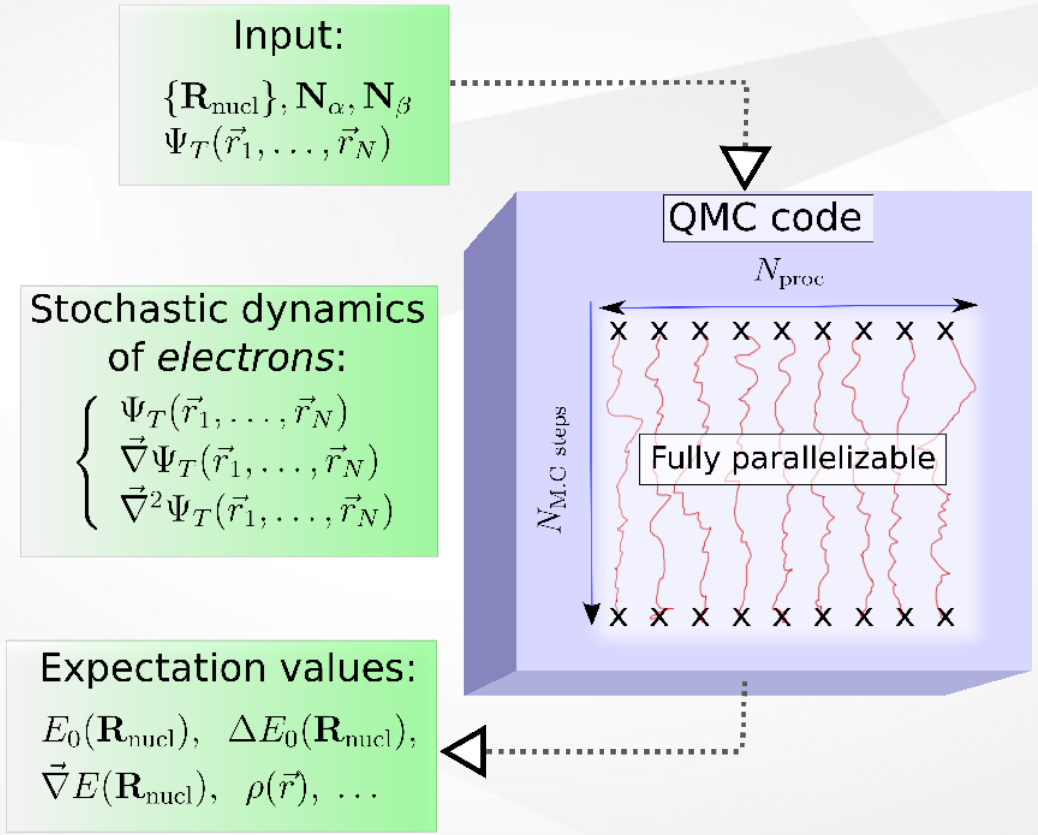
Implementation of parallelism in QMC=Chem

Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Michel Caffarel <michel.caffarel@irsamc.ups-tlse.fr>

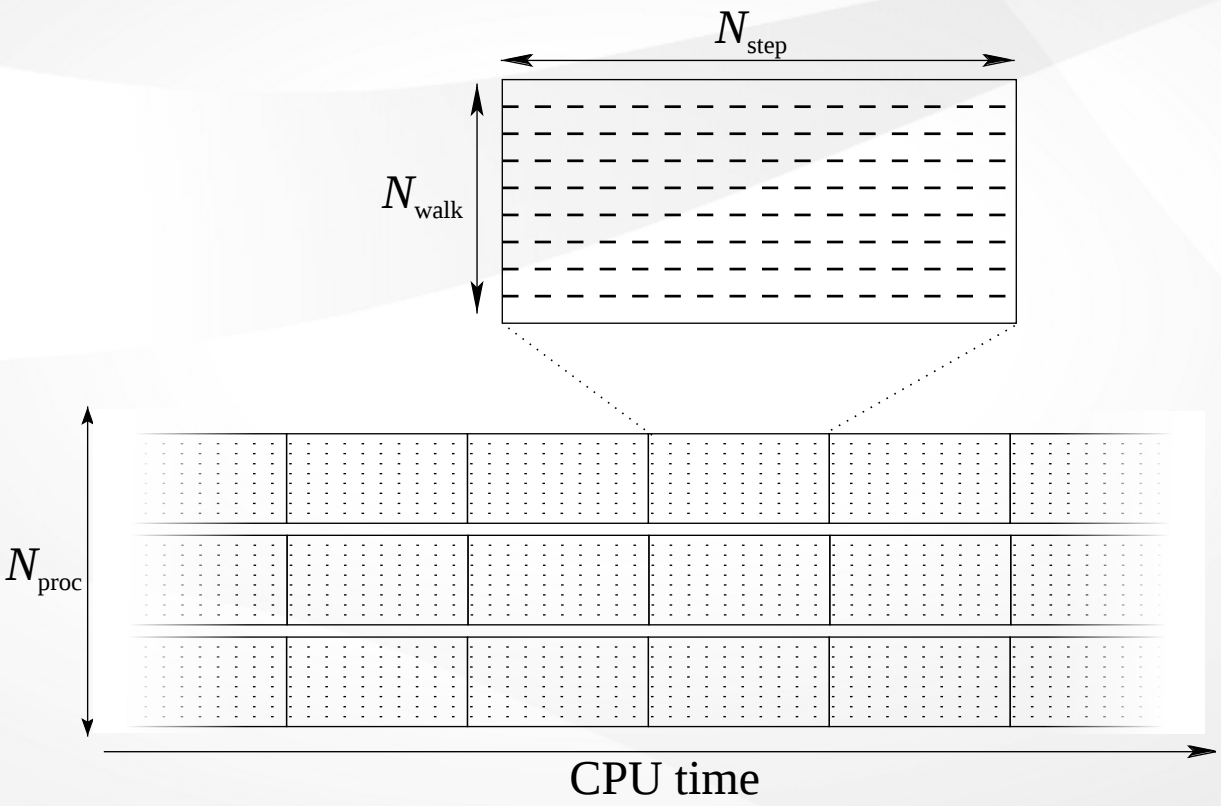
Labratoire de Chimie et Physique Quantiques
IRSAMC (Toulouse)

Parallelization of VMC

In VMC, all the trajectories are completely independent:



- Pack together a pool of N_{walk} walkers
- Cut the trajectories in smaller pieces of equal size (N_{step})
- Each CPU computes a block: N_{walk} executing N_{step}



Naive implementation:

- Parallelize with MPI
- At the end of each block, call *MPI_all_reduce* to update the running averages
- If too much memory is used, eventually add an OpenMP layer

This approach is not optimal [*]:

- At every synchronization, all processes will wait until the slowest has finished. Perfect parallel speed-up is impossible to obtain.
- The calculation can't start until all resources are available
- If one compute node crashes, all the simulation is crashed.
- If more resources become available, it is impossible to attach more CPUs to a running calculation

---

[*] "**Manager–worker-based model for the parallelization of quantum Monte Carlo on heterogeneous and homogeneous networks**", M. T. Feldmann, J. C. Cummings, D. R. Kent, R. P. Muller, W. A. Goddard III, J. *Comput. Chem.* 29, 8–16 (2008).

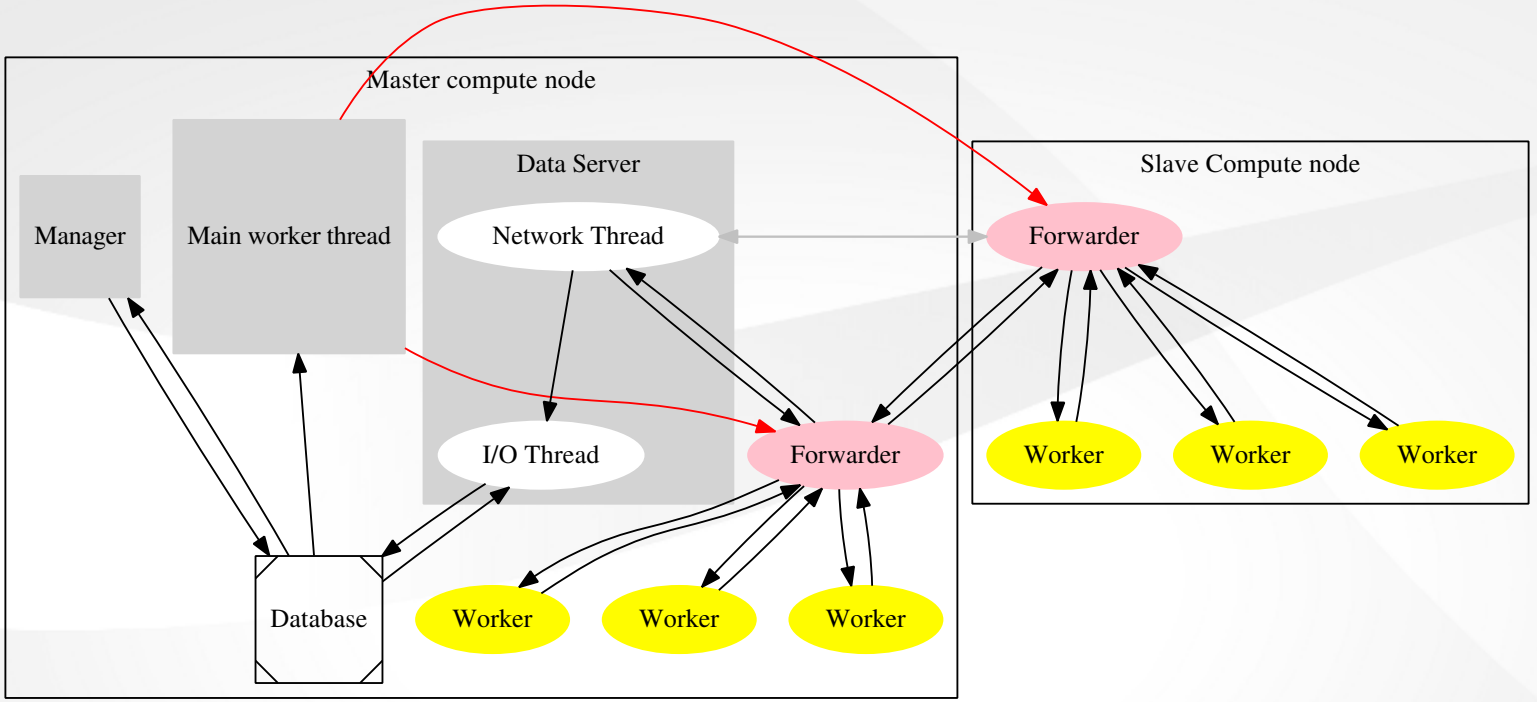
Our approach [†] :

- Manager/worker model: all CPUs are desynchronized, they start immediately
- The length of the block is not fixed: termination is immediate
- Use as less memory/core as possible (<100 MiB / core)
- Implement a client/server model (in Python):
  - allows client nodes to crash
  - allows to dynamically add/remove clients
- Avoid the traditional input/output file model. All data is stored in a database, and data is post-processed.
- Possibility to use computing grids (EGI [‡])

---

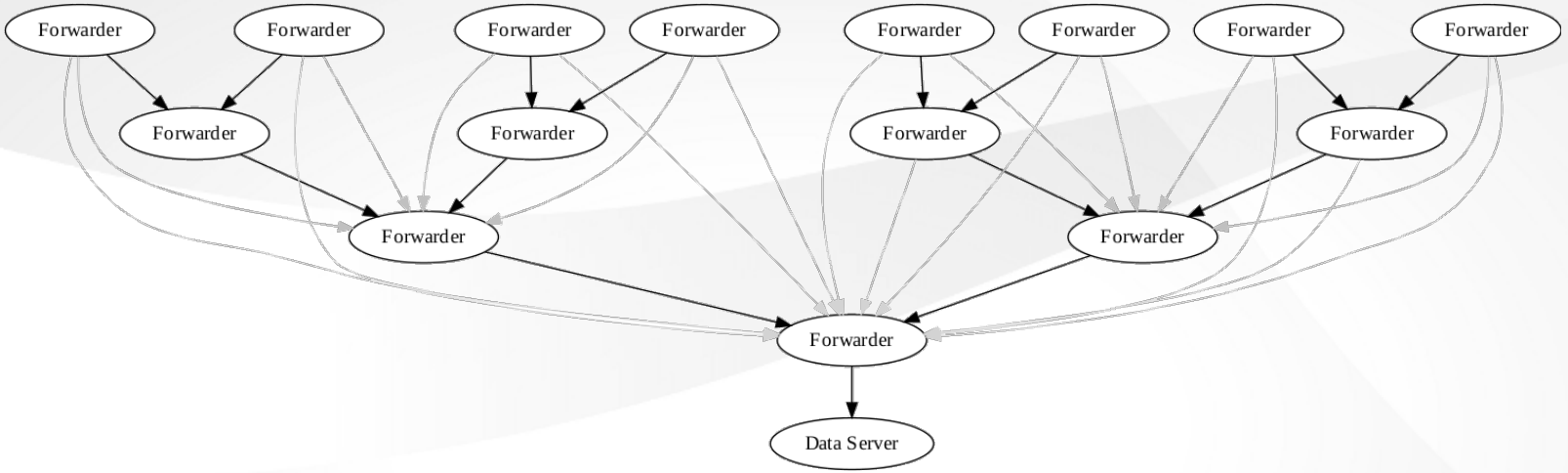
† "**Quantum monte carlo for large chemical systems: Implementing efficient strategies for petascale platforms and beyond**", A. Scemama, M. Caffarel, E. Oseret and W. Jalby, *J. Comput. Chem* 34, 938–951 (2013).

‡ "**Large-scale quantum Monte Carlo electronic structure calculations on the EGEE grid**", A. Monari, A. Scemama and M. Caffarel, *Remote Instrumentation for eScience and Related Aspects*, 195--207, Springer (2012).



- All I/O and network communications are non-blocking
- Worker: Single-core Fortran executable piped to a forwarder
- A Worker stops cleanly when its receives the *SIGTERM* signal

Fault-tolerance



- No access to the filesystem: scripts, binary and input data are broadcasted to the client nodes and stored in */dev/shm*. Local disks can crash.
- Blocks have a Gaussian distribution. Losing blocks doesn't change the average. Any worker can be removed.
- Every forwarder can always reach the data server. Any node can be removed.
- If the data server is lost, it is always possible to continue the calculation in another run.

Parallelization of DMC

- In the standard DMC algorithm, the walkers are no more independent.
- Communications are expected to kill the ideal speed-up.
- The Pure Diffusion MC algorithm [§] allows to obtain the DMC energy with re-weighting instead of branching: no more communication.
- PDMC introduces too much fluctuations in the total energies
- We use an algorithm that combines branching and re-weighting. [¶] Small populations can be used, and multiple independent runs can be done.

---

§ "**Development of a pure diffusion quantum Monte Carlo method using a full generalized Feynman–Kac formula.**", M. Caffarel, *J. Chem. Phys.* 88, 1088 (1988)

¶ "**Diffusion monte carlo methods with a fixed number of walkers**", R. Assaraf, M. Caffarel, A. Khelif, *Phys Rev E* 61(4 Pt B), 4566-75 (2000).

Why a database?

- Input and output data are tightly linked
- An output file can be generated on demand
- Easy connection to GUI
- An API simplifies scripting to manipulate results
- Checkpoint/restart is trivial
- Additional calculation can be done even if the calculation is finished. No need to re-run.
- Combining results obtained on different datacenters is trivial
- Multiple independent runs can write in the same database : dynamic number of CPUs.
- The name of the database is an MD5 key, corresponding to critical input data.

Initial conditions

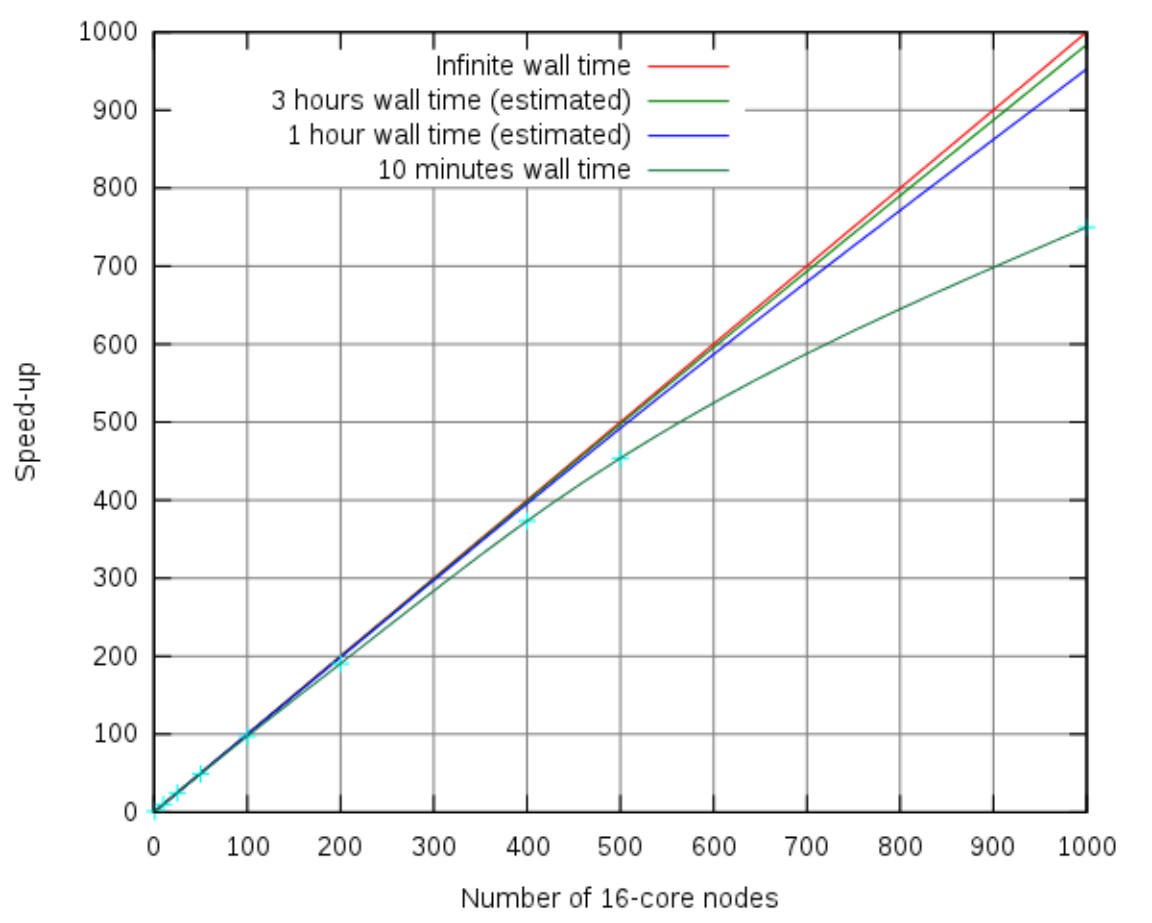
- Different initial walker positions are needed
- At the end of each block, the final positions are sent to the forwarder
- Each forwarder keeps a sample of the populations of its workers
- Sometimes, the forwarder sends its walkers to its parent in the tree
- The data server receives a sample of the population of all the walkers and merges it with its population
- Periodically, the population is saved to disk
- When a new run is started, each worker gets N_{walk} walkers drawn randomly from this population

Termination

When the manager wants to terminate the calculation (catches *SIGTERM*, or termination condition reached):

- It sends to the leaves of the tree a termination signal
- The leaves send a *SIGTERM* to the workers
- Each forwarder gets the data of the last blocks from the workers
- When the workers have terminated, the forwarder sends the data to its parent with a termination signal
- When the data server receives the termination signal, it the calculation is finished

Parallel speed-up



Estimation checked on 100 nodes/1 hour. Accuracy of 0.9992

The parallel speed-up is almost ideal.

Single-core optimization is crucial : Every percent gained on one core will be gained on the parallel simulation