# Software optimization for petaflops/s scale Quantum Monte Carlo simulations

A. Scemama[1], M. Caffarel[1], E. Oseret[2], W. Jalby[2]

[1]Laboratoire de Chimie et Physique Quantiques / IRSAMC, Toulouse, France
[2]Exascale Computing Research / Intel, CEA, GENCI, UVSQ Versailles, France
http://qmcchem.ups-tlse.fr

4 Dec 2012

# Outline

# 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
- 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
- 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 for molecular systems

Problem : Solve stochastically the Schrödinger equation for $N$ electrons in a molecule

$$
\begin{aligned}
E &= \frac{\int \mathrm{d}\mathbf{r}_1 \ldots \mathrm{d}\mathbf{r}_N \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N) \mathcal{H} \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N)}{\int \mathrm{d}\mathbf{r}_1 \ldots \mathrm{d}\mathbf{r}_N \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N) \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N)} \\
&\sim \sum \frac{\mathcal{H}\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N)}{\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N)}, \text{ sampled with } (\Psi \times \Phi)
\end{aligned}
$$

$\mathcal{H}$ : Hamiltonian operator

$E$ : Energy

$\mathbf{r}_1, \ldots, \mathbf{r}_N$ : Electron coordinates

$\Phi$ : Exact wave function

$\Psi$ : Trial wave function

# 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:**

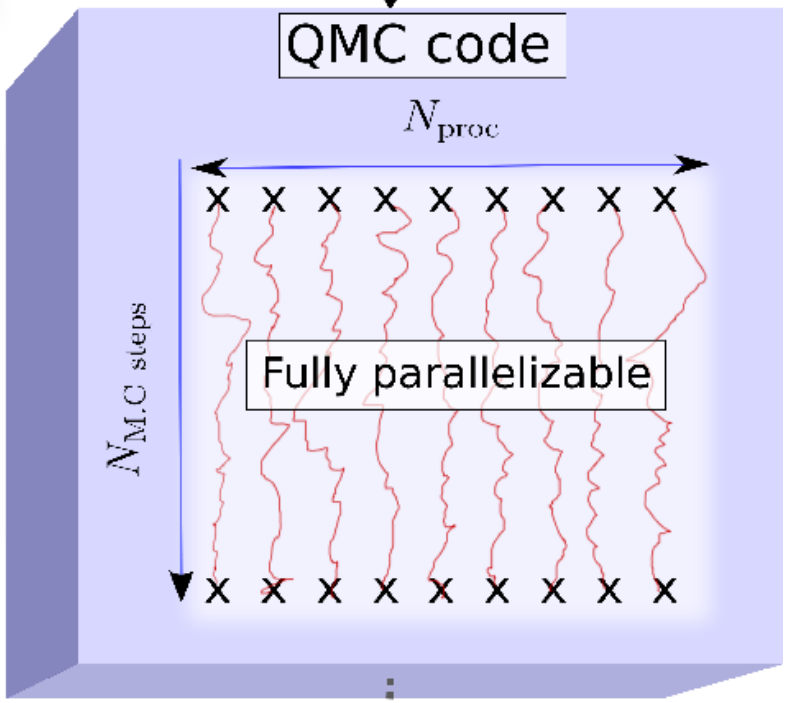
$\{\mathbf{R}_{\text{nucl}}\}, \mathbf{N}_\alpha, \mathbf{N}_\beta$
$\Psi_T(\vec{r}_1, \ldots, \vec{r}_N)$

**Stochastic dynamics of *electrons*:**

$$\begin{cases} \Psi_T(\vec{r}_1, \ldots, \vec{r}_N) \\ \vec{\nabla}\Psi_T(\vec{r}_1, \ldots, \vec{r}_N) \\ \vec{\nabla}^2\Psi_T(\vec{r}_1, \ldots, \vec{r}_N) \end{cases}$$

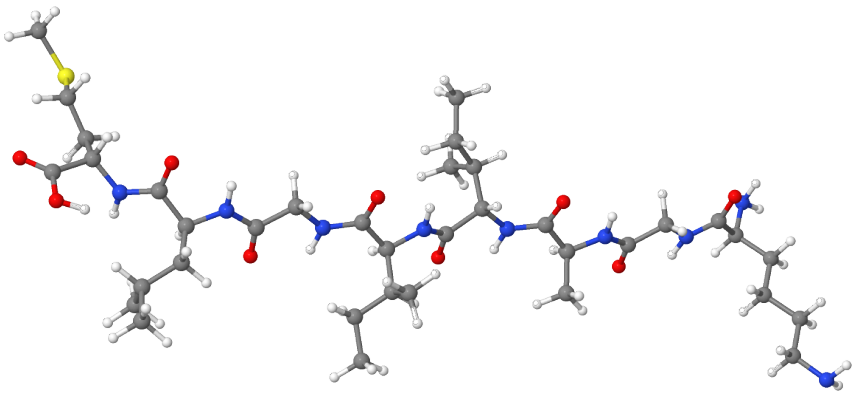
**Expectation values:**

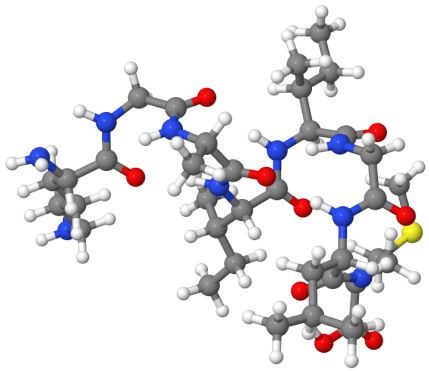
$E_0(\mathbf{R}_{\text{nucl}}), \quad \Delta E_0(\mathbf{R}_{\text{nucl}}),$
$\vec{\nabla} E(\mathbf{R}_{\text{nucl}}), \quad \rho(\vec{r}), \quad \ldots$



# 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 ! !)



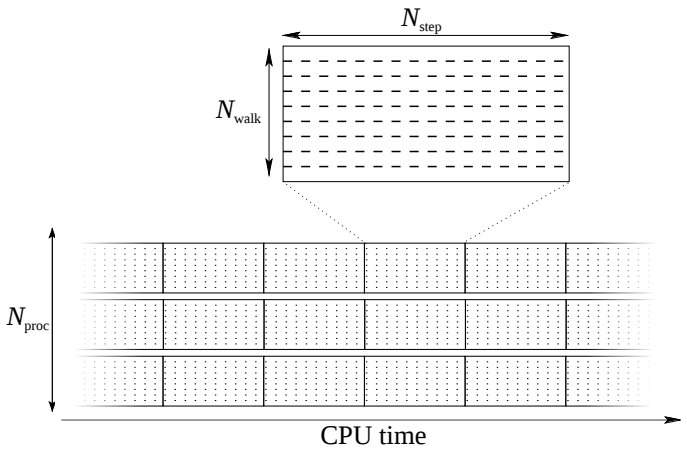



$\Rightarrow$ **SUSTAINED 960 TFlops/s on 76 800 cores of CURIE**

# Outline

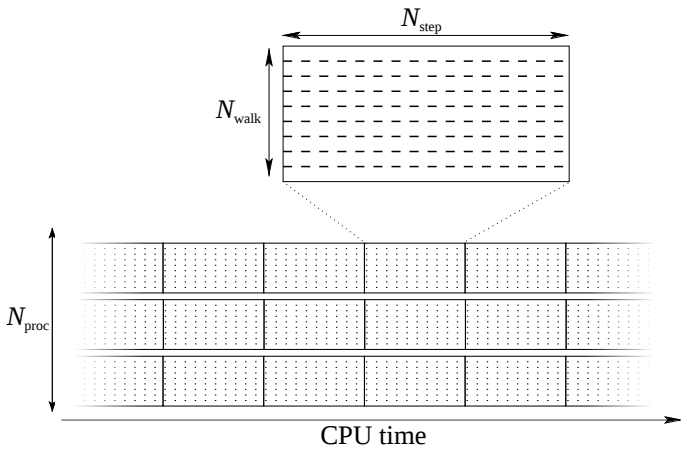
# Implementation in QMC=Chem

- 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



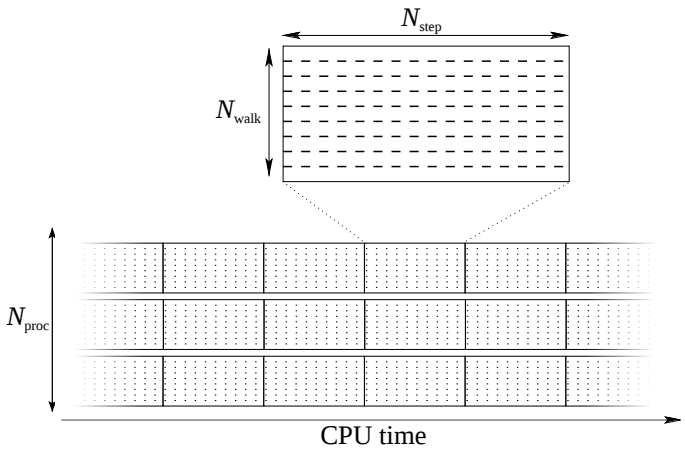
# Implementation in QMC=Chem

- 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



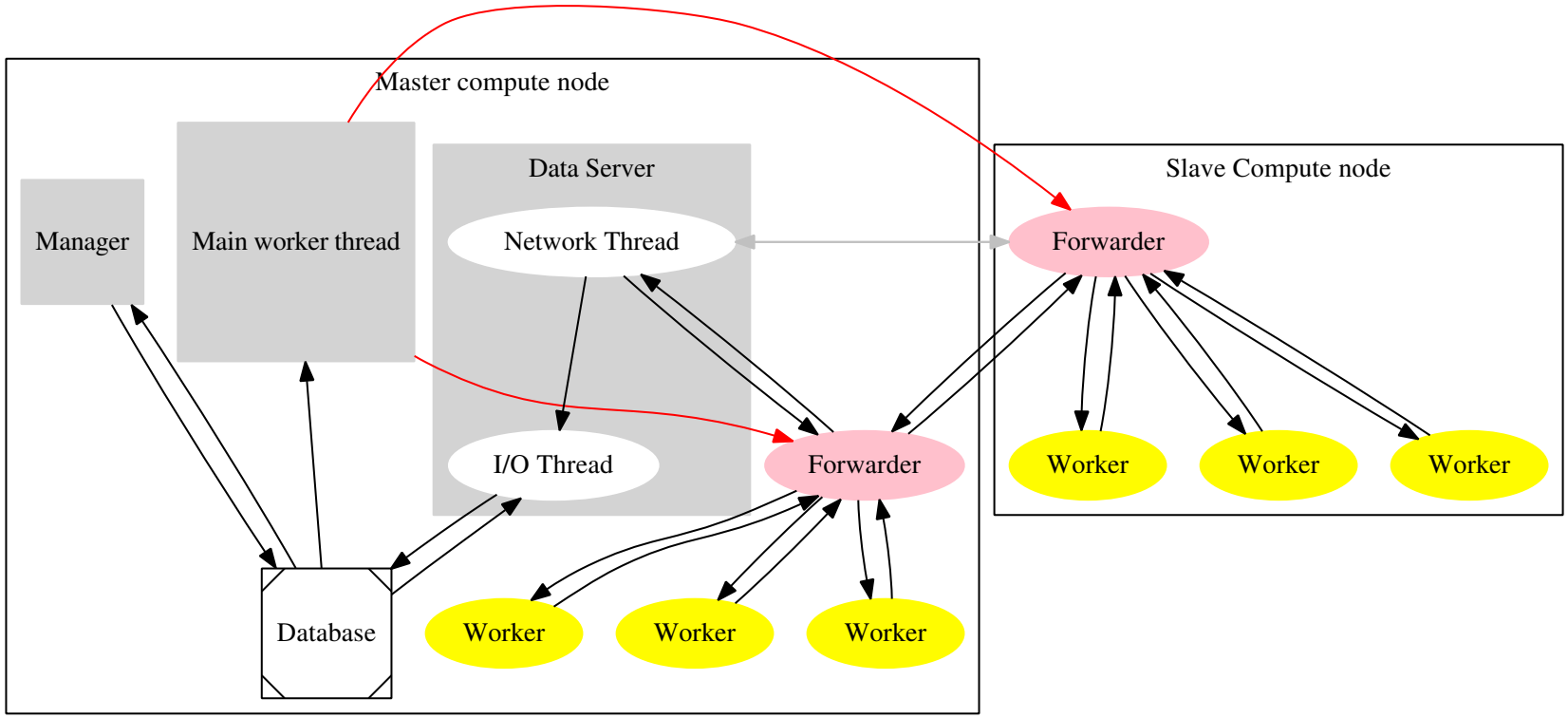
# Implementation in QMC=Chem

- 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



# Parallelism in QMC=Chem

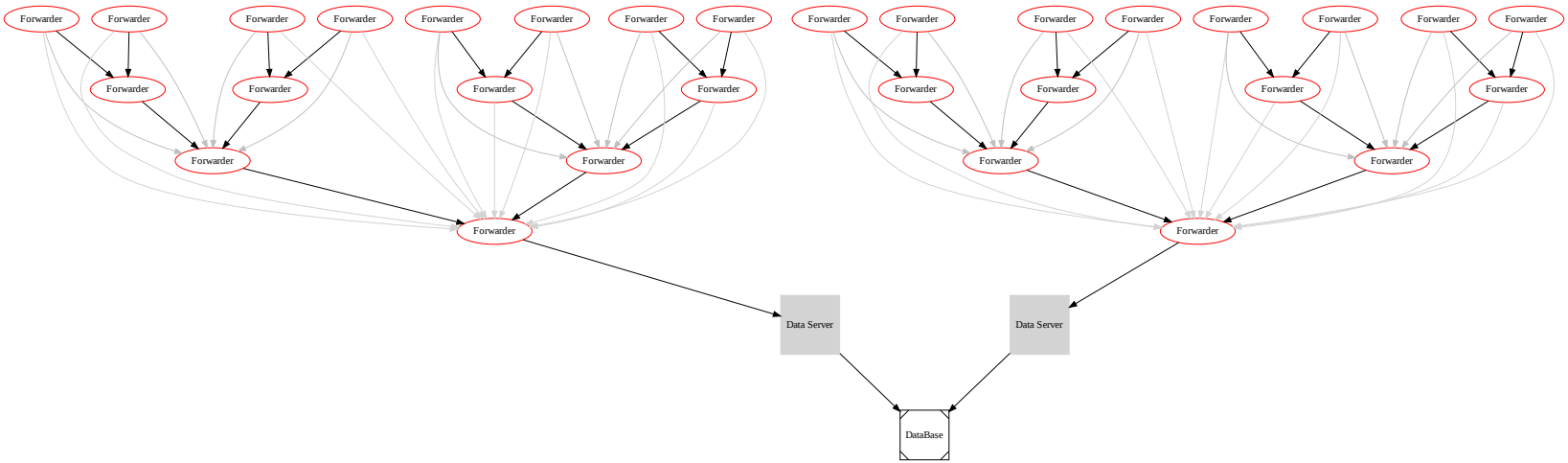
All I/O and network communications are asynchronous
Very small memory footprint (10—300 MiB/core)



# Fault-tolerance

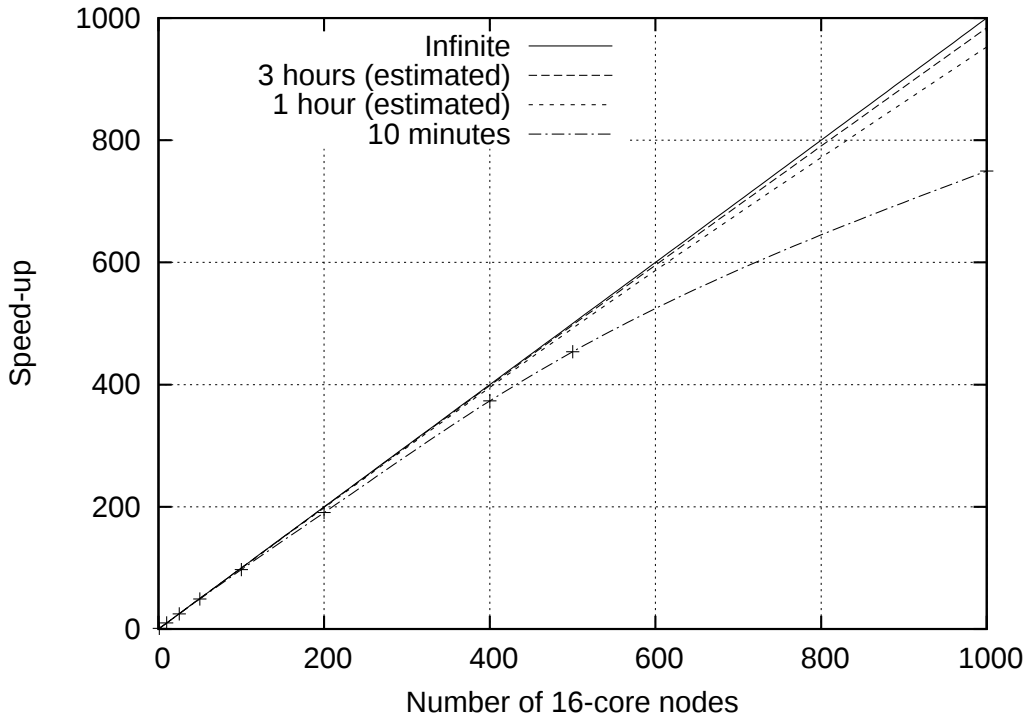
Extreme parallelism $\longrightarrow$ possible system failures

- Blocks are Gaussian $\rightarrow$ losing blocks doesn't change the average
- Simulation survives to removal of any node
- Restart always possible from data base



# QMC=Chem scaling

Almost ideal scaling $\longrightarrow$ 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 $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{elec}}$ 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 $\mathcal{O}(N^3)$ (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 $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{elec}}$ 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 $\mathcal{O}(N^3)$ (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 $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{elec}}$ 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 $\mathcal{O}(N^3)$ (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 $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{elec}}$ 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 $\mathcal{O}(N^3)$ (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 $\{B_1, \ldots, B_5\}^{N_{\text{basis}} \times N_{elec}}$ 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
$N_{\text{basis}} \sim 5 \times N_{\text{elec}}$,
$N = 2 \times N_{\text{elec}}$

# Sparse-dense matrix products

## Choose loop order that permits vectorization

```
C1 = 0. ; C2 = 0. ; C3 = 0. ; C4 = 0. ; C5 = 0.
do i=1, Number of electrons
  do k=1, Number of non-zero AOs for electron i
    do j=1, Number of molecular orbitals
      C1(j,i) += A(j,indices(k,i))*B1(k,i)
      C2(j,i) += A(j,indices(k,i))*B2(k,i)
      C3(j,i) += A(j,indices(k,i))*B3(k,i)
      C4(j,i) += A(j,indices(k,i))*B4(k,i)
      C5(j,i) += A(j,indices(k,i))*B5(k,i)
    end do
  end do
end do
```

# 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 $A$

# 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 $A$

# 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
end do
```

# 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

# 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

# Hot-spots in a Monte Carlo step

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).

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

Acknowkledgments :

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