ACES 3: Efficient Parallel Implementation of MBPT(2) and CCSD Energy, Gradient and Hessian Calculations

Erik Deumens
AcesQC and University of Florida
Jun 29, 2006
Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- How does it work in parallel?
  - Computer science and engineering
- Some examples
  - Performance analysis
What does ACES 3 do?

- Computational chemistry
  - Dynamics and structure of molecules
  - Atomic nuclei move slower
  - Electrons are fast like mosquitoes buzzing around a hiker
Computational chemistry

- Potential Energy Surfaces
  - Minima for stable molecular states
  - Saddle points for transition states
  - Reaction paths

- Need to compute
  - Energy
  - Gradient
  - Hessian
Serial to parallel

- **ACES 2**
  - Serial code
  - Developed since 1990

- **ACES 3**
  - Developed under CHSSI CBD-03
  - Parallel code for compute intense components
    - MBPT(2) energy, gradient, hessian
    - CCSD energy and gradient
Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- How does it work in parallel?
  - Computer science and engineering
- Some examples
  - Performance analysis
Why is this problem hard?

- CCSD calculations are compute and data intensive
  - Large number of T amplitudes
  - Large numbers of integrals
    - to be kept in RAM, or on disk: stored method
    - to be computed multiple times: direct method
Computer Science and Engineering

- Need sophisticated design
  - Exploit parallelism
  - Feasible to write and debug
  - Possible to tune on multiple architectures
    - Distributed memory
      - Ratio of CPU speed vs. communication speed
    - Shared or NUMA memory
  - Easy to maintain
Traditional Design

- control
- compute
- communication
- disk input output
- hardware

code
Traditional Design

- Assumptions
  - Data access latency and bandwidth
  - Computation intertwined with communication
  - Size for data that can be replicated
  - Hardware characteristics must fall in certain ranges to reach performance goals
Traditional Design

- Consequences
  - Detailed analysis by programmer
  - Match data flow with work flow
  - Manage communication deep in code
ACES 3 Design

- Control
- Compute
- Communication
- Disk I/O

Hardware

Code
ACES 3 Design

- Requirement
  - Allow flexibility to control separately at run-time:
    1. Computation
    2. Communication
    3. Disk input and output
ACES 3 Design

- Principles
  - Define units of data
    - For movement and computation
  - Define basic operations on data units
    - All movement is asynchronous
  - Schedule operations and movement
    - Optimize hiding communication behind computation for every machine
    - Optimize data size to make its computation longer than its transportation
Data organization: numbers

<table>
<thead>
<tr>
<th>T_{11}</th>
<th>T_{12}</th>
<th>T_{13}</th>
<th>T_{14}</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_{21}</td>
<td>T_{22}</td>
<td>T_{23}</td>
<td>T_{24}</td>
</tr>
<tr>
<td>T_{31}</td>
<td>T_{32}</td>
<td>T_{33}</td>
<td>T_{34}</td>
</tr>
<tr>
<td>T_{41}</td>
<td>T_{42}</td>
<td>T_{43}</td>
<td>T_{44}</td>
</tr>
</tbody>
</table>
Data organization: blocks

T on node 1
- T(1,1)
- T(1,2)
- T(2,1)

T on node 2
- T(2,2)
- T(1,3)
- T(2,3)
ACES 3 execution

- Input
- algo.sio
- algo.sial
- ACES 3
- SIAL compiler
Parallel architecture

- Distributed data in RAM of workers
  - AO direct use of integrals
  - MO use transformed integrals
- N worker tasks each with 1 GB RAM
- Array blocks are spread over all workers
- Workers compute integrals when integral instruction is called
Parallel architecture

- Served data to and from disk
  - AO no transformation of integrals
  - MO use transformed integrals
- N worker tasks and M server tasks
  - Workers are as before
  - Servers have disk cache and disk
  - Servers take and give blocks
  - Servers compute integrals when asked
ACES 3 coding

- Object oriented to the extreme
- Write code in low level language for super instruction processor to obtain optimal performance
  - Fortran, C, C++
  - Non blocking MPI
  - Asynchronous I/O
ACES 3 coding

- Write algorithm in high level **super instruction assembly language**
  - Declare (block) arrays, (block) indices
  - DO - END DO construct
  - PARDO – END PARDO construct
  - Basic operations: add and multiply and contract
  - Each line maps to a few **super instructions**
Optimize and tune ACES 3

- Optimize with traditional techniques
  - Optimize the basic contraction operations by mapping them to DGEMM calls
  - Create fast integral block code
  - Optimize memory allocation by using multiple block stacks
  - Optimize execution and data movement
Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- How does it work in parallel?
  - Computer science and engineering
- Some examples
  - Performance analysis
Some tests

- Spin unrestricted SCF and CCSD
  - H$_2$O 115 functions, 5 occupied
  - CH$_2$F$_2$ 116 functions 13 occupied
  - C$_6$H$_4$F$_2$ 140 functions 29 occupied
  - Ar$_4$ 200 functions 36 occupied
  - Ar$_6$ 300 functions 54 occupied
  - Ar$_{10}$ 500 functions 90 occupied
## Water

<table>
<thead>
<tr>
<th></th>
<th>Distrib AO</th>
<th>Distrib MO</th>
<th>Served AO</th>
<th>Served MO</th>
<th>Serial MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral transform</td>
<td>159 1</td>
<td>1,977 6/2</td>
<td>158 1</td>
<td>2,307 2/2</td>
<td>829</td>
</tr>
<tr>
<td>Total w/ o SCF</td>
<td>3,022 1</td>
<td>3,500 8</td>
<td>3,330 1</td>
<td>12,258 2/2</td>
<td>3,257</td>
</tr>
</tbody>
</table>
### CH$_2$F$_2$

<table>
<thead>
<tr>
<th></th>
<th>Distr AO</th>
<th>Distr MO</th>
<th>Served AO</th>
<th>Served MO</th>
<th>serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment 25</td>
<td>323</td>
<td>5,204</td>
<td>298</td>
<td>1,745</td>
<td>1,201</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4,879</td>
<td>1</td>
<td>1,904</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3/1</td>
<td></td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>Integral transform</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total w/o SCF</td>
<td>12303</td>
<td>11,777</td>
<td>13,719</td>
<td>23,813</td>
<td>17,657</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>10,430</td>
<td>1</td>
<td>14,540</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3/1</td>
<td></td>
<td>1/1</td>
<td></td>
</tr>
</tbody>
</table>
\[ \text{C}_6\text{H}_4\text{F}_2 \]

<table>
<thead>
<tr>
<th>Method</th>
<th>CCSD</th>
<th>MO</th>
<th>CCSD</th>
<th>AO</th>
<th>CCSD</th>
<th>Geom</th>
<th>3 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MO</td>
<td>15,856</td>
<td>12</td>
<td>7,743</td>
<td>4,848</td>
<td>1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>32</td>
<td>.76</td>
<td>64</td>
<td>.61</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AO</td>
<td>35,278</td>
<td>8</td>
<td>10,687</td>
<td>6,294</td>
<td>1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>32</td>
<td>.82</td>
<td>64</td>
<td>.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Geom</td>
<td>255,976</td>
<td>12</td>
<td>211,564</td>
<td>140,424</td>
<td>1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 steps</td>
<td>16</td>
<td>.91</td>
<td>32</td>
<td>.68</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Ar₆ $54+246=300$ bf on 64 processors

<table>
<thead>
<tr>
<th>Machine</th>
<th>SCF</th>
<th>trans</th>
<th>CCSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM P4 shelton</td>
<td>313 s</td>
<td>4,242 s</td>
<td>16,363 s 4.5 h</td>
</tr>
<tr>
<td>Cray X1 diamond</td>
<td>582 s</td>
<td>6,452 s</td>
<td>19,601 s 5.4 h</td>
</tr>
<tr>
<td>Compaq emerald</td>
<td>132 s</td>
<td>4,180 s</td>
<td>29,188 s 8.1 h</td>
</tr>
</tbody>
</table>
Conclusion

- New design and team work delivered
- On time, within budget
- New code is fast and flexible
- New code provide new tool to do new chemistry that cannot be done without it...