Software optimization for petaflops/s scale Quantum Monte Carlo simulations

A. Scemama\textsuperscript{1}, M. Caffarel\textsuperscript{1}, E. Oseret\textsuperscript{2}, W. Jalby\textsuperscript{2}

\textsuperscript{1}Laboratoire de Chimie et Physique Quantiques / IRSAMC, Toulouse, France
\textsuperscript{2}Exascale Computing Research / Intel, CEA, GENCI, UVSQ Versailles, France

4 Dec 2012
Outline

1. Quantum Monte Carlo
2. The QMC=Chem code
Quantum Monte Carlo methods

- Solve the Schrödinger equation with random walks
- State-of-the-art and routine approaches in physics: nuclear physics, condensed-matter, spin systems, quantum liquids, infrared spectroscopy...
- Still of confidential use for the electronic structure problem of quantum chemistry (as opposed to post-HF and DFT)
- Reason: Very high computational cost for small/medium systems

But:

- Very favorable scaling with system size compared to standard methods
- Ideally suited to extreme parallelism
Quantum Monte Carlo methods

- Solve the Schrödinger equation with random walks
- State-of-the-art and routine approaches in physics: nuclear physics, condensed-matter, spin systems, quantum liquids, infrared spectroscopy...
- Still of confidential use for the electronic structure problem of quantum chemistry (as opposed to post-HF and DFT)
- Reason: Very high computational cost for small/medium systems

But:
- Very favorable scaling with system size compared to standard methods
- Ideally suited to extreme parallelism
Quantum Monte Carlo methods

- Solve the Schrödinger equation with random walks
- Still of confidential use for the electronic structure problem of quantum chemistry (as opposed to post-HF and DFT)
- Reason: Very high computational cost for small/medium systems

But:
- Very favorable scaling with system size compared to standard methods
- Ideally suited to extreme parallelism
Quantum Monte Carlo methods

- Solve the Schrödinger equation with random walks
- Still of confidential use for the electronic structure problem of quantum chemistry (as opposed to post-HF and DFT)
- Reason: Very high computational cost for small/medium systems

But:
- Very favorable scaling with system size compared to standard methods
- Ideally suited to extreme parallelism
Quantum Monte Carlo methods

- Solve the Schrödinger equation with random walks
- Still of confidential use for the electronic structure problem of quantum chemistry (as opposed to post-HF and DFT)
- Reason: Very high computational cost for small/medium systems

But:
- Very favorable scaling with system size compared to standard methods
- Ideally suited to extreme parallelism
Problem: Solve stochastically the Schrödinger equation for $N$ electrons in a molecule

$$E = \frac{\int dr_1 \ldots dr_N \Phi(r_1, \ldots, r_N) \mathcal{H} \Phi(r_1, \ldots, r_N)}{\int dr_1 \ldots dr_N \Phi(r_1, \ldots, r_N) \Phi(r_1, \ldots, r_N)}$$

$$\sim \sum \frac{\mathcal{H} \Psi(r_1, \ldots, r_N)}{\Psi(r_1, \ldots, r_N)}, \text{ sampled with } (\Psi \times \Phi)$$

$\mathcal{H}$: Hamiltonian operator
$E$: Energy
$\Psi$: Trial wave function
$\Phi$: Exact wave function
$r_1, \ldots, r_N$: Electron coordinates
QMC in a few words

- **Walker** = 3N-dimensional vector containing the **positions of the** \( N \) **electrons**

- **Stochastic trajectories for walkers** (or set of electrons)
  - To impose **importance sampling** we need of an **approximate computable trial wavefunction** \( \Psi_T \) which helps to drive the electronic trajectories into the important regions
  - To get chemical properties, **averages are computed along electronic trajectories**
  - **Extreme parallelism**: Independent populations of walkers (no communications) can be distributed on different CPUs
**QMC in a few words**

- **Walker** = 3N-dimensional vector containing the positions of the $N$ electrons
- **Stochastic trajectories for walkers** (or set of electrons)
- To impose importance sampling we need of an approximate computable trial wavefunction $\Psi_T$ which helps to drive the electronic trajectories into the important regions
- To get chemical properties, averages are computed along electronic trajectories
- **Extreme parallelism** : Independent populations of walkers (no communications) can be distributed on different CPUs
QMC in a few words

- **Walker** = 3N-dimensional vector containing the positions of the $N$ electrons
- **Stochastic trajectories for walkers** (or set of electrons)
- To impose **importance sampling** we need of an approximate computable trial wavefunction $\Psi_T$ which helps to drive the electronic trajectories into the important regions
- To get chemical properties, **averages are computed along electronic trajectories**
- **Extreme parallelism**: Independent populations of walkers (no communications) can be distributed on different CPUs
**QMC in a few words**

- **Walker** = 3N-dimensional vector containing the positions of the $N$ electrons
- **Stochastic trajectories for walkers** (or set of electrons)
- To impose **importance sampling** we need of an approximate computable trial wavefunction $\Psi_T$ which helps to drive the electronic trajectories into the important regions
- To get chemical properties, **averages are computed along electronic trajectories**
- **Extreme parallelism** : Independent populations of walkers (no communications) can be distributed on different CPUs
QMC Algorithm

Input:
\{R_{\text{nucl}}\}, N_{\alpha}, N_{\beta}
\Psi_T(\vec{r}_1, \ldots, \vec{r}_N)

Stochastic dynamics of electrons:
\[
\begin{align*}
\Psi_T(\vec{r}_1, \ldots, \vec{r}_N) \\
\nabla \Psi_T(\vec{r}_1, \ldots, \vec{r}_N) \\
\nabla^2 \Psi_T(\vec{r}_1, \ldots, \vec{r}_N)
\end{align*}
\]

Expectation values:
\[
\begin{align*}
E_0(R_{\text{nucl}}), \ \Delta E_0(R_{\text{nucl}}) \\
\n\nabla E(R_{\text{nucl}}), \ \rho(\vec{r}), \ \ldots
\end{align*}
\]
Amyloid $\beta$ peptide simulation on CURIE machine (GENCI/TGCC/CEA, France)

First step of our scientific project on Alzheimer disease: Energy difference of the $\beta$-strand and the $\alpha$-helix conformations of $A_\beta$(28-35) (a 1302-dimensional PDE to solve!!)

⇒ SUSTAINED 960 TFlops/s on 76 800 cores of CURIE
1. Quantum Monte Carlo

2. The QMC=Chem code
Block: $N_{\text{walk}}$ walkers executing $N_{\text{step}}$ steps

- Compute as many blocks as possible, as quickly as possible
- Block are independent: block averages have a Gaussian distribution
Block: $N_{\text{walk}}$ walkers executing $N_{\text{step}}$ steps

Compute as many blocks as possible, as quickly as possible

Blocks are independent: block averages have a Gaussian distribution
Block: \( N_{\text{walk}} \) walkers executing \( N_{\text{step}} \) steps
- Compute as many blocks as possible, as quickly as possible
- Block are independent: block averages have a Gaussian distribution
All I/O and network communications are asynchronous.
Very small memory footprint (10—300 MiB/core)
Fault-tolerance

Extreme parallelism \(\rightarrow\) possible system failures
- Blocks are Gaussian \(\rightarrow\) losing blocks doesn’t change the average
- Simulation survives to removal of any node
- Restart always possible from data base
Almost ideal scaling \(\rightarrow\) single-core optimization is crucial.
Hot-spots in a Monte Carlo step

1. **Matrix inversion** $\mathcal{O}(N^3)$ (DP, Intel MKL)

2. **Sparse $\times$ dense matrix products** $\mathcal{O}(N^2)$ (SP, our implementation)

Multiply one dense matrix $A^{N \times N_{\text{basis}}}$ with 5 sparse matrices $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{\text{elec}}}$ with the same non-zero pattern to produce 5 dense matrices $\{C_1, \ldots, C_5\}^{N \times N_{\text{elec}}}$. 

- $N_{\text{basis}}$: number of basis functions, $N_{\text{elec}}$: number of electrons
- $N_{\text{basis}} \sim 5 \times N_{\text{elec}}$
- $N = 2 \times N_{\text{elec}}$
Hot-spots in a Monte Carlo step

1. Matrix inversion $O(N^3)$ (DP, Intel MKL)
2. Sparse $\times$ dense matrix products $O(N^2)$ (SP, our implementation)

Multiply one dense matrix $A^{N \times N_{\text{basis}}}$

with 5 sparse matrices $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{\text{elec}}}$ with the same non-zero pattern

to produce 5 dense matrices $\{C_1, \ldots, C_5\}^{N \times N_{\text{elec}}}$.

$N_{\text{basis}}$ : number of basis functions, $N_{\text{elec}}$ : number of electrons

$N_{\text{basis}} \sim 5 \times N_{\text{elec}}$, 

$N = 2 \times N_{\text{elec}}$
Hot-spots in a Monte Carlo step

1. Matrix inversion $O(N^3)$ (DP, Intel MKL)
2. Sparse $\times$ dense matrix products $O(N^2)$ (SP, our implementation)

Multiply one dense matrix $A^{N \times N_{\text{basis}}}$ with 5 sparse matrices $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{\text{elec}}}$ with the same non-zero pattern to produce 5 dense matrices $\{C_1, \ldots, C_5\}^{N \times N_{\text{elec}}}$.

$N_{\text{basis}}$ : number of basis functions, $N_{\text{elec}}$ : number of electrons
$N_{\text{basis}} \sim 5 \times N_{\text{elec}}$, $N = 2 \times N_{\text{elec}}$
Hot-spots in a Monte Carlo step

1. Matrix inversion $O(N^3)$ (DP, Intel MKL)
2. Sparse $\times$ dense matrix products $O(N^2)$ (SP, our implementation)

Multiply one dense matrix $A^{N \times N_{basis}}$ with 5 sparse matrices $\{B_1, \ldots, B_5\}^{N_{basis} \times N_{elec}}$ with the same non-zero pattern to produce 5 dense matrices $\{C_1, \ldots, C_5\}^{N \times N_{elec}}$.

$N_{basis}$: number of basis functions, $N_{elec}$: number of electrons

$N_{basis} \sim 5 \times N_{elec}$,

$N = 2 \times N_{elec}$


Hot-spots in a Monte Carlo step

1. Matrix inversion $O(N^3)$ (DP, Intel MKL)
2. Sparse $\times$ dense matrix products $O(N^2)$ (SP, our implementation)

Multiply one dense matrix $A^{N \times N_{\text{basis}}}$ with 5 sparse matrices $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{\text{elec}}}$ with the same non-zero pattern to produce 5 dense matrices $\{C_1, \ldots, C_5\}^{N \times N_{\text{elec}}}$.

- $N_{\text{basis}}$: number of basis functions
- $N_{\text{elec}}$: number of electrons

$N_{\text{basis}} \sim 5 \times N_{\text{elec}}$, $N = 2 \times N_{\text{elec}}$
Choose loop order that permits vectorization

\[ C_1 = 0.; \quad C_2 = 0.; \quad C_3 = 0.; \quad C_4 = 0.; \quad C_5 = 0. \]

\[
do\ \text{i=1, Number of electrons}
    \do\ \text{k=1, Number of non-zero AOs for electron i}
        \do\ \text{j=1, Number of molecular orbitals}
            \begin{align*}
                C_1(j,i) &\quad +=\quad A(j,\text{indices}(k,i)) \times B_1(k,i) \\
                C_2(j,i) &\quad +=\quad A(j,\text{indices}(k,i)) \times B_2(k,i) \\
                C_3(j,i) &\quad +=\quad A(j,\text{indices}(k,i)) \times B_3(k,i) \\
                C_4(j,i) &\quad +=\quad A(j,\text{indices}(k,i)) \times B_4(k,i) \\
                C_5(j,i) &\quad +=\quad A(j,\text{indices}(k,i)) \times B_5(k,i)
            \end{align*}
        \end{do}
    \end{do}
\end{do}
Hand-written optimizations for Sandy Bridge architecture:

- All arrays are 256-bit aligned (compiler directives)
- LDA are multiples of 8 (Single precision, 256 bit AVX)
- Unroll and jam to reduce nb of stores
- Loop distribution to avoid register spilling
- Blocking/sorting to reduce cache misses in access to $A$
Sparse-dense matrix products

Hand-written optimizations for Sandy Bridge architecture:
- All arrays are 256-bit aligned (compiler directives)
- LDA are multiples of 8 (Single precision, 256 bit AVX)
- Unroll and jam to reduce nb of stores
- Loop distribution to avoid register spilling
- Blocking/sorting to reduce cache misses in access to $A$
Sparse-dense matrix products

Hand-written optimizations for Sandy Bridge architecture:

- All arrays are 256-bit aligned (compiler directives)
- LDA are multiples of 8 (Single precision, 256 bit AVX)
- Unroll and jam to reduce nb of stores
- Loop distribution to avoid register spilling
- Blocking/sorting to reduce cache misses in access to \( A \)
Hand-written optimizations for Sandy Bridge architecture:

- All arrays are 256-bit aligned (compiler directives)
- LDA are multiples of 8 (Single precision, 256 bit AVX)
- Unroll and jam to reduce nb of stores
- Loop distribution to avoid register spilling
- Blocking/sorting to reduce cache misses in access to A
Sparse-dense matrix products

Hand-written optimizations for Sandy Bridge architecture:
- All arrays are 256-bit aligned (compiler directives)
- LDA are multiples of 8 (Single precision, 256 bit AVX)
- Unroll and jam to reduce nb of stores
- Loop distribution to avoid register spilling
- Blocking/sorting to reduce cache misses in access to $A$
Sparse-dense matrix products

do i=1, Number of electrons
   do k=1, Number of non-zero AOs for electron i, 4

      do j=1, Number of molecular orbitals
         C1(j,i) += A(j,indice(k ,i)) * B1(k ,i) + A(j,indice(k+1,i)) * B1(k+1,i) 
         & + A(j,indice(k+2,i)) * B1(k+2,i) + A(j,indice(k+3,i)) * B1(k+3,i)
         C2(j,i) += A(j,indice(k ,i)) * B2(k ,i) + A(j,indice(k+1,i)) * B2(k+1,i) 
         & + A(j,indice(k+2,i)) * B2(k+2,i) + A(j,indice(k+3,i)) * B2(k+3,i)
      end do

      do j=1, Number of molecular orbitals
         C3(j,i) += A(j,indice(k ,i)) * B3(k ,i) + A(j,indice(k+1,i)) * B3(k+1,i) 
         & + A(j,indice(k+2,i)) * B3(k+2,i) + A(j,indice(k+3,i)) * B3(k+3,i)
         C4(j,i) += A(j,indice(k ,i)) * B4(k ,i) + A(j,indice(k+1,i)) * B4(k+1,i) 
         & + A(j,indice(k+2,i)) * B4(k+2,i) + A(j,indice(k+3,i)) * B4(k+3,i)
      end do

      do j=1, Number of molecular orbitals  ! Unrolled 2x by compiler
         C5(j,i) += A(j,indice(k ,i)) * B5(k ,i) + A(j,indice(k+1,i)) * B5(k+1,i) 
         & + A(j,indice(k+2,i)) * B5(k+2,i) + A(j,indice(k+3,i)) * B5(k+3,i)
      end do

   end do

A. Scemama, M. Caffarel, E. Oseret, W. Jalby

Petascale QMC for chemistry
Sparse-dense matrix products

Efficiency of the matrix products:

- Static analysis (MAQAO): Full-AVX (no scalar operations), inner-most loops perform 16 flops/cycle
- Decremental analysis (DECAN): good balance between flops and memory operations
- Up to 64% of the peak measured on Xeon E5
Sparse-dense matrix products

Efficiency of the matrix products:

- **Static analysis (MAQAO):** Full-AVX (no scalar operations), inner-most loops perform 16 flops/cycle.

- **Decremental analysis (DECAN):** Good balance between flops and memory operations.

- Up to 64% of the peak measured on Xeon E5.
Efficiency of the matrix products:

- Static analysis (MAQAO): Full-AVX (no scalar operations), inner-most loops perform 16 flops/cycle
- Decremental analysis (DECAN): good balance between flops and memory operations
- Up to 64% of the peak measured on Xeon E5
**Table:** Single core performance (GFlops/s), % peak in parenthesis, Measured on Intel Xeon E31240, 3.30GHz (52.8 GFlops/s SP, 26.4 GFlops/s DP).

<table>
<thead>
<tr>
<th>$N_{\text{elec}}$</th>
<th>$N_{\text{basis}}$</th>
<th>Matrix inversion</th>
<th>Matrix products</th>
<th>Overall (1 core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>158</td>
<td>404</td>
<td>6.3 (24%)</td>
<td>26.6 (50%)</td>
<td>8.8 (23%)</td>
</tr>
<tr>
<td>434</td>
<td>963</td>
<td>14.0 (53%)</td>
<td>33.1 (63%)</td>
<td>11.8 (33%)</td>
</tr>
<tr>
<td>434</td>
<td>2934</td>
<td>14.0 (53%)</td>
<td>33.6 (64%)</td>
<td>13.7 (38%)</td>
</tr>
<tr>
<td>1056</td>
<td>2370</td>
<td>17.9 (67%)</td>
<td>30.6 (58%)</td>
<td>15.2 (49%)</td>
</tr>
<tr>
<td>1731</td>
<td>3892</td>
<td>17.8 (67%)</td>
<td>28.2 (53%)</td>
<td>16.2 (55%)</td>
</tr>
</tbody>
</table>
Acknowledgments:

- GENCI
- CALMIP
- CEA
- PRACE
- Intel
- Bull
- ANR