Daubechies Wavelets in Electronic Structure Calculation: BigDFT Code Tutorial

ORNL/NICS – OAK RIDGE, TENNESSEE

BigDFT approach to High Performance Computing

Luigi Genovese

L_Sim – CEA Grenoble

February 8, 2012

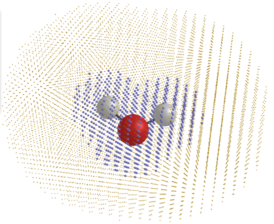
Outline

1. The code
   - Formalism and properties
   - The needs for hybrid DFT codes
   - Main operations, parallelisation
2. Performance evaluation
   - Evaluating GPU gain
   - Practical cases
3. Discussion

A DFT code based on Daubechies wavelets

BigDFT: a PSP Kohn-Sham code

A Daubechies wavelets basis has unique properties for DFT usage



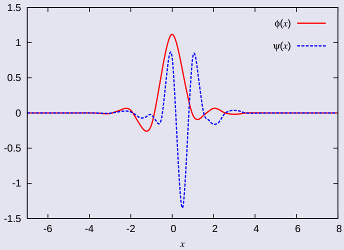
- Systematic, Orthogonal
- Localised, Adaptive
- Kohn-Sham operators are analytic

Short, Separable convolutions

$$\tilde{c}_\ell = \sum_j a_j \, c_{\ell-j}$$

- Peculiar numerical properties

Daubechies Wavelets



Real space based, highly flexible

Big & inhomogeneous systems

Basis set features

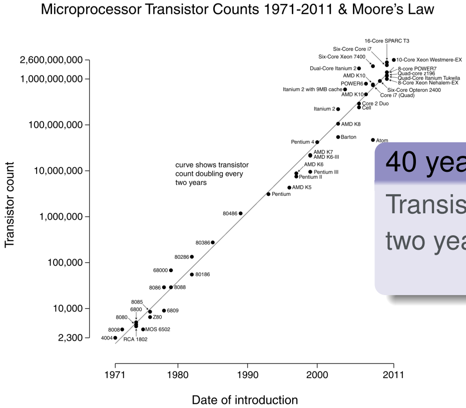
BigDFT features in a nutshell

- ✔ Arbitrary absolute precision can be achieved
  Good convergence ratio for real-space approach ($\mathcal{O}(h^{14})$)
- ✔ Optimal usage of the degrees of freedom (adaptivity)
  Optimal speed for a systematic approach (less memory)
- ✔ Hartree potential accurate for various boundary conditions
  Free and Surfaces BC Poisson Solver
  (present also in CP2K, ABINIT, OCTOPUS)
- ☛ Data repartition is suitable for optimal scalability
  Simple communications paradigm, multi-level parallelisation possible (and implemented)

Improve and develop know-how

Optimal for *advanced* DFT functionalities in HPC framework

Moore's law


Microprocessor Transistor Counts 1971-2011 & Moore's Law

40 years of improvements

Transistor counts double every two years...

...but how?

Power is the limiting factor

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

Multiple slower devices *preferable* than one superfast device

☛ More performance with less power → software problem?

Hybrid Supercomputing nowadays

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)?

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 × 2000 cores × 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?)

Operations performed

The SCF cycle

Orbital scheme:

- Hamiltonian
- Preconditioner

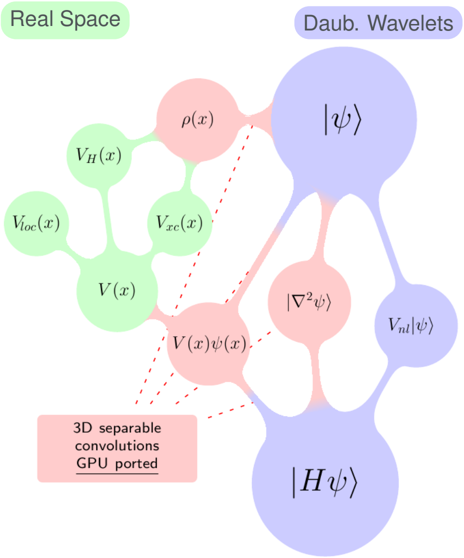
Coefficient Scheme:

- Overlap matrices
- Orthogonalisation

Comput. operations

- Convolutions
- `BLAS` routines
- FFT (Poisson Solver)

Why not GPUs?



3D separable convolutions GPU ported

Separable convolutions

We must calculate

$$F(l_1,l_2,l_3) = \sum_{j_1,j_2,j_3=0}^{L} h_{j_1} h_{j_2} h_{j_3} G(l_1 - j_1, l_2 - j_2, l_3 - j_3)$$

$$= \sum_{j_1=0}^{L} h_{j_1} \sum_{j_2=0}^{L} h_{j_2} \sum_{j_3=0}^{L} h_{j_3} G(i_1 - j_1, i_2 - j_2, i_3 - j_3)$$

Application of three successive operations

1. $A_3(l_3, i_1, i_2) = \sum_j h_j G(i_1, i_2, l_3 - j) \qquad \forall i_1, i_2;$
2. $A_2(l_2, l_3, i_1) = \sum_j h_j A_3(l_3, i_1, l_2 - j) \qquad \forall l_3, i_1;$
3. $F(l_1, l_2, i_3) = \sum_j h_j A_2(l_2, l_3, l_1 - j) \qquad \forall l_2, l_3.$

Main routine: Convolution + transposition

$$F(I, a) = \sum_j h_j G(a, I - j) \qquad \forall a\,;$$

CPU performances of the convolutions

Initially, naive routines (FORTRAN?)

$$y(j,I) = \sum_{\ell=L}^{U} h_\ell x(I+\ell, j)$$

- Easy to write and debug
- Define reference results

```
do j=1,ndat
  do i=0,n1
    tt=0.d0
    do l=lowfil,lupfil
tt=tt+x(i+l,j)*h(l)
    enddo
    y(j,i)=tt
  enddo
enddo
```

Optimisation can then start (Ex. X5550,2.67 GHz)

Method	GFlop/s	% of peak	SpeedUp
Naive (FORTRAN)	0.54	5.1	1/(6.25)
Current (FORTRAN)	3.3	31	1
Best (C, SSE)	7.8	73	2.3
OpenCL (Fermi)	97	20	29 (12.4)

How to optimize?

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

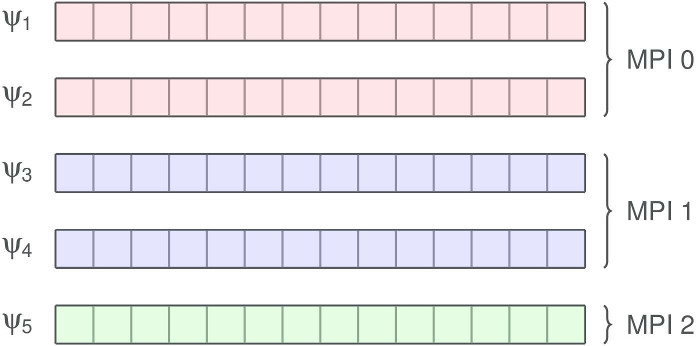
- Only one out of 3 convolution type has been implemented
- About 20 different patterns have been studied for one 1D convolution
- Tedious work, huge code ⟶ Maintainability?

☛ Automatic code generation under study

MPI parallelization I: Orbital distribution scheme

Used for the application of the hamiltonian

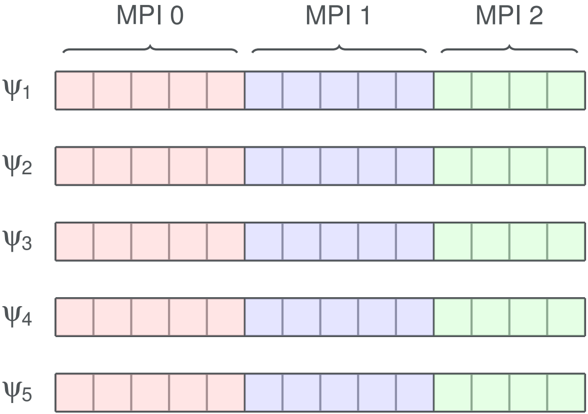
Operator approach: The hamiltonian (convolutions) is applied separately onto each wavefunction



MPI parallelization II: Coefficient distribution scheme

Used for scalar products & orthonormalisation

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



At present, MPI_ALLTOALL(V) is used to switch

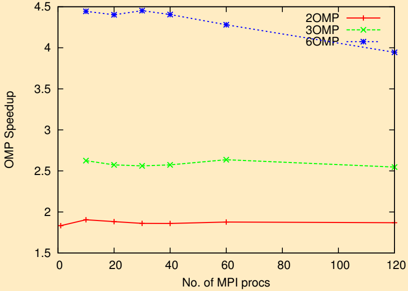
OpenMP parallelisation

Innermost parallelisation level

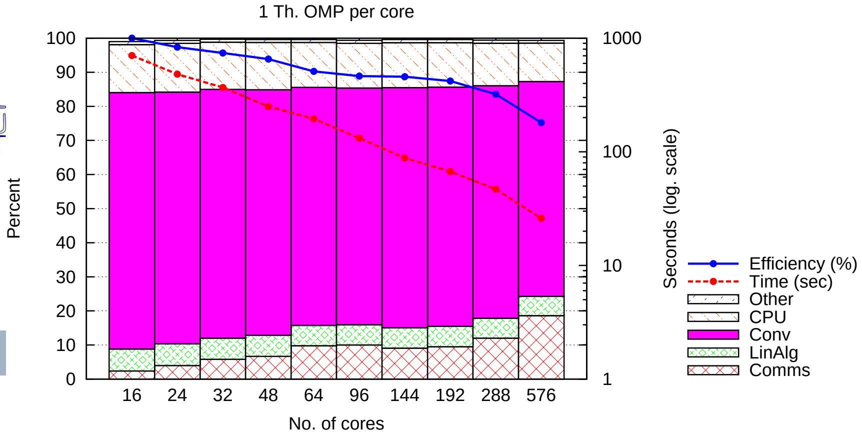
(Almost) Any BigDFT operation is parallelised via OpenMP

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

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



Task repartition for a small system (ZnO, 128 atoms)



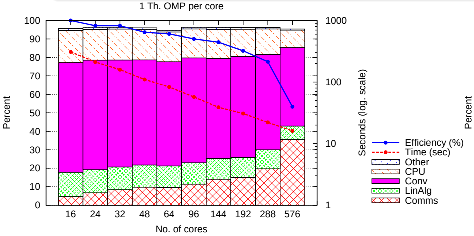
What are the ideal conditions for GPU

GPU-ported routines should take the majority of the time

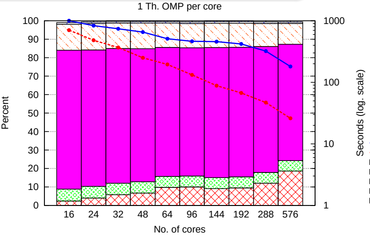
What happens to parallel efficiency?

Parallelisation and architectures

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?

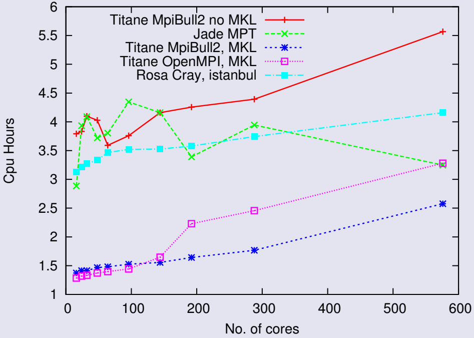
1. Calculation power has increased more than networking
2. Better libraries (MKL)

☛ Walltime reduced, but lower parallel efficiency

This will always happen while using GPU!

Architectures, libraries, networking

Same runs, same sources; different user conditions



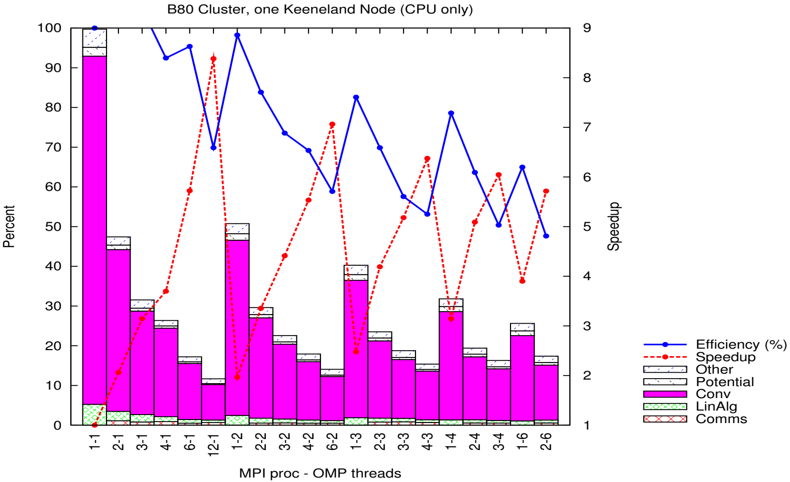
Differences up to a factor of 3!

A case-by-case study

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

☛ Know your code!

Intranode bandwidth problem



Scalability does not depend only on communication

Amdahl's law is a upper limit!

Using GPUs in a Big systematic DFT code

Nature of the operations

- Operators approach via convolutions
- Linear Algebra due to orthogonality of the basis
- Communications and calculations do not interfere

☛ A number of operations which can be accelerated

Evaluating GPU convenience

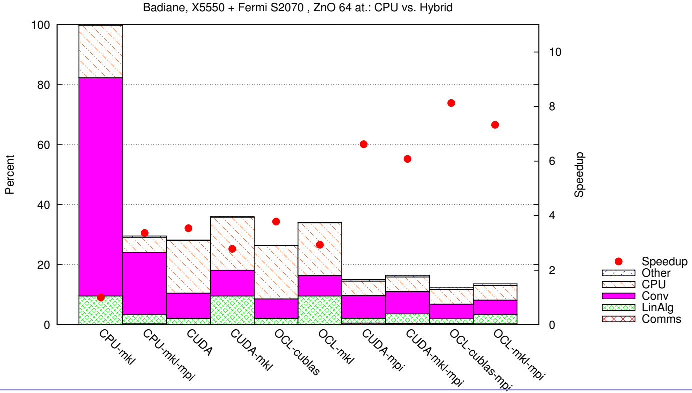
Three levels of evaluation

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

BigDFT in hybrid codes

Acceleration of the full BigDFT code

- Considerable gain may be achieved for suitable systems
  Amdahl's law should always be considered
- Resources can be used concurrently (OpenCL queues)
  More MPI processes may share the same card!



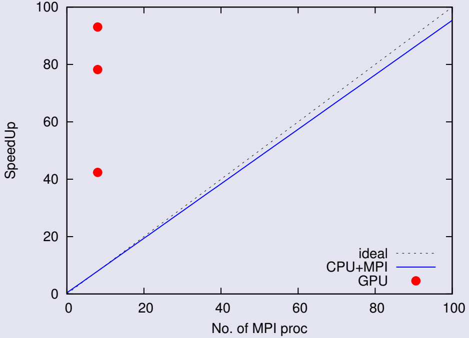
The time-to-solution problem I: Efficiency

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

Parallel efficiency of 98%, convolutions largely dominate.

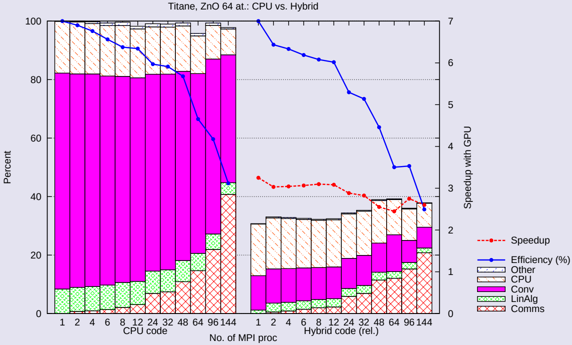
Node:
2× Fermi + 8 ×
Westmere
8 MPI processes

# 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

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
- ✔ The hybrid code *always* goes faster

Hybrid and Heterogeneous runs with OpenCL

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 look in near future: science with HPC DFT codes

A concerted set of actions

- Improve codes functionalities for present-day and next generation supercomputers
- Test and develop new formalisms
- Insert ab-initio codes in new scientific workflows (Multiscale Modelling)

The Mars mission

Is Petaflop performance possible?

- GPU acceleration → one order of magnitude
- Bigger systems, heavier methods → (more than) one order of magnitude bigger

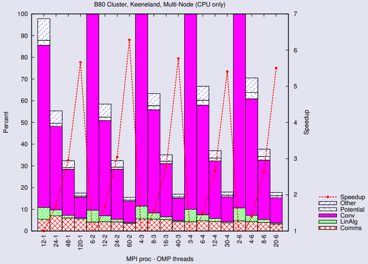
BigDFT experience makes this feasible

An opportunity to achieve important outcomes and know-how

Exercises of this afternoon

Profiling and interpreting BigDFT result

- Dimension job size
- Interpret and distinguish intranode and internode behaviour



BigDFT usage of GPU resources

- Combining MPI, OpenMP and GPU acceleration