QMC=Chem: a quantum Monte Carlo program for large-scale simulations in chemistry at the petascale level and beyond

M. Caffarel[1], A. Scemama[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

18 July 2012


Scientific and technological challenge

One of the grand challenges of modern science : Quantitative description of complex chemical systems

Important scientific and technological applications :

- the field of drug design (**pharmaceutical industry**)
- nano- and micro-electronics involving various devices at the molecular level (**electronics industry**)
- the field of new materials : carbon nanotubes, graphene, etc. (**materials industry**)
- various domains where catalysis plays a central role : enzymatic reactions in **biochemistry** and catalytic reactions in **petroleum industry**
- etc.

Computational chemistry

The mathematical problem to be solved is particularly tough !

N-electron Schrödinger equation = $3N$-Partial Differential Equation (PDE) with **N very large (thousand's...)**

Standard approaches of computational chemistry :

- **Density Functional Theory (DFT)** : Transform the PDE into a set of N reduced 3D equations describing electrons moving in a 3D-external potential.
- **Post-Hartree-Fock wavefunction approaches** : Keeps the full $3N$-dependence of the wave function and is based on Galerkin-type approaches using a finite gaussian basis set.

Computational chemistry II

- **Density Functional Theory (DFT)** :
  - Computational effort $O(N^3)$
  - Systematic error but loosely controlled error
- **Post-Hartree-Fock wavefunction approaches** :
  - Much more controlled error than DFT
  - Computational cost at least $O(N^5)$ but preferrably $O(N^6)$ or $O(N^7)$

Quantum Monte Carlo = Emerging alternative approach based on solving the Schrödinger equation using random walks.

Quantum Monte Carlo

- Computational cost $O(N^3)$ (CPU bound, not memory bound) : **GOOD !**
- Error much more controlled than DFT :**GOOD !**
- Price to pay : QMC is very CPU intensive : **BAD !**

BUT....

QMC ideally suited to extreme parallelism : becoming **GOOD** at the petascale level and beyond...

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** Ψ_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

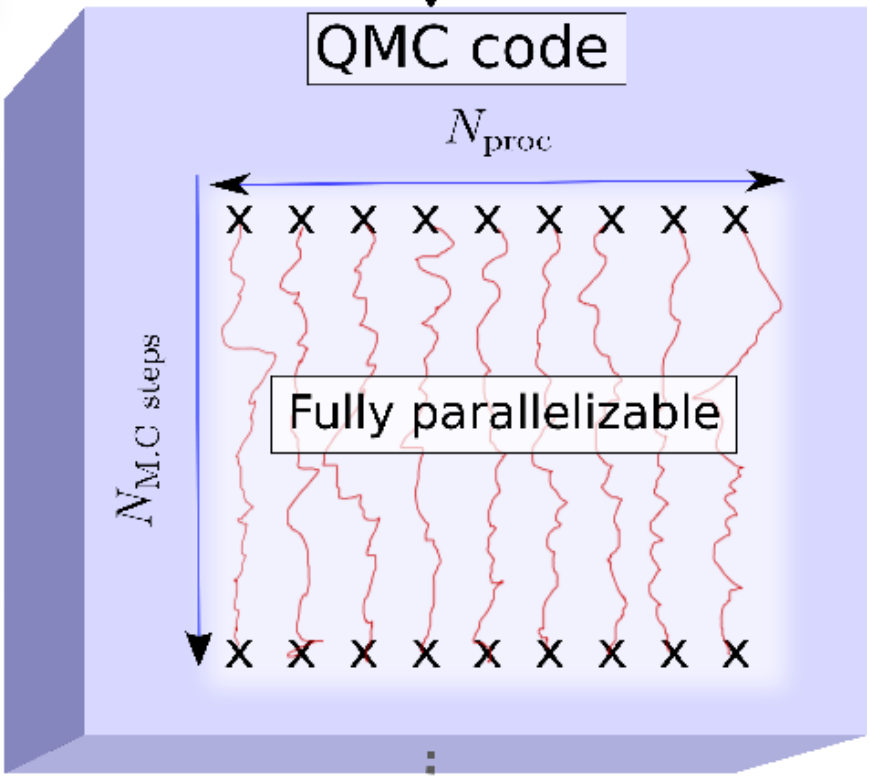
QMC Algorithm

Input:

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

Stochastic dynamics of *electrons*:

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

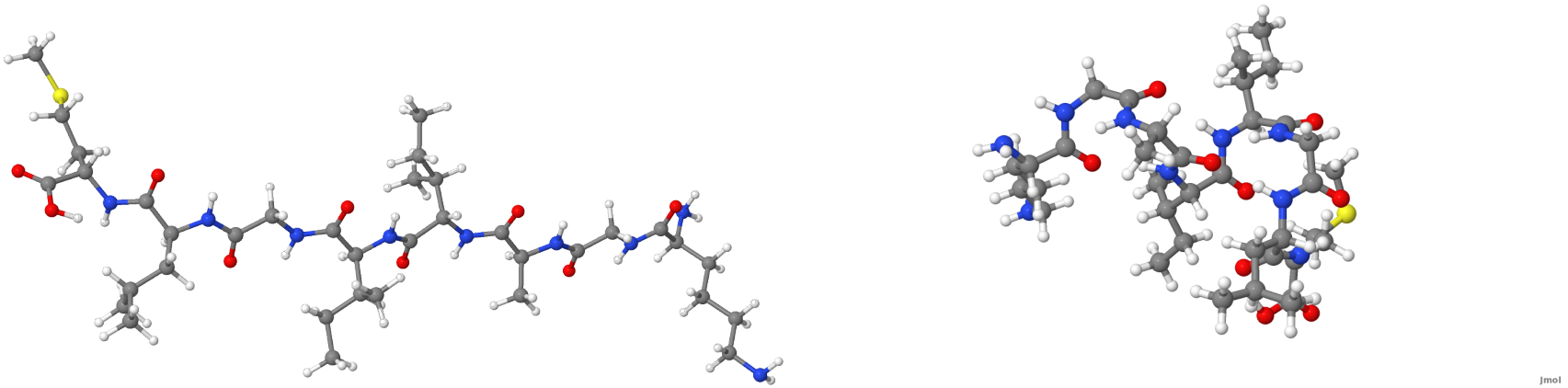


Expectation values:

$E_0(\mathbf{R}_{\text{nucl}}), \quad \Delta E_0(\mathbf{R}_{\text{nucl}}),$

Amyloid β peptide simulation on CURIE machine (GENCI/TGCC/CEA, France)

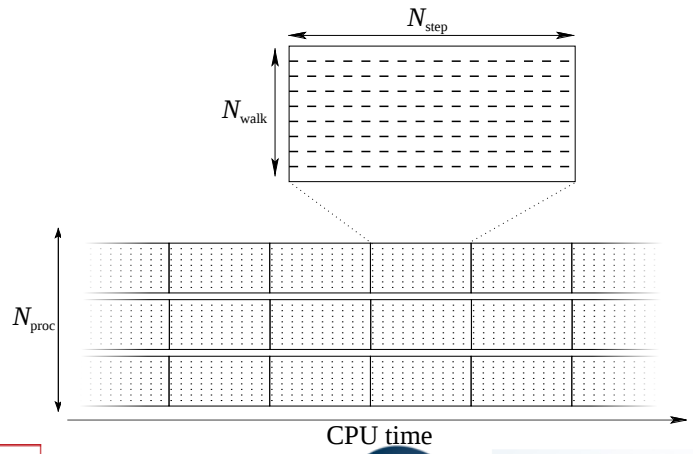
First step of our scientific project on Alzheimer disease : Energy difference of the β-strand and the α-helix conformations of Aβ(28-35) (a 1302-dimensional PDE to solve ! !)



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

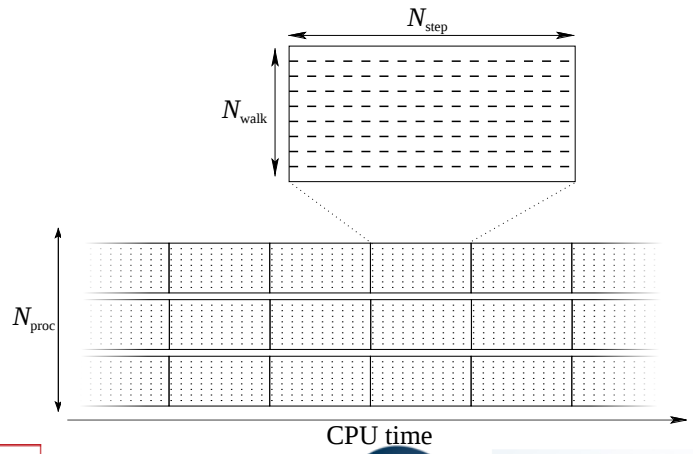
Implementation in QMC=Chem

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



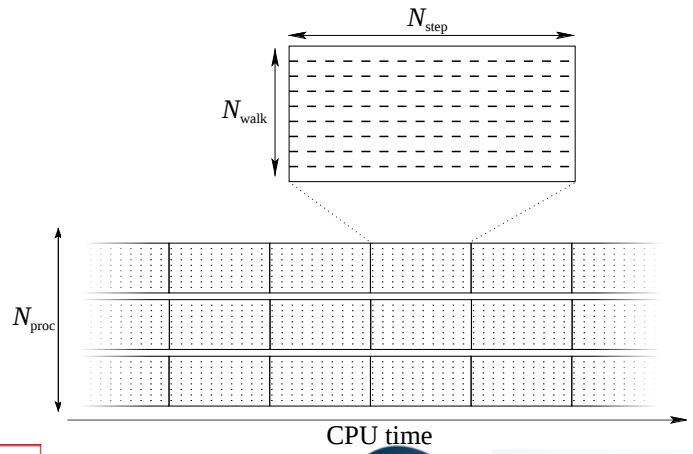
Implementation in QMC=Chem

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



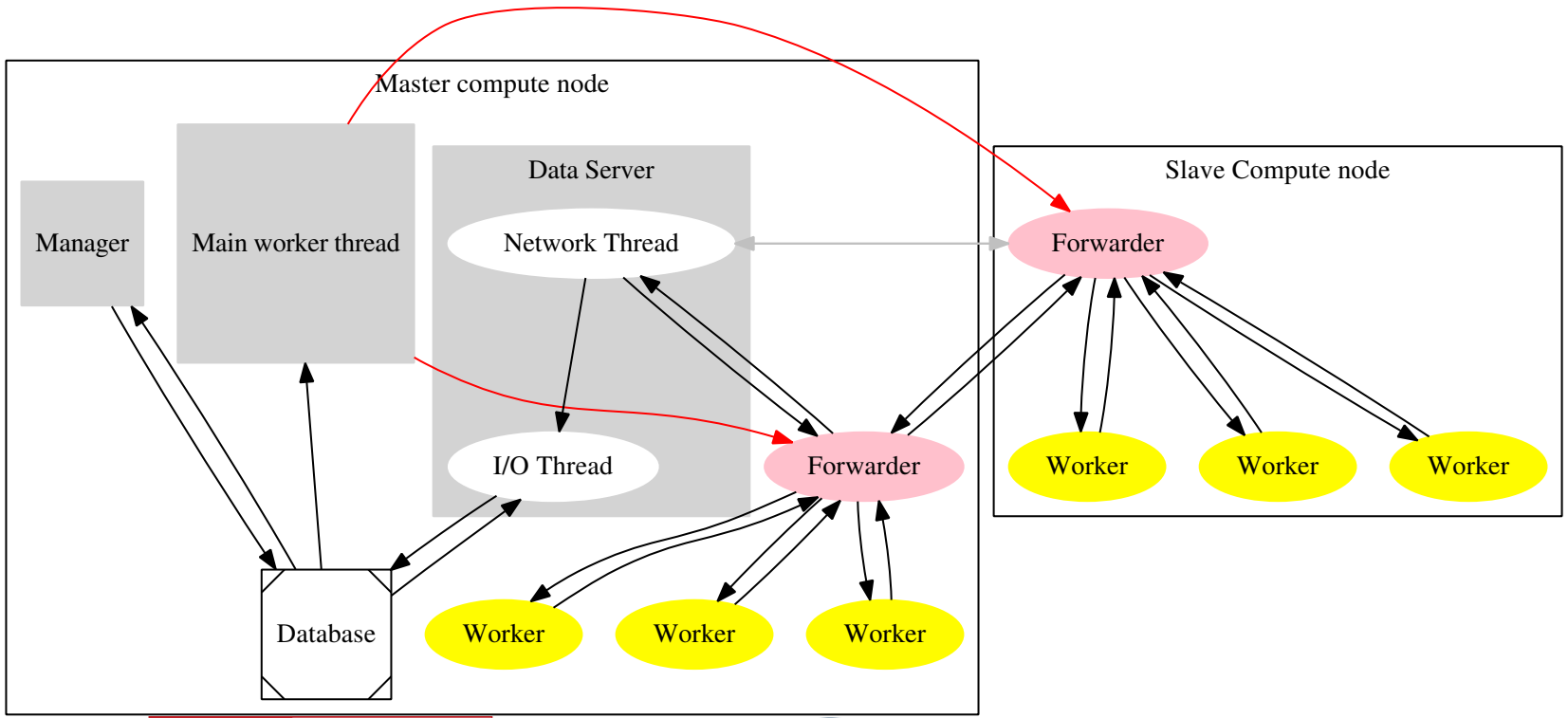
Implementation in QMC=Chem

- Block : N_{walk} walkers executing N_{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

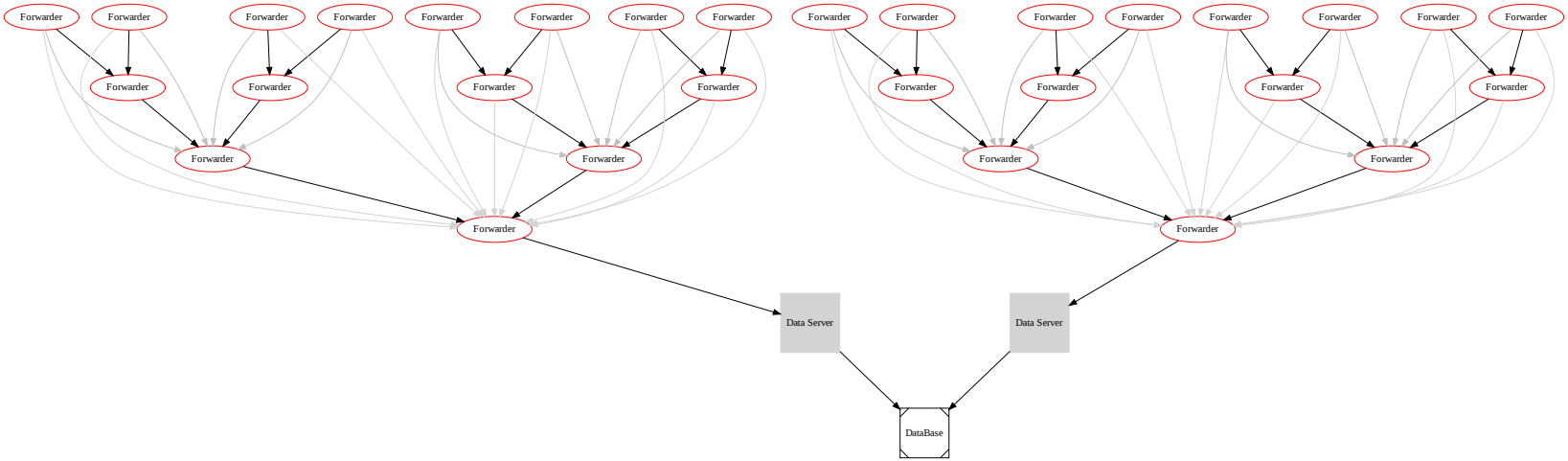
All I/O and network communications are asynchronous
Very small memory footprint (10—300 MiB/core)



Fault-tolerance

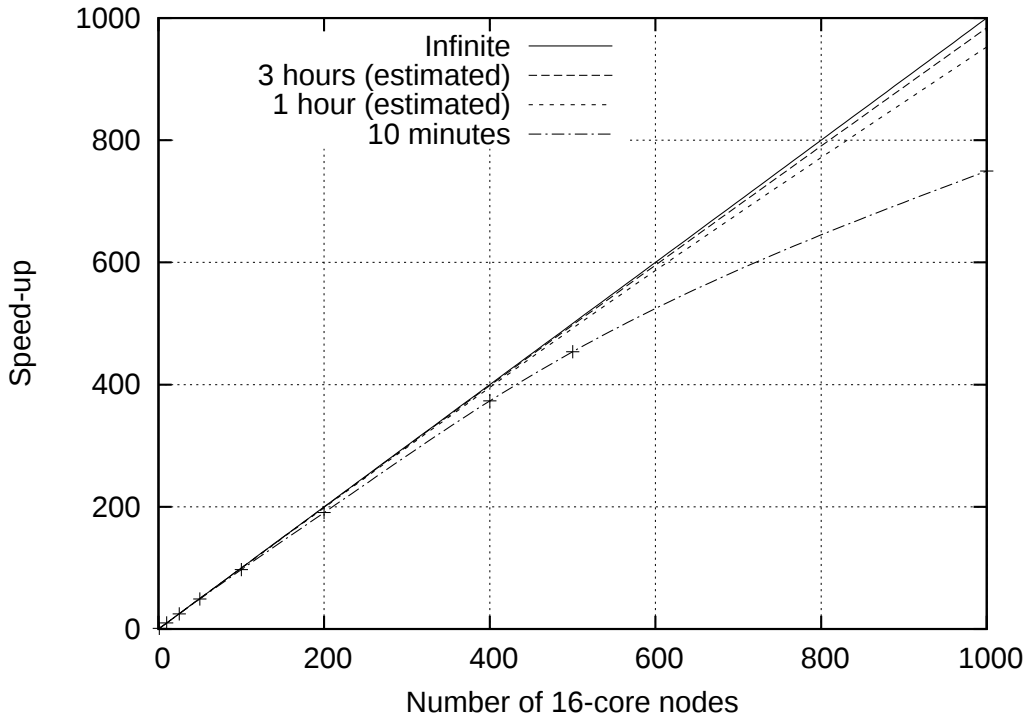
Extreme parallelism $\longrightarrow$ possible system failures

- 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_{basis} : number of basis functions, N_{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_{basis} : number of basis functions, N_{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_{basis} : number of basis functions, N_{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_{basis} : number of basis functions, N_{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_{basis} : number of basis functions, N_{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

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).

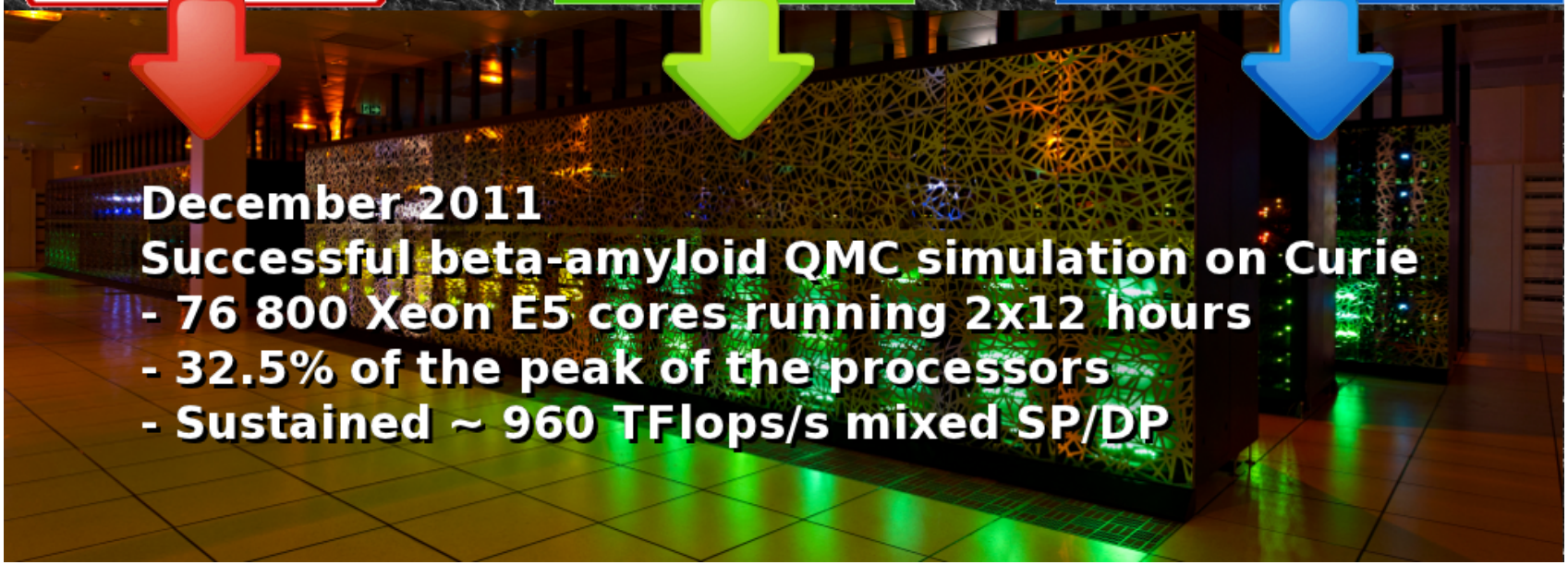
N_{elec}	N_{basis}	Matrix inversion	Matrix products	Overall (1 core)
158	404	6.3 (24%)	26.6 (50%)	8.8 (23%)
434	963	14.0 (53%)	33.1 (63%)	11.8 (33%)
434	2934	14.0 (53%)	33.6 (64%)	13.7 (38%)
1056	2370	17.9 (67%)	30.6 (58%)	15.2 (49%)
1731	3892	17.8 (67%)	28.2 (53%)	16.2 (55%)

Partners

M. Caffarel, A. Scemama
* QMC Methods
* QMC=Chem code

GENCI: Leader of national HPC resources
* Provides Curie

W. Jalby, E. Oseret
* Performance analysis tools
* Xeon E5 optimization



December 2011
Successful beta-amyloid QMC simulation on Curie
- 76 800 Xeon E5 cores running 2x12 hours
- 32.5% of the peak of the processors
- Sustained ~ 960 TFlops/s mixed SP/DP