Quantum chemistry towards exascale with QMC=Chem

A. Scemama, M. Caffarel
Laboratoire de Chimie et Physique Quantiques
CNRS - IRSAMC
Université de Toulouse, France

E. Oseret, W. Jalby
Exascale Computing Research Laboratory
GENCI-CEA-Intel
Université de Versailles St Quentin, France
Ab initio Quantum chemistry

- Many chemical problems need highly accurate models, using an *ab initio* quantum mechanical description (Configuration Interaction or Coupled Cluster methods).

- Due to their iterative nature, synchronizations limit the parallel efficiency.

- These methods are not yet suited to massively parallel machines.

- Such methods need a large amount of memory and disk space.
Quantum Monte Carlo

Quantum Monte Carlo (QMC) are methods that:

- require a small amount of memory (~100MB per core)
- make very few network communications
- have a much better scaling than standard methods with the size of the chemical system

but:
- require a large amount of CPU time

The evolution of massively parallel machines is very favorable to QMC methods.
QMC methods are good candidates for exascale

During the run, each process is completely autonomous (single core processes). The scaling is ideal!

Communications are mandatory only at the initialization and the finalization stages.

The initialization and finalization times don't depend on the length of the run.
QMC=Chem key points

- All the processes are completely independent
- Additional compute nodes can be added/removed dynamically to a running simulation
- Fault tolerance: any compute node can fail without killing the whole run
- I/O and network communications are fully asynchronous
- An almost ideal scaling with the number of cores is obtained
- Very good single core performance
Parallel efficiency of QMC=Chem

Benchmarks performed on Curie (TGCC/CEA/GENCI, France) in April 2011.
The blue curve is estimated from the data collected on the green curve.

The performance of the application is determined by the efficiency of the single-core executable.
Single-core performance

The scaling of one Monte Carlo step is limited by
- A matrix inversion, via the Intel MKL library ($O(N^3)$)
- Matrix-matrix products using a sparse-dense implementation ($O(N^2)$)

Two approaches to optimize the matrix product (ECR contribution)

- Static Analysis with MAQAO: Disassemble the binary and give information on the inner-most loops
- Decremental Analysis with DECAN: remove FP instructions or memory instructions from the binary and compare the timings with the real binary
Dense x Sparse Matrix multiplication
Static Analysis

!DIR$ VECTOR ALIGNED
do j=1,LDC
  C1(j,i)=C1(j,i)+ (A(j,k_vec(1))\*d11 &
                  + A(j,k_vec(2))\*d21 &
                  + A(j,k_vec(3))\*d31 &
                  + A(j,k_vec(4))\*d41)
  C2(j,i)=C2(j,i)+ (A(j,k_vec(1))\*d12 &
                  + A(j,k_vec(2))\*d22 &
                  + A(j,k_vec(3))\*d32 &
                  + A(j,k_vec(4))\*d42)
enddo

- Examine the two hottest loops with MAQAO
- FLOP/cycle was not optimal : 12.8 but could be 16 (AVX, 32 bits elements, perfect ADD / MUL balance)
- Loop count (LDC) is always a multiple of 8. Replacing loop count with its hard coded value allows the compiler to factor loads
- We obtained a theoretically perfect efficiency for these loops

MAQAO Static analysis before (top) and after (bottom) optimization
<table>
<thead>
<tr>
<th>Function</th>
<th>Loop weight</th>
<th>original</th>
<th>MI</th>
<th>FP</th>
<th>MAQAO asymptotic</th>
<th>Speed-up over original</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MI</td>
</tr>
<tr>
<td>sparse_full_mm5_</td>
<td>20.0%</td>
<td>517</td>
<td>449</td>
<td>289</td>
<td>224</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>14.0%</td>
<td>389</td>
<td>285</td>
<td>289</td>
<td>224</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>6.7%</td>
<td>241</td>
<td>205</td>
<td>201</td>
<td>140</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>1.1%</td>
<td>1353</td>
<td>1197</td>
<td>33</td>
<td>280</td>
<td>1.1</td>
</tr>
<tr>
<td>bld_ao_oned_bloc_</td>
<td>2.9%</td>
<td>109</td>
<td>69</td>
<td>57</td>
<td>112</td>
<td>1.6</td>
</tr>
<tr>
<td>provide_elec_dist_</td>
<td>1.4%</td>
<td>1801</td>
<td>1775</td>
<td>221</td>
<td>324</td>
<td>1.0</td>
</tr>
<tr>
<td>provideAo_value_block_</td>
<td>1.2%</td>
<td>1213</td>
<td>1217</td>
<td>277</td>
<td>248</td>
<td>1.0</td>
</tr>
<tr>
<td>Invert (MKL)</td>
<td>16%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

Fully vectorized loops
DECAN: results on QMC==Chem

- FP matching MI matching original (3 loops out of 7)
  - Code balanced, no clear bottleneck, no obvious optimization
- MI matching original (3 loops out of 7)
  - Loop dominated by memory accesses
- MAQAO Asymptotic match FP (4 loops out of 7)
  - No short vector issue
- Measurement accuracy challenged
  - 1 loop out of 7 is less than 100 cycles per instance
- Computational efficiency
  - MKL “invert” 54%
  - 3 hand written loops are 99%, 75% and 53%
Single-core performance

The scaling of one Monte Carlo step is limited by

- A matrix inversion, via the Intel MKL library \(O(N^3)\)
- Matrix-matrix multiplications using an efficient sparse-dense implementation \(O(N^2)\)

<table>
<thead>
<tr>
<th>Molecular System</th>
<th>Number of electrons</th>
<th>% Peak performance of Multiplication</th>
<th>% Peak performance of Inversion</th>
<th>time(Inversion)/time(Multiplication)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Molecule 1" /></td>
<td>158</td>
<td>50%</td>
<td>24%</td>
<td>0.5</td>
</tr>
<tr>
<td><img src="image" alt="Molecule 2" /></td>
<td>434</td>
<td>64%</td>
<td>53%</td>
<td>0.6</td>
</tr>
<tr>
<td><img src="image" alt="Molecule 3" /></td>
<td>1056</td>
<td>58%</td>
<td>68%</td>
<td>2.0</td>
</tr>
<tr>
<td><img src="image" alt="Molecule 4" /></td>
<td>1731</td>
<td>53%</td>
<td>68%</td>
<td>2.6</td>
</tr>
</tbody>
</table>
## Overall single-core performance

<table>
<thead>
<tr>
<th>Molecular System</th>
<th>Number of electrons</th>
<th>RAM/core (MB)</th>
<th>CPU time / step on Core2(^1) (s)</th>
<th>CPU time / step on Sandy Bridge(^2) (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Molecule 1" /></td>
<td>158</td>
<td>9.8</td>
<td>0.0073</td>
<td>0.0033</td>
</tr>
<tr>
<td><img src="image2" alt="Molecule 2" /></td>
<td>434</td>
<td>65</td>
<td>0.0504</td>
<td>0.0186</td>
</tr>
<tr>
<td><img src="image3" alt="Molecule 3" /></td>
<td>1056</td>
<td>133</td>
<td>0.3421</td>
<td>0.0980</td>
</tr>
<tr>
<td><img src="image4" alt="Molecule 4" /></td>
<td>1731</td>
<td>313</td>
<td>1.2480</td>
<td>0.4226</td>
</tr>
</tbody>
</table>

1. Intel Xeon 5140, Core 2 2.33GHz, Dual core, 4MB shared L2 cache  
2. Intel Xeon E31240, Sandy Bridge 3.30GHz, Quad core, 256KB L2 cache/core, 8MB shared L3 cache
## Overall single-core performance

<table>
<thead>
<tr>
<th>Molecular System</th>
<th>Number of electrons</th>
<th>RAM/core (MB)</th>
<th>% CPU Peak performance on Core2²</th>
<th>% CPU Peak performance on Sandy Bridge²</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Molecule 1" /></td>
<td>158</td>
<td>9.8</td>
<td>25%</td>
<td>23%</td>
</tr>
<tr>
<td><img src="image2.png" alt="Molecule 2" /></td>
<td>434</td>
<td>65</td>
<td>34%</td>
<td>38%</td>
</tr>
<tr>
<td><img src="image3.png" alt="Molecule 3" /></td>
<td>1056</td>
<td>133</td>
<td>37%</td>
<td>49%</td>
</tr>
<tr>
<td><img src="image4.png" alt="Molecule 4" /></td>
<td>1731</td>
<td>313</td>
<td>47%</td>
<td>55%</td>
</tr>
</tbody>
</table>

1. Intel Xeon 5140, Core 2 2.33GHz, Dual core, 4MB shared L2 cache
2. Intel Xeon E31240, Sandy Bridge 3.30GHz, Quad core, 256KB L2 cache/core, 8MB shared L3 cache
Latest performance results

We have modelled a peptide involved in Alzheimer's disease on a BullX supercomputer:

- Intel Sandy Bridge sockets, 2.7GHz, 8 cores, 20MB cache
- 190 dual socket nodes (3040 cores)
- 64GB RAM/node

An average of 27.8 TFlops/s was obtained on a 7 minutes run.
For a 1 hour run we would obtain 31.2 TFlops/s.
We would be able to reach 1PFlops/s with 97 500 cores for 1 hour.