Scalable codes for precision calculations of properties of complex atomic systems

C. Cheung1, M. S. Safronova1,2, S.G. Porsev1,3, M. G. Kozlov1,4
1Department of Physics and Astronomy, University of Delaware, DE, USA
2Joint Quantum Institute, NIST and the University of Maryland, College Park, MD, USA
3Petersburg Nuclear Physics Institute, Gatchina 188300, Russia
4St. Petersburg Electrotechnical University “LETI”, St. Petersburg, Russia

Abstract

High precision atomic data is indispensable for experiments involving studies of fundamental interactions, astrophysics, atomic clocks, plasma science, and others. We develop new parallel atomic structure codes and explore the difficulties of load-balancing in these codes. Efficient load-balancing of matrix elements for many-electron systems is very difficult due to the intrinsic nature of the computational methods used to compute them. By arithmetically selecting determinants for each core, we achieve very even workload distribution, and attain near-perfect linear scalability and efficiency with the number of cores. Our newly developed codes enable computations that were not possible before due to lack of memory or prohibitive computation times and allow a broader range of correlations to be investigated in a shorter period of time. We also remove the need for users to set certain array parameters. We tried several parallelization schemes and explore the difficulties of load-balancing in these codes.

Summary

Challenges of parallelizing calculations of many-electron systems

Case 1:

<table>
<thead>
<tr>
<th>Determinant</th>
<th>47</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
<th>55</th>
<th>56</th>
<th>57</th>
<th>58</th>
<th>59</th>
<th>60</th>
<th>61</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormH: Near-perfect linear scaling!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case 2:

<table>
<thead>
<tr>
<th>Determinant</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
<th>55</th>
<th>56</th>
<th>57</th>
<th>58</th>
<th>59</th>
<th>60</th>
<th>61</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormH: Even computational time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Question: How many differences in orbital number between determinants?

<table>
<thead>
<tr>
<th># differences</th>
<th>Matrix element</th>
<th># computations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$H_{ab} = \sum \frac{1}{2} Y_{lj}$</td>
<td>$-N_v + N_v^2$</td>
</tr>
<tr>
<td>1</td>
<td>$H_{ab} = \sum Y_{lj}$</td>
<td>$-I$</td>
</tr>
<tr>
<td>2</td>
<td>$H_{ab} = \sum Y_{lj}$</td>
<td>$-I$</td>
</tr>
<tr>
<td>3+</td>
<td>$N_v$</td>
<td></td>
</tr>
</tbody>
</table>

CI algorithm in a nutshell:

Loop index $a$ over all determinants:

Loop index $k = 1$ to $N_v$:

Want even distribution of matrix elements and even computational time

We tried several parallelization schemes

Parallelization schemes

Scheme 1: Equally distribute total number of determinants
- • heavily uneven distribution of time and matrix elements

Scheme 2a: Save indices of nonzero matrix elements in memory
- • up to 80% efficiency, but requires memory to store indices

Scheme 2b: Brute force parallelize both stages
- • very even distribution of matrix elements, but 50% efficiency

Scheme 3 (current): Arithmetic selection of determinants per core
- • distributed evenly in time and in matrix elements!
- • near-perfect linear scaling and efficiency
- • matrix elements not stored sequentially, causing slow-down during Davidson procedures

New codes developed and tested under UD Caviness cluster (552 cores, 13 TB memory)

Code Developments

Language: Fortran
Libraries: OpenMPI

Completed Tasks:
- • F77 → F90+
- • Dynamic memory allocation
- • Memory footprint reduction of at least 33%
- • Removed user input for defining array parameters
- • MPI parallelization
- • MPI I/O
- • Updated build process with CMake

To Do:
- • Restructuring of code package
- • Modularize and re-package codes
- • Update documentation of codes
- • Optimize and parallelize remaining serial bottlenecks
- • Implement checkpointing

Acknowledgements

This work was supported by U.S. NSF Grant Nos. PHY-1620687 and PHY-1931339.