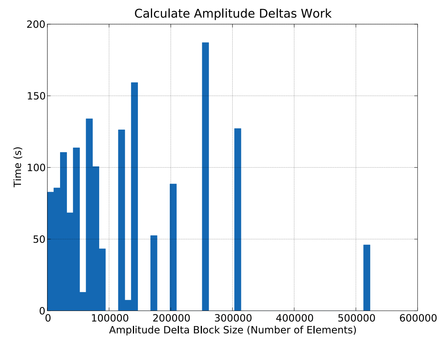
Each rank then performs many matrix multiplications using its local copy of the interaction matrix to update the permuted amplitudes. Each rank communicates the local update to all other ranks.

The permuted matrix multiply is harder to balance because of the large diversity of task sizes.

Step 3

Calculate and distribute local regular amplitudes



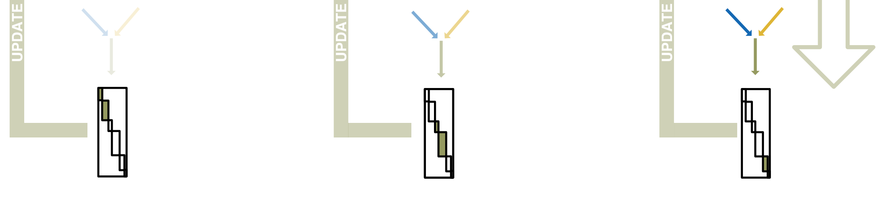


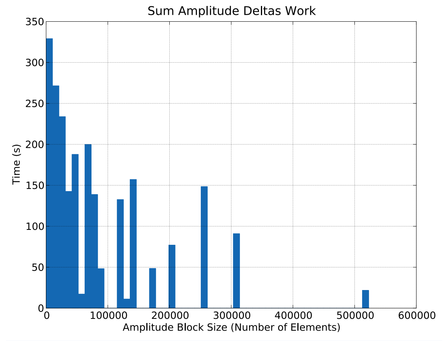
We perform a similar process as step 2 on the non-permuted (regular) amplitudes.

Small and medium size matrices dominate the calculation.

Step 4

Calculate final regular amplitudes



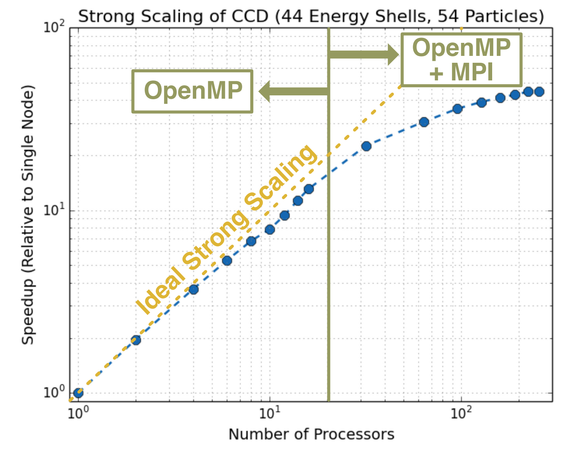


Each rank receives information from all other ranks to calculate the complete updated amplitude, finishing the iteration.

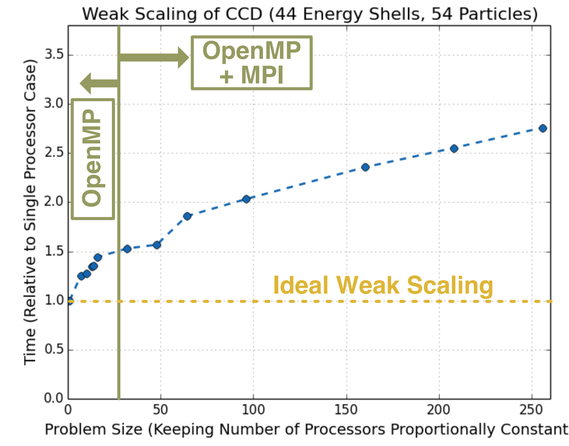
The summation of the final amplitude is not a costly step.

Results

Scaling Results



We achieved **strong scaling** up to 256 processors. We show shared memory on-node parallelism to the left of the green line and distributed memory internode parallelism to the right. Test cases were run on up to 16 Intel® Xeon® E5-2670 CPUs @ 2.60 GHz with 16 cores each.



Weak scaling is within a factor of **2.8** of the ideal for up to 256 processors. We show shared memory on-node parallelism to the left of the green line and distributed memory internode parallelism to the right. Test cases were run on up to 16 Intel® Xeon® E5-2670 CPUs @ 2.60 GHz with 16 cores each.

Future Work

To increase the scalability of the application, we will extend the static load balancing scheme to include the distribution of the amplitude matrices, further reducing the per-node memory requirements, while balancing the increasing communication overhead.

This is necessary when the amplitude matrices can no longer fit on a single node.

References

[1] Shavitt, I. and Bartlett, R. J. , Cambridge University Press (2009).

[2] Dean, David J., Physics Today, Nov. 2007, Page 48.

Acknowledgements

We would like to thank our mentors Bob Robey, Hai Ah Nam and Gabe Rockefeller from the Parallel Computing Summer Research Internship and LANL Institutional Computing Resources.

Los Alamos NATIONAL LABORATORY EST.1943

LA-UR-16-25027