Quantum chemistry in the Cloud

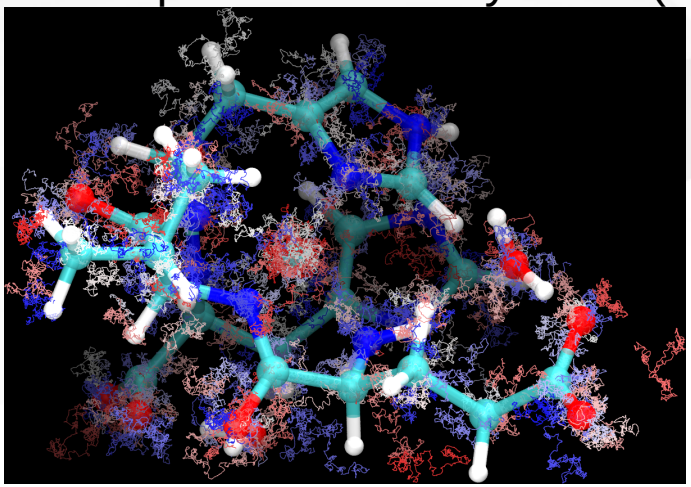
Anthony Scemama[1] <scemama@irsamc.ups-tlse.fr>
Thomas Applencourt[1], Michel Caffarel[1]
Georges da Costa[2]

[1] Labratoire de Chimie et Physique Quantiques
IRSAMC (Toulouse)
[2] IRIT (Toulouse)

LCPQ Laboratoire de Chimie et Physique Quantiques
IRIT CNRS - INPT - UPS - UT1 - UTM
UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse
cnrs

Quantum Chemistry

Quantitative description of complex chemical systems (energy differences).



Important scientific and technological applications :

- *Pharmaceutical industry* : drug design
- *Electronics industry* : Nano- and micro-electronic devices at the molecular level
- *Materials industry* : carbon nanotubes, graphene, etc.
- *Catalysis* : Enzymatic reactions in biochemistry, reactions in petroleum industry, ...

The scientific problem

Sloving the electronic **Schrödinger Equation**:

$$\mathcal{H}\Psi(\mathbf{R}) = -\frac{1}{2}\Delta\Psi(\mathbf{R}) + V(\mathbf{R}) = E_0\Psi(\mathbf{R})$$

- Ψ: electronic wave function of the ground state of the system
- $\mathbf{R}$: a 3N-dimensional vector of electron coordinates

Difficulty : It is a PDE in a 3N-dimensional space!

Standard approaches :

- Express $\Psi(\mathbf{R})$ on a finite 3N-dimensional basis set and solve the approximate problem using Krylov methods : linear algebra with very large matrices $(\sim 10^8 \times 10^8)$

Our approach :

1. Solve the problem in a perturbatively selected smaller subspace ($\sim 10^4 \times 10^4$) (CIPSI)
2. Use stochastic methods to solve the exact problem in the rest of the space (quantum Monte Carlo, QMC)

To show the accuracy of the method, we need to run atomization energy calculations on 55 molecules

Be	CH3Cl	F	HCO	Na	P	SiH2(3B1)
BeH	CH4	F2	HF	Na2	P2	SiH3
C	Cl	H2CO	HOCl	NaCl	PH2	SiH4
C2H2	Cl2	H2O	Li	NH	PH3	SiO
C2H4	ClF	H2O2	Li2	NH2	S	SO
C2H6	ClO	H2S	LiF	NH3	S2	SO2
CH	CN	H3COH	LiH	NO	Si	
CH2(1A1)	CO	H3CSH	N	O	Si2	
CH2(3B1)	CO2	HCl	N2	O2	Si2H6	
CH3	CS	HCN	N2H4	OH	SiH2(1A1)	

Atomization energy :

$$\mathrm{AE}(\mathrm{H_2O}) = E(\mathrm{H_2O}) - [2 \times E(\mathrm{H}) + E(\mathrm{O})]$$

One run :

1. Run a CIPSI Calculation
2. Run a QMC Calculation

Usually we run calculations on :

- Curie (TGCC/CEA/Genci) : ~5000 2x8-core nodes (Sandy bridge)
- Occigen (Cines) : ~2100 2x12-core nodes (Haswell)
- Eos (Calmip) : ~600 2x10-core nodes (Ivy-bridge)
- lpqsv26.ups-tlse.fr (LCPQ) : Our general purpose cluster ~30 2x16 core nodes (Sandy/Ivy/Haswell/Nehalem/AMD/etc)
- 130.120.xxx.45 (LCPQ) : HP moonshot cluster 68 1x8 core nodes (Atom)

Our motivations for the cloud

- In a near future, we will have supercomputers with millions (maybe billions) of cores
- Resilience *needs* to be considered
- Tightly coupled implementations (MPI) will have difficulties to scale
- Embarrassingly parallel algorithms are better candidates
- Production is not constant in a year : peaks of production

If our codes run in the Cloud :

1. They will run even better on supercomputers
2. They will provide more flexibility to the users
3. It may be energetically more efficient

Multiple workflows

1) Collaborative development

We develop most of our codes. We need to check if all the install process works fine on different Linux distributions before pushing to GitHub. (~30 min CPU)

2) Large number of independent runs

Typically the 55 molecules benchmark. 55 independent 16-cores shared-memory runs. (~500 - 5 000 CPU hours / job).

3) Large distributed jobs

Typically a QMC run. A single job is distributed on a very large number (>thousands?) of VMs on different clouds. (~5 000 - 200 000 CPU hours / job).

1) Collaborative development

- Need for a multi-cpu VM : Test the parallel build process
- Test on *.deb and *.rpm based Linux
- Install packages with *apt-get* or *yum* : *gcc gcc-c++ gcc-gfortran make lapack-devel graphviz zip m4*
- Download and compile : resultsFile Zlib EZFIO Ocaml IRPF90 EMSL Ninja docopt
- Run non-regression tests

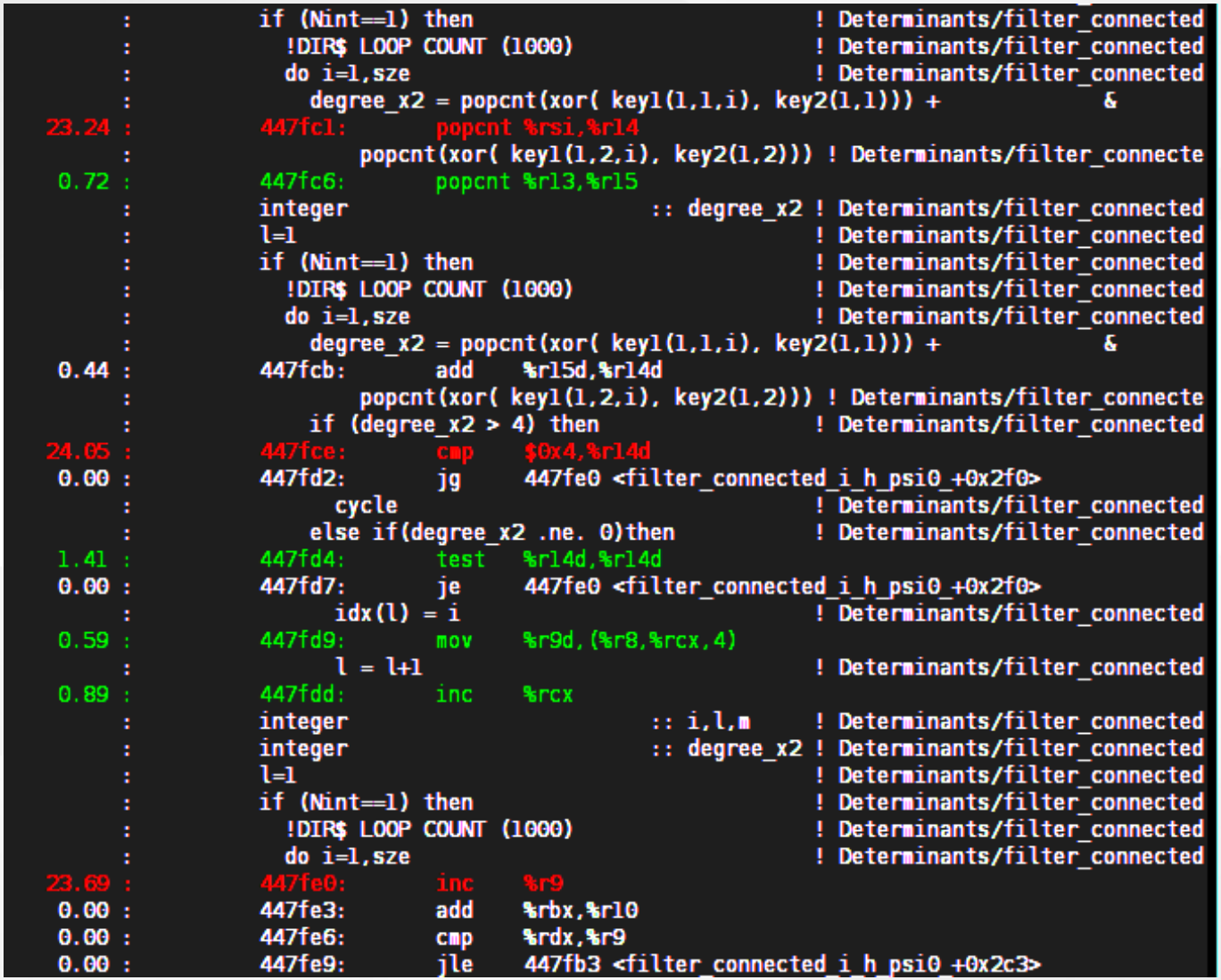
Extremely useful for everyday work.

2) CIPSI code : Quantum Package

- For the moment, OpenMP implementation
- Memory bound : Runs efficiently on low-power CPUs (Intel Atom in HP Moonshot chassis)
- Needs as much shared-memory CPU cores as possible : benchmarks with up to 384 cores on SGI Altix UV show a very good scaling
- Dominated by `popcnt` instruction, implemented in hardware on x86 CPUs since SSE4.2

```
top - 14:04:14 up 14 days, 22:43,  1 user,  load average: 329.54, 307.65, 325.17
Tasks: 2690 total,   3 running, 2686 sleeping,   0 stopped,   1 zombie
Cpu(s): 95.5%us,  3.1%sy,  0.0%ni,  1.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3101742M total,   540847M used,  2560895M free,      293M buffers
Swap:     8191M total,        0M used,     8191M free,   425631M cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  P COMMAND
234124 scemama   20   0 18.4g 9.8g 7196 R 18032  0.3 167678:08 194 full_ci
161321 scemama   20   0 13.1g 2.7g 6956 R 18004  0.1 280:36.59   2 full_ci
161311 scemama   20   0 11404 3724  932 R     9  0.0   0:13.95   1 top
  1286 root      20   0     0    0    0 S     1  0.0   1:36.72 319 kworker/319:1
     9 root      20   0     0    0    0 S     0  0.0   0:01.02   1 ksoftirqd/1
```


if (Nint==1) then ! Determinants/filter_connected
!DIR$ LOOP COUNT (1000) ! Determinants/filter_connected
do i=1,sze ! Determinants/filter_connected
degree_x2 = popcnt(xor(key1(1,1,i), key2(1,1))) + &
447fc1: popcnt %rsi,%r14
popcnt(xor(key1(1,2,i), key2(1,2))) ! Determinants/filter_connecte
447fc6: popcnt %r13,%r15
integer :: degree_x2 ! Determinants/filter_connected
l=1 ! Determinants/filter_connected
if (Nint==1) then ! Determinants/filter_connected
!DIR$ LOOP COUNT (1000) ! Determinants/filter_connected
do i=1,sze ! Determinants/filter_connected
degree_x2 = popcnt(xor(key1(1,1,i), key2(1,1))) + &
447fcb: add %r15d,%r14d
popcnt(xor(key1(1,2,i), key2(1,2))) ! Determinants/filter_connecte
if (degree_x2 > 4) then ! Determinants/filter_connected
447fce: cmp $0x4,%r14d
447fd2: jg 447fe0 <filter_connected_i_h_psi0_+0x2f0>
cycle ! Determinants/filter_connected
else if(degree_x2 .ne. 0)then ! Determinants/filter_connected
447fd4: test %r14d,%r14d
447fd7: je 447fe0 <filter_connected_i_h_psi0_+0x2f0>
idx(l) = i ! Determinants/filter_connected
447fd9: mov %r9d,(%r8,%rcx,4)
l = l+1 ! Determinants/filter_connected
447fdd: inc %rcx
integer :: i,l,m ! Determinants/filter_connected
integer :: degree_x2 ! Determinants/filter_connected
l=1 ! Determinants/filter_connected
if (Nint==1) then ! Determinants/filter_connected
!DIR$ LOOP COUNT (1000) ! Determinants/filter_connected
do i=1,sze ! Determinants/filter_connected
447fe0: inc %r9
447fe3: add %rbx,%r10
447fe6: cmp %rdx,%r9
447fe9: jle 447fb3 <filter_connected_i_h_psi0_+0x2c3>

Benchmarks

ID	CPU	GHz	Cores	Cache	Where
CALMIP	Intel Xeon E5-2680 v2	2.8	2x10	25 MiB	CALMIP
DESK	Intel Xeon E3-1271 v3	3.6	1x4	8 MiB	Desktop
MOON	Intel Atom C2730	2.4	1x8	4 MiB	Moonshot
IPHC	Westmere (E/L/X)56xx (Nehalem-C)	2.5	2x8	4 MiB	IPHC Cloud
LAL	QEMU Virtual CPU (cpu64-rhel6)	2.7	1x8	4 MiB	LAL Cloud

IPHC

Institut Pluridisciplinaire Hubert CURIEN, Strasbourg
Openstack

LAL

Laboratoire de l'Accélérateur Linéaire, Orsay
Stratuslab

Single thread execution (seconds)

Instructions	CALMIP	DESK	MOON	IPHC	LAL
SSE2 gfortran		1040.1			
SSE2 ifort		687.0			1193.0
>=SSE4.2 ifort	160.3	122.0	375.9	195.4	

Renormalized to 1GHz CPU (removing turbo):

Instructions	CALMIP	DESK	MOON	IPHC	LAL
SSE2 gfortran		3744.3			
SSE2 ifort		2473.2			3180.5
>=SSE4.2 ifort	448.84	439.2	902.16	488.5	

Multi-thread execution (seconds)

Instructions	CALMIP		DESK		MOON	IPHC	LAL
N(cores)	20	20(HT)	4	8(HT)	8	16	8
SSE2 ifort			267.0	487.0			153.7
>=SSE4.2 ifort	9.7	9.3	31.1	24.1	60.0	19.0	

Conclusions

- Single core code at IPHC is ~10% slower than on CALMIP. Probably due to memory frequency (1600 vs 1833 MHz ?)
- VMs at LAL don't provide SSE4.2 => huge performance loss 6.5x slower.
- Multi-threaded code performs well on VMs : 10.3x speedup on 16 cores
- Multi-threaded code performs better on physical machines : 17.2x speedup on 20 cores

Single node benchmark

Production runs

The workflow of independent jobs works :

- We wrote a simple XML-RPC queuing system with Python
- The server has a queue of CIPSI jobs to run
- The server runs on a desktop computer
- VM Contextualization : when the VM starts, it fetches the next input
- When the job is finished, the result is sent to the server

Main difficulty : Pass through all the security between the VMs and the server

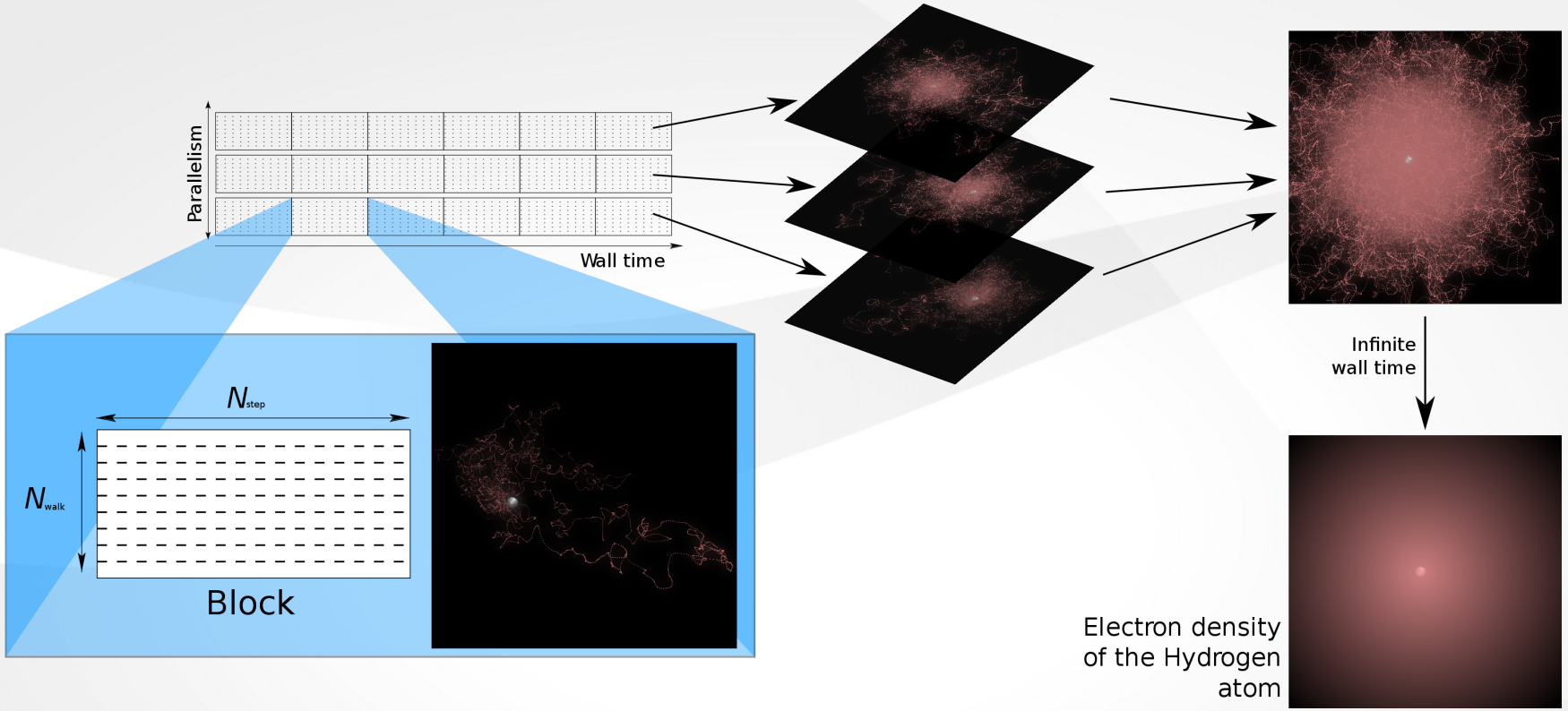
- Use SSH tunnels (`ssh -L` option) to tunnel the XML-RPC port
- Use ssh key authentication with no password for all jumps
- Use netcat in the ssh configuration to jump through the gateway :

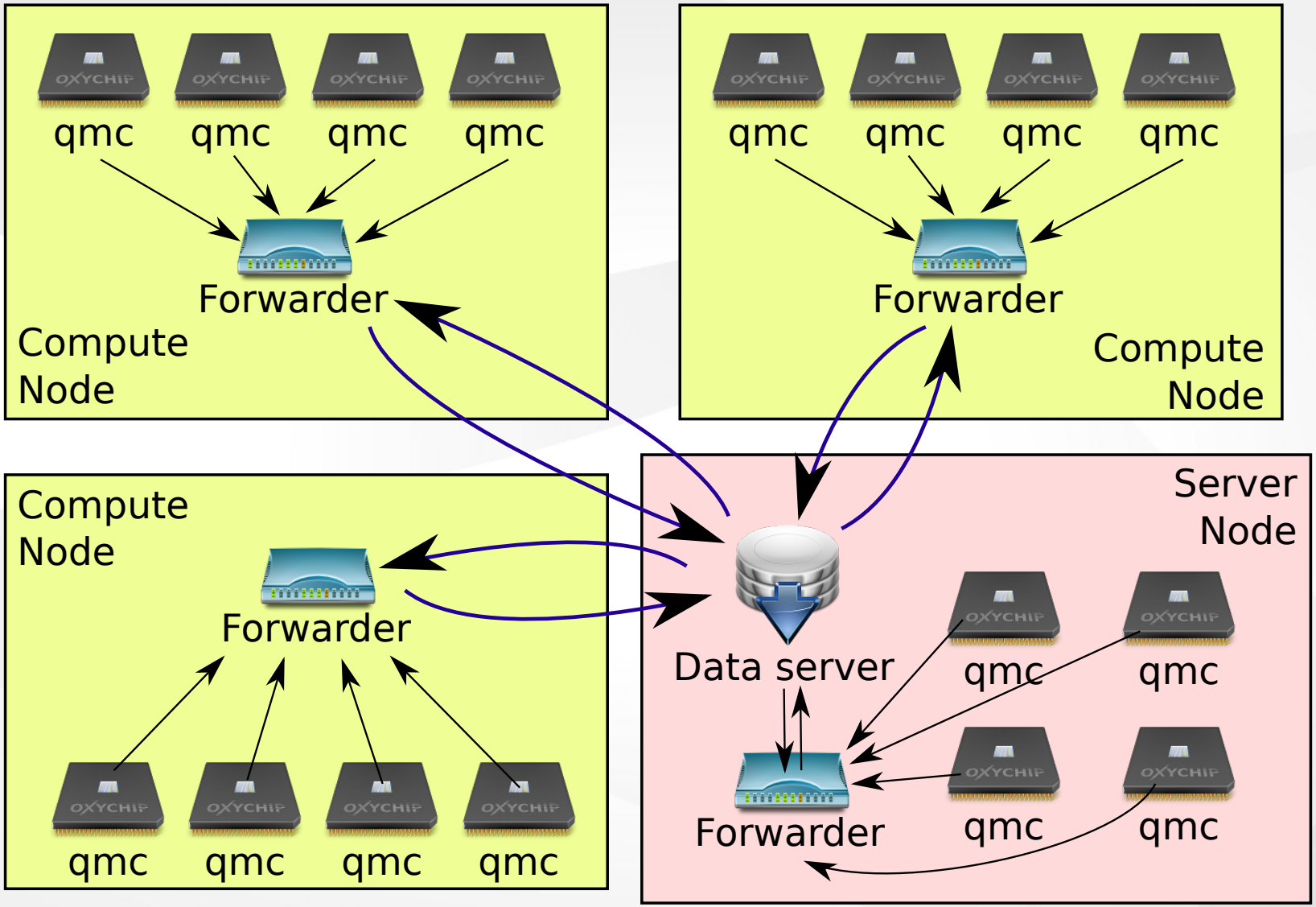
```
Host my_desktop
  User           scemama
  HostName       my_desktop
  ProxyCommand   ssh scemama@lcpq_gateway.ups-tlse.fr nc %h %p
```

3) QMC code : QMC=Chem

- Embarrassingly parallel implementation using a client/server model
- Very low memory footprint : < 100MiB per core, 2.1MiB in the benchmarks
- Optimized "by hand" : Array alignment, vectorization, padding etc
- CPU/cache bound : Intense use of vector FP instructions. Better performance with better CPUs, especially on large molecules.
- Needs as much CPU cores as possible : 76 800 cores on Curie in 2011 with 0.96 PFlops/s for 24h.

- Computing processes : Single-threaded Fortran
- Communications : ZeroMQ High performance networking library
- Forwarder/Data server : Ocaml
- Fault tolerant : any process can be killed without affecting the rest of the simulation


Parallelism
Wall time
N_{step}
N_{walk}
Block
Infinite
wall time
Electron density
of the Hydrogen
atom


qmc
qmc
qmc
qmc
Forwarder
Compute
Node
qmc
qmc
qmc
qmc
Forwarder
Compute
Node
Compute
Node
Forwarder
qmc
qmc
qmc
qmc
Server
Node
Data server
qmc
qmc
Forwarder
qmc
qmc

Benchmarks (seconds)

Inst	CALMIP	DESK	MOON	IPHC	LAL
SSE2	15.39	10.30	43.01	19.12	17.80
SSE4.2	14.51	9.52	42.01	18.87	
AVX	13.78	8.55			
AVX2		8.21			

Renormalized to 1GHz CPU (removing turbo): next slide

Inst	CALMIP	DESK	MOON	IPHC	LAL
SSE2	43.09	37.15	103.22	47.80	47.45
SSE4.2	40.63	34.27	100.82	47.16	
AVX	40.38	30.78			
AVX2		29.56			

LAL faster than IPHC

SSE4.2 gain not competitve wrt frequency difference

SSE4.2 not much better than SSE2

low-level optimzation makes an efficient SSE2 code

SSE2 faster than LAL/IPHC on CALMIP (and Desktop)

MKL autodetects the CPU : Not a pure SSE2 binary

SSE2 faster on Desktop than CALMIP

Haswell CPU has Increased cache bandwidth wrt Ivy Bridge

Poor performance on Atom, but...

3x slower than Desktop, but 2x more cores and 6.7x less power (12W / 80W)

Distributed QMC run

Usual run

```
$ qmcchem run zn.ezfio
MD5 : 072e909b86094cc68c06b0dd3b88b32a
Scheduler : SLURM
Launcher  : srun
  25278 : /home/scemama/QmcChem/bin/qmcchem run -d zn.ezfio
MD5 : 072e909b86094cc68c06b0dd3b88b32a
Server address: tcp://130.120.229.139:41578
  25314 : srun /home/scemama/QmcChem/bin/qmcchem run -q tcp://130.120.229.139:41578 zn.ezfio
```

- `qmcchem run -d` : Run a data server only
- `qmcchem run -q` : Run one forwarder per node. Each forwarder will spawn multiple *qmc* processes
- Asynchronous connexions between *qmc* and *forwarder* : tcp://localhost:<port>
- Asynchronous connexions between *forwarder* and *data server* : tcp://<ip>:<port>

Distributed Run in the cloud (Proof of concept)

1. Start one VM at LAL (m1.xlarge) : 8 cores
   - Fetch the QMC=Chem code, and the input
   - Start a 8-core run : dataserver + forwarder + 8x qmc
   - Get the IP address and port number of the data server

```
$ qmcchem run zn.ezfio &
MD5 : 645285a41b990efb80f98706b711d159
Scheduler : Batch
Launcher  : env
  4172 : /root/QmcChem/bin/qmcchem run -d zn.ezfio
MD5 : 645285a41b990efb80f98706b711d159
Server address: tcp://134.158.75.78:34298
  4193 : env /root/QmcChem/bin/qmcchem run -q tcp://134.158.75.78:34298 zn.ezfio

root@onevm-78:~# for i in {1..7}
> do taskset -c $i qmcchem run -q tcp://134.158.75.78:34298 zn.ezfio &
> done
```

2. Start one VM at IPHC (m1.2xlarge)

- Copy the SSH private key to allow this VM to connect to the LAL VM via SSH
- Fetch the QMC=Chem code, and the input
- Install sshuttle : "Poor man's VPN"
- Create SSH tunnel to dataserver
- Run a forwarder with multiple *qmc*'s

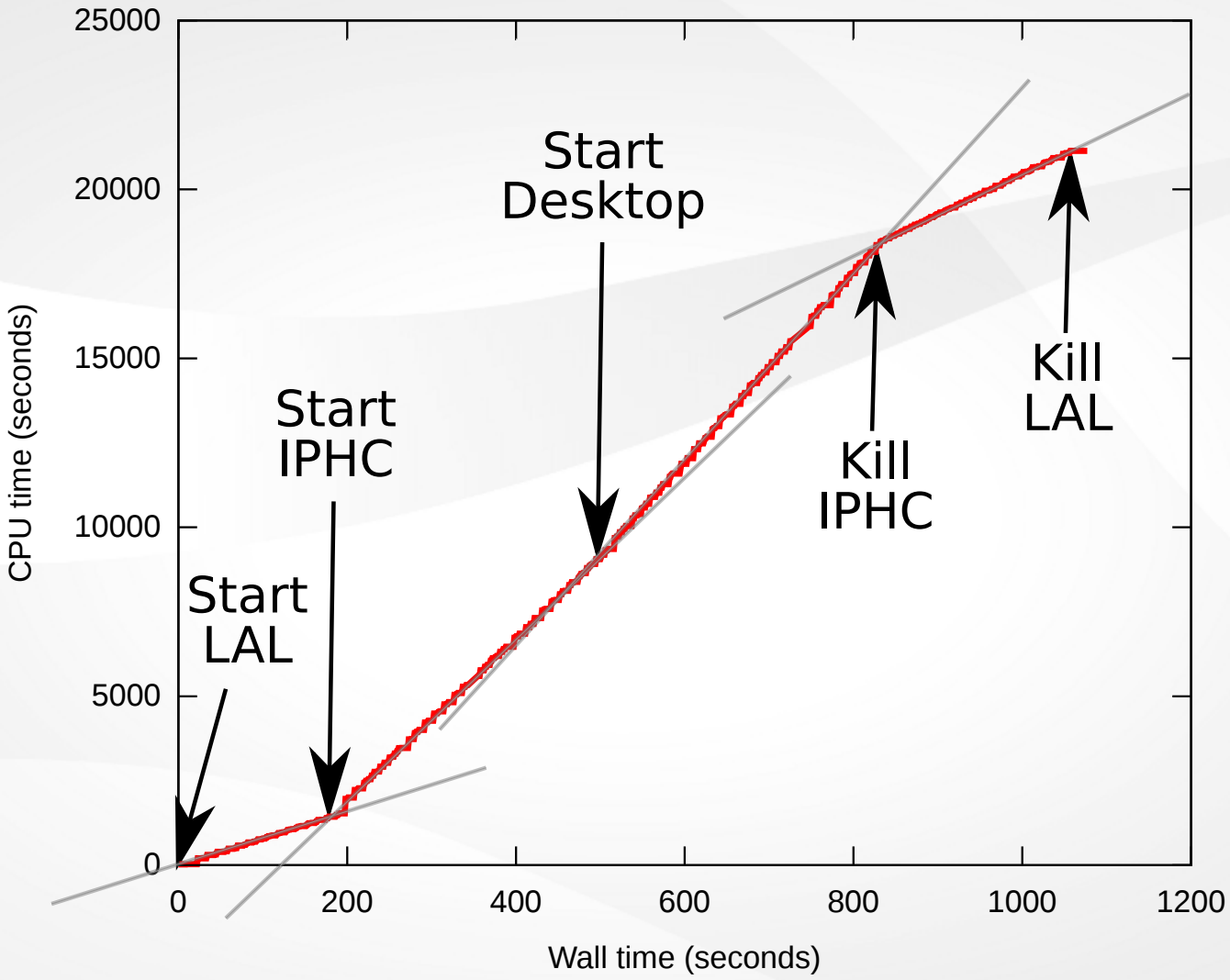
```
root@qmc:~# for i in {0..15}
> do taskset -c $i qmcchem run -q tcp://134.158.75.78:34298 zn.ezfio &
> done
```

3. Start a client on my Desktop

- Create SSH tunnel to dataserver
- Run a forwarder

```
root@qmc:~# for i in {0..3}
> do taskset -c $i qmcchem run -q tcp://134.158.75.78:34298 zn.ezfio &
> done
```

4. Kill the VM at IPHC
    - The simulation continues...
5. Kill the VM at LAL
    - After a timeout, my Desktop kills its process


25000
20000
15000
10000
5000
0
CPU time (seconds)
0
200
400
600
800
1000
1200
Wall time (seconds)
Start
Desktop
Start
IPHC
Start
LAL
Kill
IPHC
Kill
LAL

Efficiency of CPUs is not uniform :

Host	CPU (s)	# MC steps	# MC steps/s/core	# MC steps/s
Desktop	1807.3	17346179	9597.7	38390.8
IPHC	10455.3	29776927	2848.0	45568.0
LAL	7556.6	33417897	4422.3	35378.4

As for CIPSI, when a multi-core VM runs multiple processes, it becomes less efficient.

A better choice would be to run 16x m1.small instead of 1x m1.2xlarge.

Conclusion

Schrödinger is in the cloud!

Summary

- We can use the cloud to check the parallel build of our codes on multiple Linux distributions
- Main Difficulty we felt : security.
- The CIPSI code requires to have access to advanced instructions : works much better on IPHC cloud
- The QMC code works fine with SSE2. The frequency is the most important : works better on LAL cloud
- We are able to run many independent CIPSI OpenMP calculations
- We are able to run a QMC simulation simultaneously on different clouds

Perspectives

- Measure the energy efficiency of our calculations on the Cloud
- Configure a VPN to overcome our difficulties with security
- Automate the spawning of QMC runs
- Add web interfaces to our codes that can be hosted on VMs
- Use the Cloud for demos / diffusion of our codes
- Our QMC code is driven by an Ocaml program : it should not be too difficult to build unikernels with MirageOS

Acknowledgments

- Cécile Cavet
- Jérôme Pansanel
- François Thiebolt
- France Grilles (LAL, IPHC, IRIT, LUPM, CC-IN2P3)
- CALMIP