HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Electronic Structure Calculation Methods on Accelerators

ORNL/NICS – OAK RIDGE, TENNESSEE

Wavelet-Based DFT calculations on Massively Parallel Hybrid Architectures

Luigi Genovese

L_Sim – CEA Grenoble

February 7, 2012

A basis for nanosciences: the BigDFT project

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

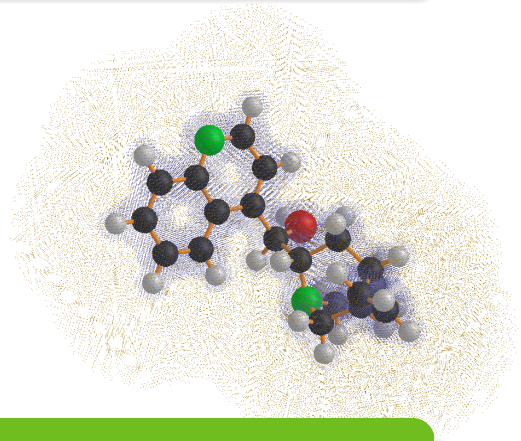
Conclusion

STREP European project: BigDFT(2005-2008)

Four partners, 15 contributors:
CEA-INAC Grenoble (T.Deutsch), U. Basel (S.Goedecker), U. Louvain-la-Neuve (X.Gonze), U. Kiel (R.Schneider)

Aim: To develop an ab-initio DFT code based on Daubechies Wavelets, to be *integrated in ABINIT*.

BigDFT 1.0 $\longrightarrow$ January 2008



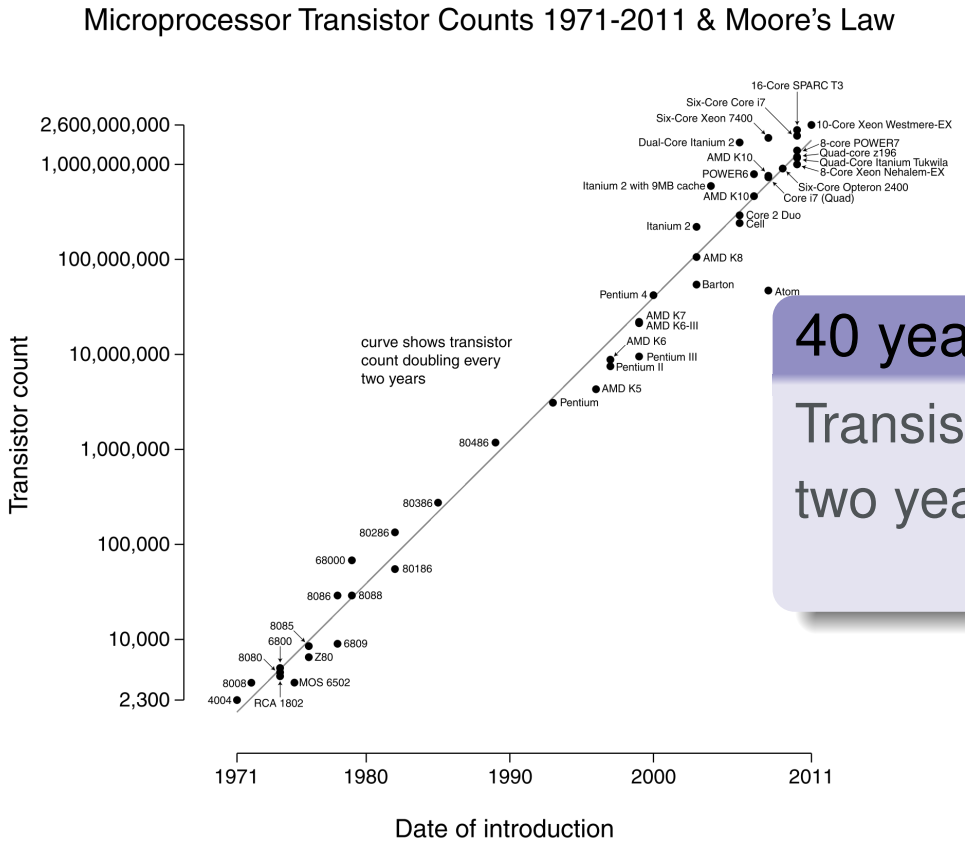
In this presentation

- Present HPC scenario
- GPU exploitation in BigDFT runs
- Consideration of interest for Electronic Structrure Calculations

Outline

1. Parallel computing and architectures
   - From past to present: software
   - HPC nowadays
   - Memory bottleneck
2. (DFT) Developer point of view
   - Present Situation
   - Optimization
3. User viewpoint
   - Frequent mistakes
   - Performance evaluation
   - Material accelerators: Evaluating GPU gain
   - Practical cases
4. Conclusion and Messages

Moore's law



40 years of improvements

Transistor counts double every two years...

...but how?

Power is the limiting factor (around 100 W nowadays)

Power $\propto$ Frequency^3 ☛ Clock rate is limited

Multiple slower devices *preferable* than one superfast device

☛ More performance with less power $\rightarrow$ software problem?

Why software problem?

The power cost of frequency

	Cores	Hz	(Flop/s)	W	Flop/s/W
Superscalar	1	1.5 ×	1.5 ×	3.3 ×	0.45
Multicore	2	0.75 ×	1.5 ×	0.8 ×	1.88

Exercise:

- Take a given computational problem
- Write a code at a time t_0.
  Solve the problem on a computer.
- Freeze your code and wait some time $t_1 - t_0$
- Take a new computer at time t_1.
  Solve again the same problem.
- What happens to your performances?

HPC thumb-rules have changed

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Frequency-dominated era

- ✔ Parallelism is not improved by the architecture
- ✔ Frequency increases → No. Flop/s increases
- ☛ Code runs faster

Manycore era

- ✔ Parallelism is *dramatically changed* in the architecture
- ✔ Frequency *decreases*
- ☛ Code runs slower
- ✖ The code should be changed

The parallel behaviour of a code (oversimplification)

- Capacity computing: many independent jobs
- Capability computing: single job, parallel intensive

The cost of the memory transfer

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

1W $\times$ 1 Year = 1$ (neglecting cooling and storage)

Some facts about memory transfer: Memory bandwidth

40 GB/s (CPU); 20 GB/s (RAM); 3.5 GB/s (interconnect)

Bandwidth evolves less faster than computational power:

✔ ∼90's (Math co-processor): 1 Flop/s each 4 Bytes transferred

✖ Nowadays: 62 Flop/s per Bytes transferred

The cost in energy of data movement

Computation: a FMA costs now 100 pJ (10 pJ in the future)

- Move data in RAM costs 4.8 nJ (1.92 nJ)
- Communicating data (MPI) costs 7.5 nJ (2.5 nJ)

☛ Moore's law revisited:

Thread number executions will double each year

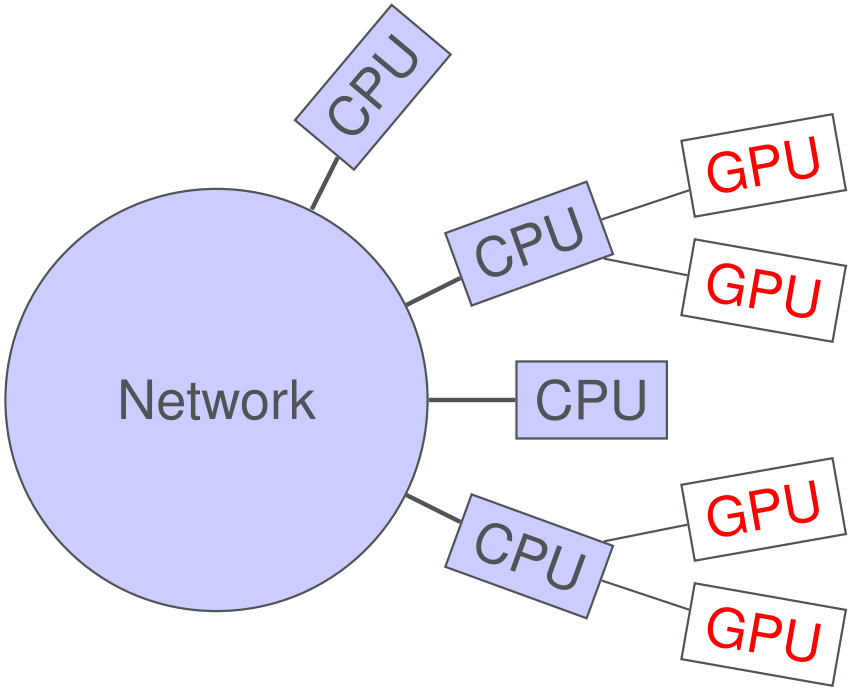
A complicated scenario for HPC with ab initio codes

Distribute the data on hybrid supercomputer

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck
Developer approach
Present Situation
Optimization
User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases
Conclusion

How a code can be executed on hybrid CPU-GPU architectures?



Data transfer is still MPI-based

Only on-board communication between GPU and CPU

Data distribution should depend on the presence of GPUs on the nodes → Multilevel parallelization required

(Hybrid) Supercomputing: developer's view

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

GPGPU on Supercomputers

- Traditional architectures are somehow saturating
  More cores/node, memories (slightly) larger but not faster
- Architectures of Supercomputers are becoming hybrid
  3 out to 5 Top Supercomputers are hybrid machines
- Extrapolation: In 2015, No. 500 will become petafloppic
  Likely it will be a hybrid machine

Codes should be conceived differently

- # MPI processes is limited for a fixed problem size
- Performances increase only by enhancing parallelism
- Further parallelisation levels should be added (OpenMP, GPU)

Does electronic structure calculations codes are suitable?

How *far* is petaflop (for DFT)?

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

At present, with traditional architectures

Routinely used DFT calculations are:

- Few dozens (hundreds) of processors
- Parallel intensive operations (blocking communications, 60-70 percent efficiency)
- Not *freshly* optimised (legacy codes, monster codes)

☛ Optimistic estimation: 5 GFlop/s per core $\times$ 2000 cores $\times$ 0.9 = 9 TFlop/s = 200 times less than Top 500's #3!

It is such as

Distance Earth-Moon = 384 Mm

Distance Earth-Mars = 78.4 Gm = 200 times more

Moon is reached. . . can we go to Mars? (. . . in 2015?)

BigDFT as a case study: operations performed

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

The SCF cycle

Orbital scheme:

- Hamiltonian
- Preconditioner

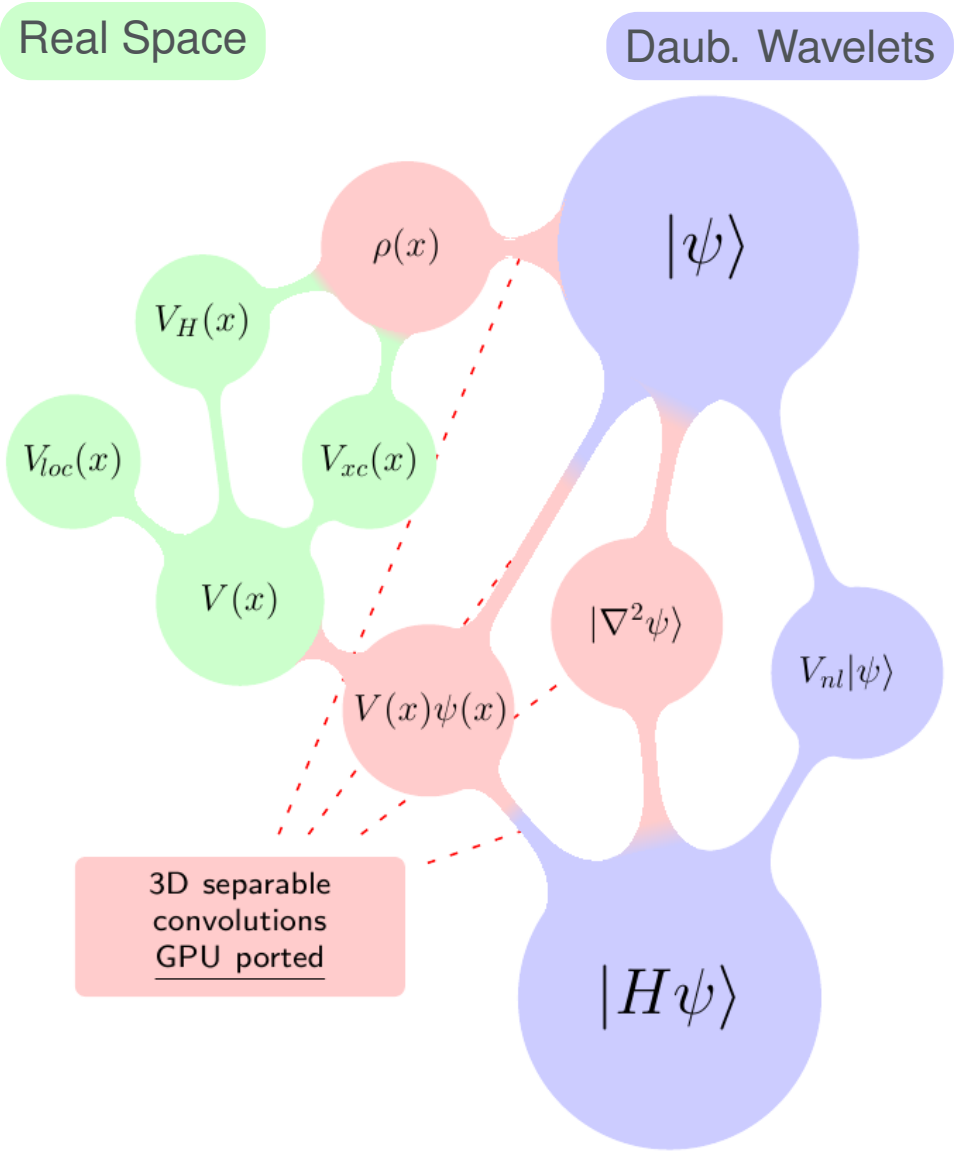
Coefficient Scheme:

- Overlap matrices
- Orthogonalisation

Comput. operations

- Convolutions
- BLAS routines
- FFT (Poisson Solver)

Why not GPUs?



How to optimize *particular* operations?

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

A trade-off between benefit and effort

FORTRAN based

- ✔ Relatively accessible (loop unrolling)
- ✔ Moderate optimisation can be achieved relatively fast
- ✖ Compilers fail to use vector engine efficiently

Push optimisation at the best

- About 20 different patterns have been studied for one 1D convolution
- Tedious work, huge code $\longrightarrow$ Maintainability?

☛ Automatic code generation?

Consider new programming paradigms

New coding approaches are *most* welcome

$\rightarrow$ Kronos' OpenCL standard

MPI parallelization I: Orbital distribution scheme

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

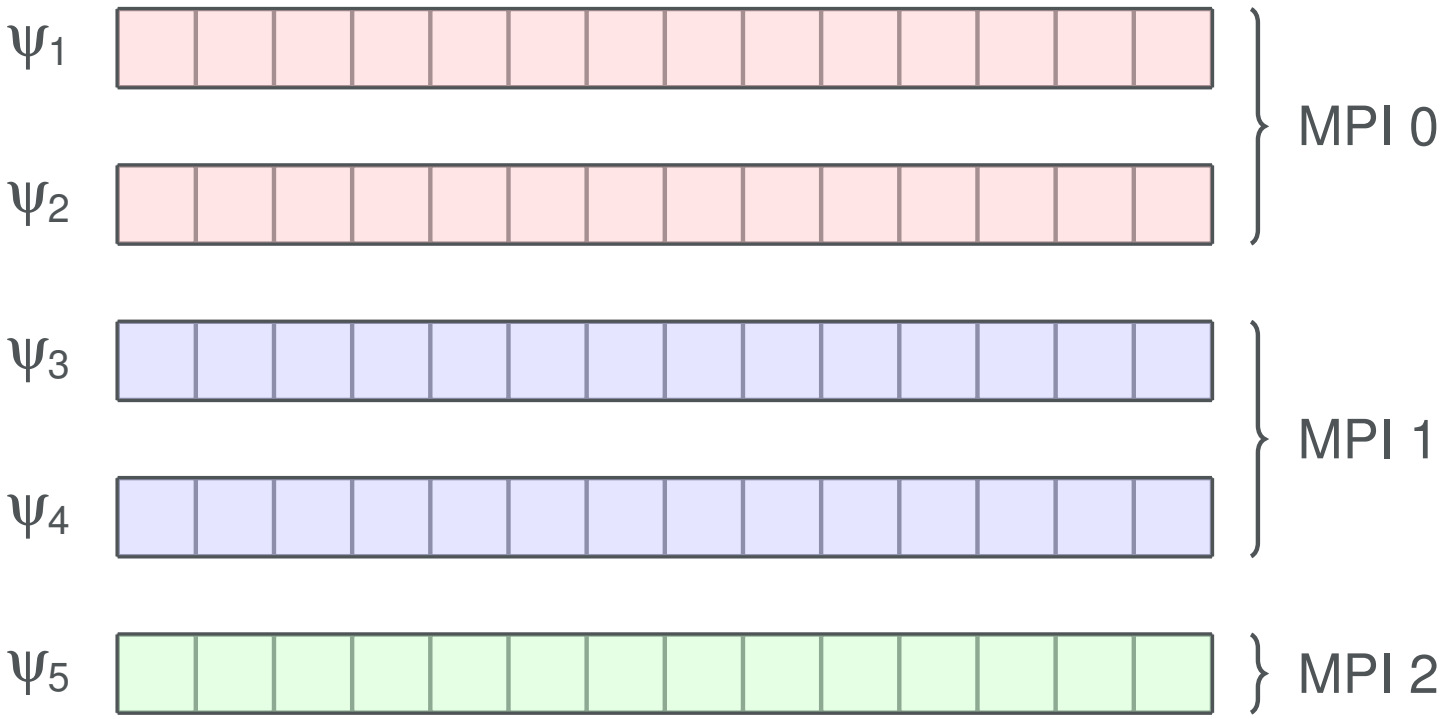
Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Used for the application of the hamiltonian

Operator approach: The hamiltonian (convolutions) is applied separately onto each wavefunction



MPI parallelization II: Coefficient distribution scheme

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

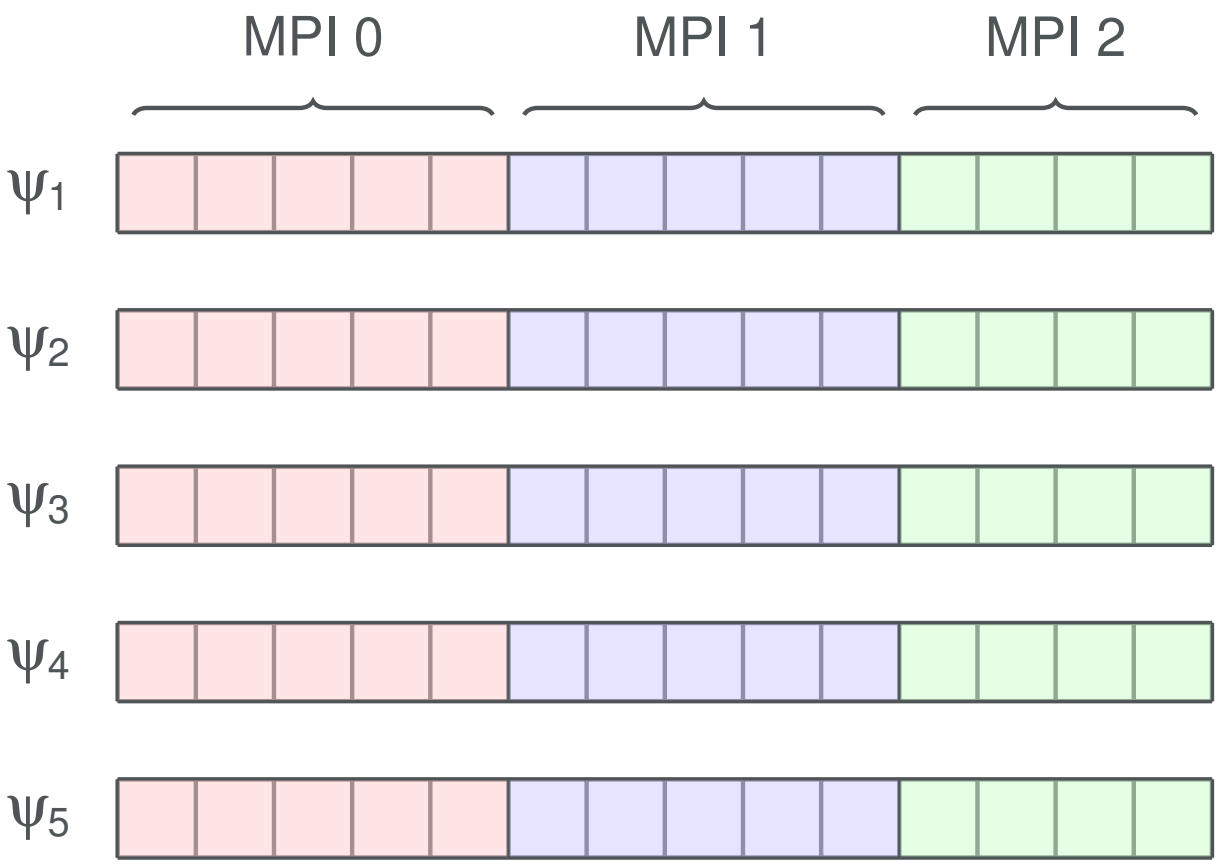
Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Used for scalar products & orthonormalisation

BLAS routines (level 3) are called, then result is reduced



At present, MPI_ALLTOALL(V) is used to switch

OpenMP parallelisation

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

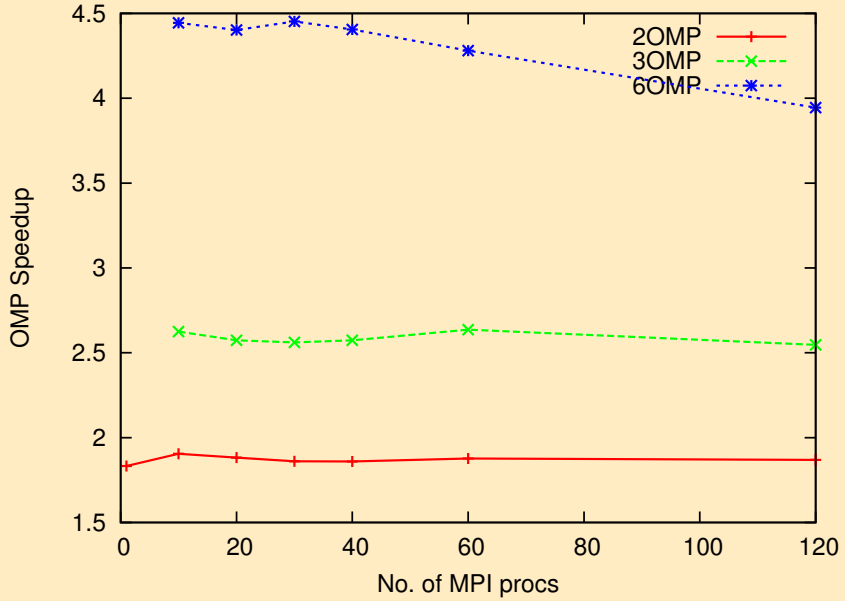
Conclusion

Innermost parallelisation level

(Almost) Any BigDFT operation is parallelised via OpenMP

- ✔ Useful for memory demanding calculations
- ✔ Allows further increase of speedups
- ✔ Saves MPI processes (less intra-node Message Passing)

- ✖ Less efficient than MPI (intranode)
- ✖ Compiler and system dependent
- ✖ OMP sections should be regularly maintained



Interpretation of HPC behaviour:

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Evaluate the code behaviour:

A not so easy task (especially nowadays)

Frequent mistakes:

- Parallel efficiency is not walltime
- Scalability is not only communication-driven
- Performance evaluation is a multicritierion evaluation process
  - Best scalability (Machine point of view)
  - Best acceleration efficiency (Vendor point of view)
  - Best walltime (User point of view)

But also robustness, fault tolerance

Anticipated messages

Far from trivial situation:

- No golden rule
- HPC Optimal Stategies should be *interpreted*

Amdahl's law

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

A basic concept

The speedup with N cores depends of the parallel fraction (P) of the code:

$$\text{speedup} = \frac{1}{\frac{P}{N} + (1 - P)}$$

It represents the limits to the scalability of a given code

An important definition

Parallel Efficiency $= \frac{Time(N_{\text{ref}})}{Time(N)} \frac{N}{N_{\text{ref}}}$ Often used as a benchmark of a code in a parallel environment

Lots of factors involved

- Scalability of the problem
- Communication performances
- Computational cost of operations

Intranode bandwidth problem

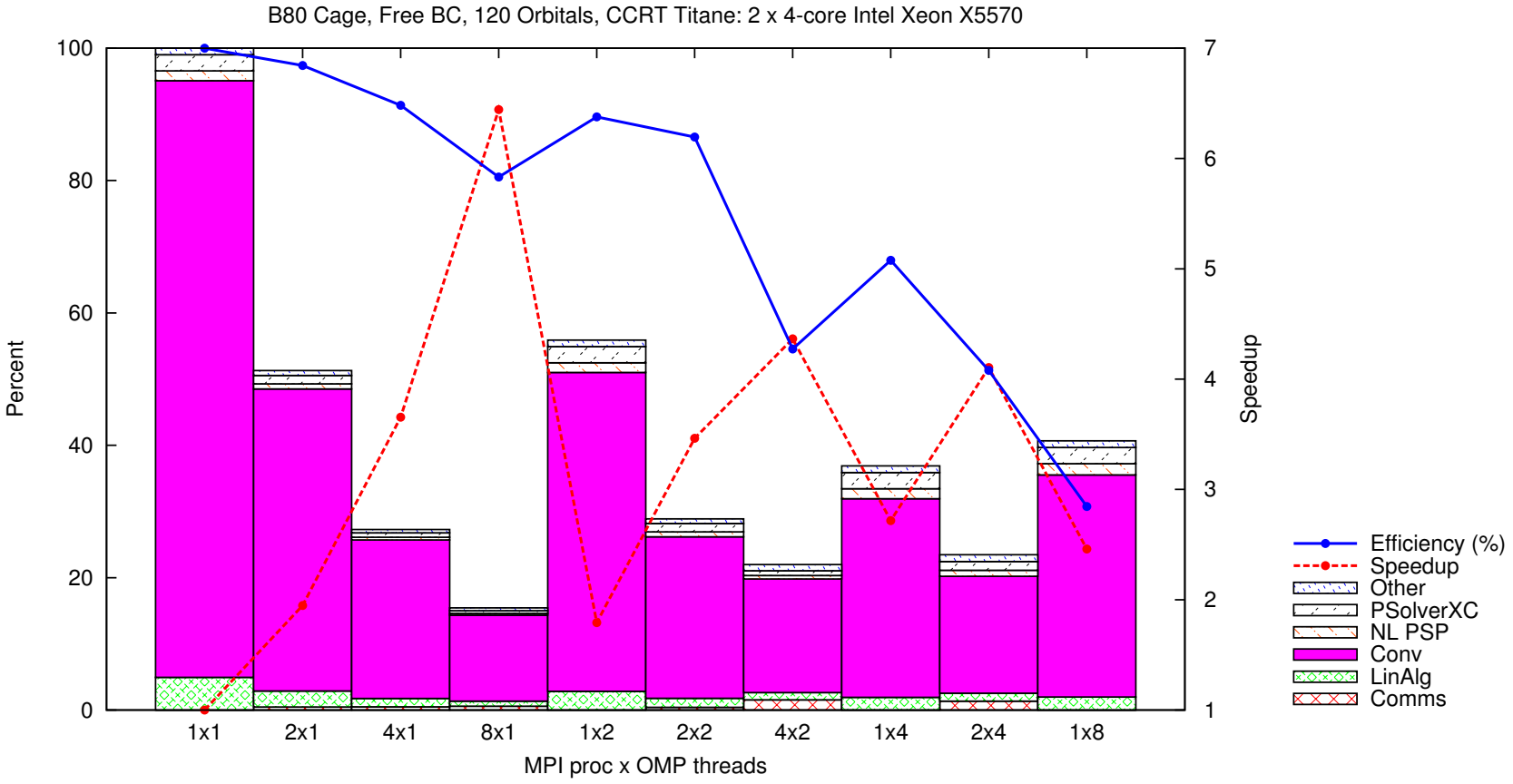
HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion



Scalability does not depend only on communication

Amdahl's law is a upper limit!

Parallelisation and architectures

HPC and BigDFT

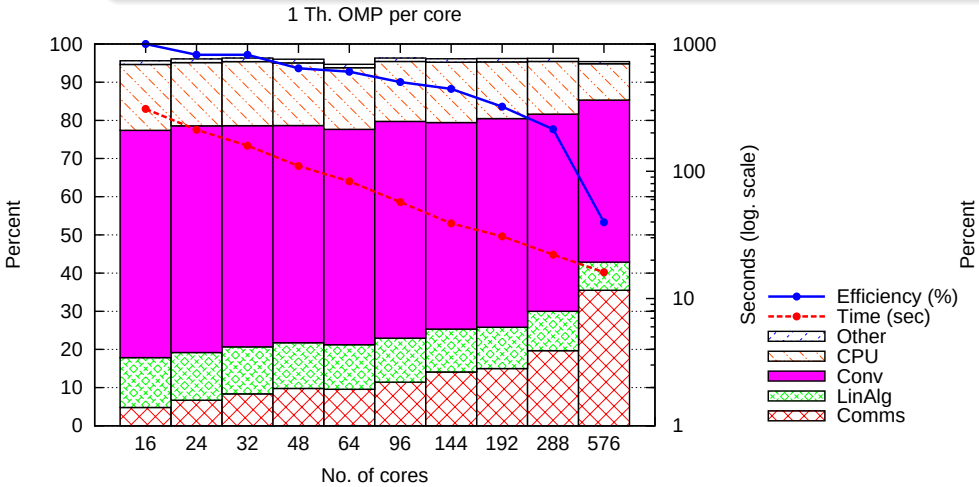
Architectures
- Software problem
- HPC nowadays
- Memory bottleneck

Developer approach
- Present Situation
- Optimization

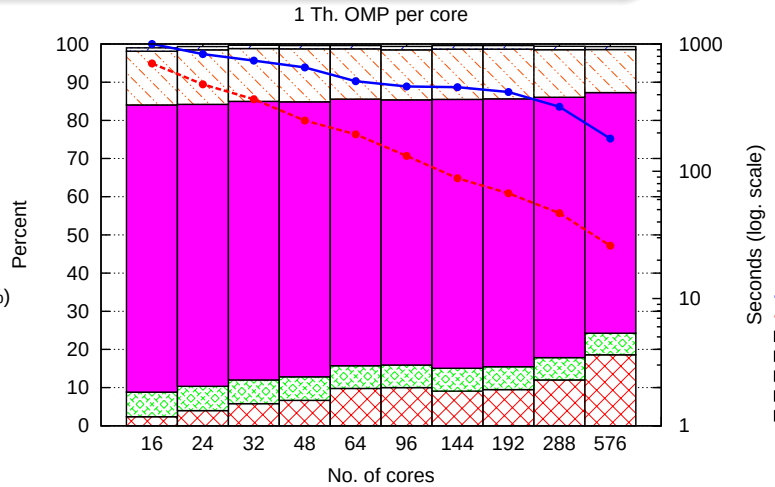
User viewpoint
- Frequent mistakes
- Performance evaluation
- GPU
- Practical cases

Conclusion

Same code, same runs. Which is the best?



CCRT Titane (Nehalem, Infiniband)



CSCS Rosa (Opteron, Cray XT5)

Titane is 2.3 to 1.6 times faster than Rosa!

Degradation of parallel performances: why?

1. Calculation power has increased more than networking
2. Better libraries (MKL)

☛ Walltime reduced, but lower parallel efficiency

This will always happen while using GPU!

Architectures, libraries, networking

HPC and BigDFT

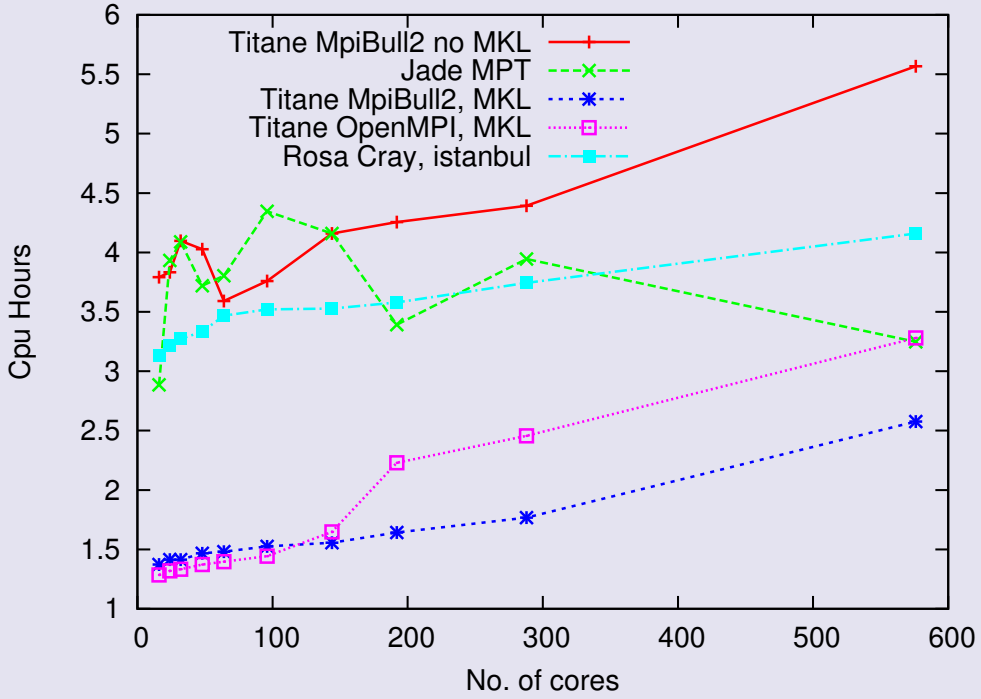
Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Same runs, same sources; different user conditions



Differences up to a factor of 3!

A case-by-case study

Consideration are often system-dependent, a thumb rule not always exists.

☛ Know your code!

A (even more) frequent mistake

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Example: two DFT codes running on the same system.

Naive question: Which one is *faster*?

The running conditions

- Machine generation (CPU, cores, cache,. . .)
- Parallel environment (MPI procs, OMP threads, GPUs)
- Binaries (libraries, compiler,. . .)
- Network vs. Computation performance

The code conditions (DFT example)

- Basis set (formalism, cut-off, . . .)
- Self-Consistentcy (Input Guess, minimization scheme)

How this question should be posed?

- Which is lowest time-to-solution possible for this system on a given machine?
- Which is the fastest machine for this system?

Using GPUs in a given (DFT) code

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Developer and user dilemmas

- Does my code fits well? For which systems?
- How much does porting costs?
- Should I always use GPUs?
- How can I interpret results?

Evaluating GPU convenience

Three levels of evaluation

1. Bare speedups: GPU kernels *vs.* CPU routines
   Does the operations are suitable for GPU?
2. Full code speedup on one process
   Amdahl's law: are there hot-spot operations?
3. Speedup in a (massively?) parallel environment
   The MPI layer adds an extra level of complexity

The time-to-solution problem I: Efficiency

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

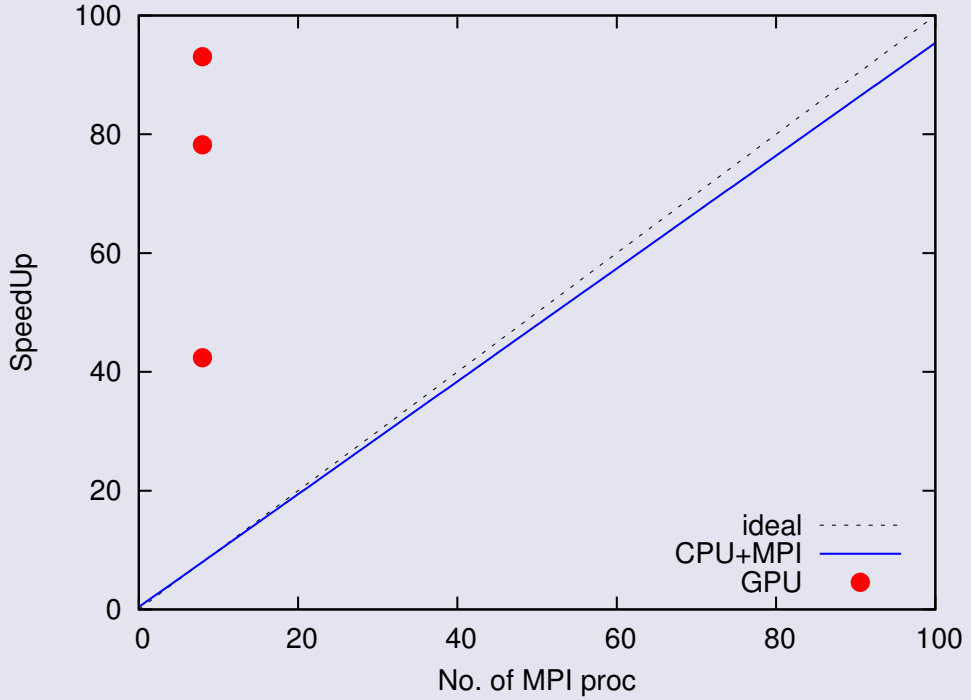
Conclusion

Good example: 4 C at, surface BC, 113 Kpts

Parallel efficiency of 98%, convolutions largely dominate.

Node:
2× Fermi + 8 × Westmere
8 MPI processes

# GPU added	2	4	8
SpeedUp (SU)	5.3	9.8	11.6
# MPI equiv.	44	80	96
Acceler. Eff.	1	.94	.56



The time-to-solution problem II:Robustness

HPC and BigDFT

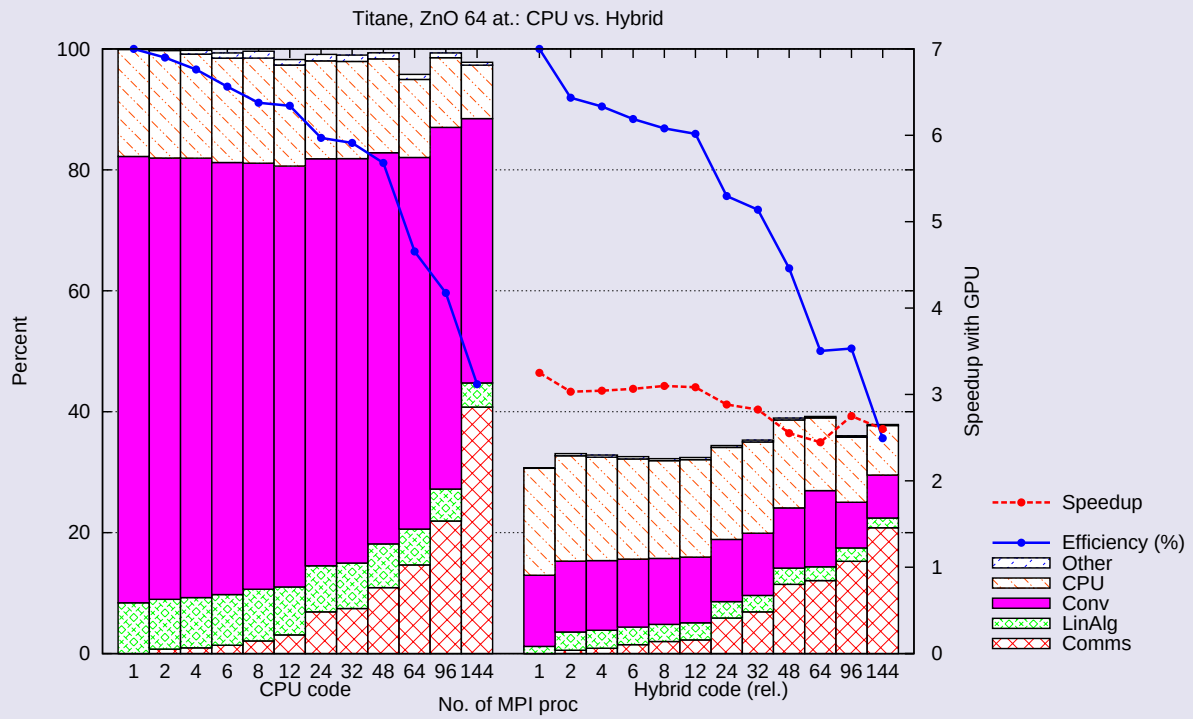
Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Not so good example: A too small system



- ✘ CPU efficiency is poor (calculation is too fast)
- ✘ Amdahl's law not favorable (5x SU at most)
- ✔ GPU SU is almost independent of the size
- ✔ Users' keyword: robustness and reliability

Hybrid and Heterogeneous runs with OpenCL

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

NVidia S2070

Connected each to a Nehalem Workstation

BigDFT may run on both

ATI HD 6970

Sample BigDFT run: Graphene, 4 C atoms, 52 kpts

No. of Flop: $8.053 \cdot 10^{12}$

MPI	1	1	4	1	4	8
GPU	NO	NV	NV	ATI	ATI	NV + ATI
Time (s)	6020	300	160	347	197	109
Speedup	1	20.07	37.62	17.35	30.55	55.23
GFlop/s	1.34	26.84	50.33	23.2	40.87	73.87

Next Step: handling of Load (un)balancing

A rapidly evolving situation

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Architecture evolutions

- Manycore era (multilevel parallelisation)
- Memory traffic as *the* limiting factor

Software evolutions

- Superposition of parallelization layers
- Optimization issues: maintainability vs. robustness

Users ability

- Architecture dimensioning: adapt the runs to the system
- Performance evaluation approach

And it is not going better:

- New set of architectures (GPU, MIC, BG/Q,. . .)
- New development paradigms (MPI, OpenMP, OpenCL,. . .)
- HPC codes must follow (HPC projects, Users how-to,. . .)

. . . and algorithms

$O(N)$ approach (traditional $O(N^3)$)

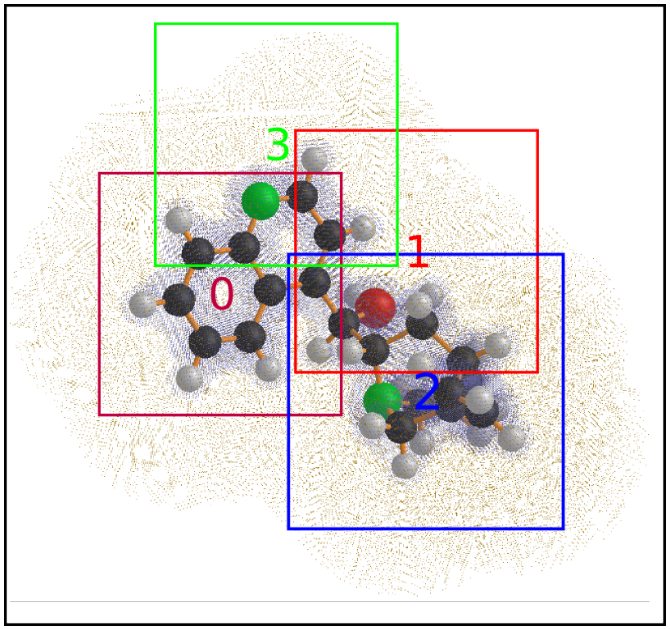
HPC and BigDFT

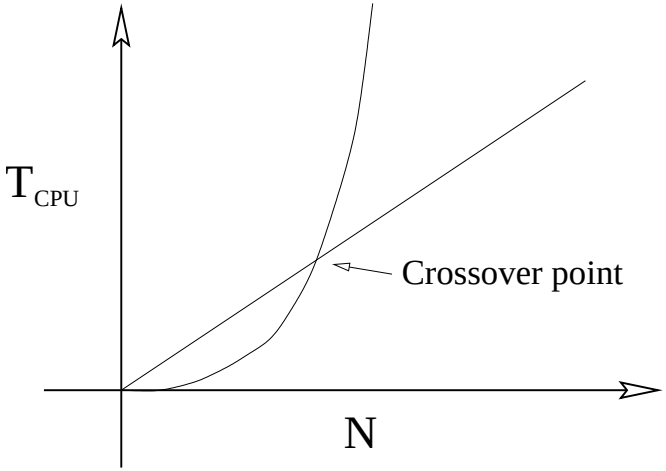
Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion





Formalism Advantage

- Use locality of wavelets
- Localization regions
- Better flexibility
- Different schemes to localise wavefunctions

Where are we?

- A prototype version validated since 2007
- New localisation schemes already tested with cubic paradigm
- ☛ Underlying infrastructure ready, soon in production

A look in near future: science with HPC DFT codes

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

Enhancing BigDFT functionalities

- PAW formalism
  Should further reduce computational overhead
- $\mathcal{O}(N)$ approach, production code
  Possible thanks to wavelets localisation and orthogonality
- New parallelisation scheme suitable for very large platforms
- Further refine formalims for Quantum Chemistry
  Systematic basis set extension for accurate treatment

The Mars mission

Is Petaflop performance possible?

- Multilevel parallelization $\rightarrow$ one order of magnitude
- Bigger systems, heavier methods $\rightarrow$ (more than) one order of magnitude bigger

General considerations

HPC and BigDFT

Architectures
Software problem
HPC nowadays
Memory bottleneck

Developer approach
Present Situation
Optimization

User viewpoint
Frequent mistakes
Performance evaluation
GPU
Practical cases

Conclusion

What is desirable? (Does it *open* new directions?)

Performance should lead to improvements

Optimisation effort

- Know the code behaviour and features
  Careful performance study of the complete algorithm
- Identify and *make modular* critical sections
  Fundamental for mainainability and architecture evolution
- Optimisation cost: consider *end-user* running conditions
  Robustness is more important than best performance

Performance evaluation know-how

- No general thumb-rule: what means High Performance?
  A multi-criterion evaluation process
- Multi-level parallelisation always to be used
  Your code will not (anymore) become faster via hardware