ABINIT ON PRE-EXASCALE COMPUTERS:
HYBRID PARALLELISM
AND NUMERICAL STABILITY

Marc Torrent, Jordan Bieder
CEA, DAM, DIF, Bruyères-le-Châtel, France

Yohan Chatelain, Pablo Oliveira
Versailles/St-Quentin-en-Yvelines University, France
OUTLINE

ABINIT

Overview of the project
Many-core architectures: development strategy

ABINIT – Porting on many-core architectures

Diagonalization algorithm
Hybrid parallelism and hardware topology
Abstraction layer

ABINIT - Numerical stability

Origin of numerical instabilities
Verifitracer tool and application to ABINIT

Performances
ABINIT
OVERVIEW OF THE PROJECT
ABINIT – OVERVIEW

- **Open-source** package, freely distributed on the web
  Initiated in 1997 in *Louvain-la-Neuve* (Belgium)

- GNU-GPL license; written in **Fortran 2008/2011**.
  Determination of the total energy and the electronic structure of systems made of electrons and nuclei (periodic systems)

- **Density-Functional Theory**
  - **Pseudo-potential** formalism (**Projector Augmented-Wave**)
  - **Plane-wave** basis (native) or wavelet basis (**BigDFT**)

- *Ground state, excited states*, linear and non-linear *response function* (**DFPT**), *structural relaxation*, *molecular dynamics*, …

- **“Beyond DFT”** theories: *Many Body Perturbation Theory*,
  *Dynamical Mean-Field Theory*, *Hybrid* functionals,
  *Time-dependent DFT*,…
ABINIT – OVERVIEW

- Implemented basic theories
  - **Density-Functional Theory** (DFT)
    - Ground state properties: energy, charge density, forces...
    - Structure relaxation, Molecular Dynamics
  - **Density-Functional Perturbation Theory** (DFPT)
    - Phonons, elastic tensor, dielectric properties, non-linear response...
  - **Many-Body Perturbation Theory** (MBPT)
    - Time-Dependent Density-Functional Theory (TDDFT)
    - Excited states, spectroscopy
  - **Dynamical Mean Field Theory** (DMFT)
    - Electronic correlations
A “general” DFT software
- Can handle all materials (metal, insulator, magnetic, …)
- Scalability (DFT): \( \sim N_{\text{electron}}^3 \)

A “software package”
- One driver, several formalisms: DFT, DFPT, MBPT, DMFT, …
- One “ecosystem”: post-processing tools, Python environment, …
- Atomic data tables

A code for research…
- GPL v3 license
- Open to any contribution, anybody can contribute
ABINIT – SOME DATA

As of May 2018:
1160 F90 files
1188 autom. Tests
900 kLines of F90
76 Mbytes tar.gz

~1800 registered users
90 names if file “contributors”
25 active developer branches

Used for teaching (>30 on-line tutorials)
Installed in most of the computing centers
Used as benchmark by computing centers
Training schools regularly organized
ABINIT – COLLABORATIVE DEVELOPMENT

- **Continuous Integration (CI)** development workflow
  - Extensive test suites (1200 automatic tests)
  - Daily reviewing of contributions
  - Code quality checking
  - Computer farm management

- Management with **git, gitlab, buildbot**

- Regular developer meetings

- Advisory Committee

**Active developer groups**
- Louvain-la-Neuve university (Belgium)
- Liège university (Belgium)
- CEA, Bruyères-le-Châtel (France)
- CEA, Grenoble (France)
- CEA, Saclay (France)
- Montréal university (Canada)
- Dalhousie university, Halifax, (Canada)
- Rutgers university, New-Jersey (USA)
- Santander university (Spain)
- West Virginia university (USA)
- Bogota university (Colombia)
MANY-CORE ARCHITECTURES: DEVELOPMENT STRATEGY
Message Passing Interface (MPI)
Computational load distribution
- Cells, k-pts, spins/spinors, atoms, bands, plane waves
- Each level has its own parallel efficiency
Distribution of data
- Cells, k-pts, spins/spinors, atoms, bands, plane waves
- Only collective communications used
No computation/communication overlap optimization

Vectorisation
- 70% (lines)

OpenMP
- All internal loops (low level)
- Linear and Matrix Algebra, FFT

Bottlenecks for many-core architectures:
ABINIT is **memory or latency bound**
Communications, data locality, memory access
1. Improve the **scalability** of the diagonalization algorithm
   *More calculation, less storage, less communications*

2. Efficiently use the **shared memory** *(openMP)*
   in a « medium grained » mode
   *Adapt the code to the hardware topology*
   *Give longer tasks to the elementary computing units*
   *Decrease the data movements*

3. **Externalize** the elementary operations
   *Express the physics in terms of elementary operations*
   *Use vendor (or optimized) libraries*

4. Add an **abstraction** layer
   *Isolate low level optimized operations*

---

*Decrease Time To Solution*
INTERNAL ALGORITHM SCALABILITY

Repartition of time in a ground-state calculation varying the number of band CPU cores (MPI, strong scaling)

TEST CASE
A vacancy in a 108 atoms cell (gold)
Gamma k-point only, PAW
Computation of total energy and forces

2016

Iterative solver (eigenvalues)
Without Hamiltonian application

Hamiltonian application
Linear algebra, FFT

PASC 2018 – July 2, 2018
Parallelism efficiency was dominated by the scalability of the eigensolver.

Target computations: $1000 < N_{\text{band}} < 50,000$, $N_{PW} \sim 100,000 \ldots 250,000$

Direct diagonalization unachievable ($\sim 10^{12}$)

In search of the eigenvectors associated with the lowest eigenvalues.

Need an **iterative algorithm**.

Different kind of algorithms but the elementary bricks are the same:

- **Hamiltonian application** (linear algebra, FFT) $\Rightarrow$ computation
- **Rayleigh-Ritz procedure** (linear algebra, diago, ortho) $\Rightarrow$ communication
INTERNAL ALGORITHM SCALABILITY
CHEBYSHEV FILTERING (ITERATIVE DIAGO)

Scalability of ABINIT internal algorithm (bands+FFT)

**ABINIT 1995**
- 10 tasks
- Conjugate gradient
- 1 H application per iteration
- 1 complete orthogonalisation per iteration

**ABINIT 2007**
- 300 tasks
- Block conjugate gradient (LOBPCG)
- 1 H application per iteration
- 1 partial orthogonalisation per iteration
- 1 complete orthogonalisation

**ABINIT 2016**
- 15 000 tasks
- Chebyshev filtering
- 5 H applications per iteration
- 1 complete orthogonalisation

**ABINIT 2019**
- 100 000 tasks?
- Slicing
- 20 H applications per iteration
- No orthogonalisation
**Chebyshev filtering**

An « old » algorithm modernized (U. Texas)

The eigenvalue spectrum is projected in the area of interest

---

**ABINIT – MPI only**

TGCC-Curie – Sandy Bridge

**Block Conjugate Gradient (LOBPCG)**

vs Chebyshev Filtering

---

EFFICIENT USE OF SHARED MEMORY

- Adopt a *coarse grained* model to « feed » the threads

- Connect physics and *hardware topology*
  Match electrons and computer nodes?

- Identify « sharable » data

- Ensure *thread safety*
  *Not obvious for external libraries (FFT, Scalapack, …)*
  *Not obvious when MPI is activated*
EFFICIENT USE OF SHARED MEMORY, ADAPT THE CODE TO THE HARDWARE TOPOLOGY

One eigenvalue determination

BEFORE

Elementary operations


AFTER

Several eigenvalues determination

Comm.
EFFICIENT USE OF SHARED MEMORY, MEDIUM GRAINED MODEL

\[ H = -\frac{1}{2}\Delta + V_{\text{loc}}[\rho] + PV_{\text{nonloc}}[\rho]P^T \]

- **Fine-grained**: previous code = low-level multithreading
- **Coarse-grained**: Multi-threaded *Hamiltonian* application for one band

Actually implemented

- **Medium-grained model**: Multi-threaded *Hamiltonian* application called for several bands (MPI)
- Use hybrid GEMM (*MPI*+*openMP*)
- Use multi-threaded-FFT in « batch » mode
EFFICIENT USE OF SHARED MEMORY, BLOCK CONJUGATE GRADIENT (LOBPCG)

Elementary bricks of the algorithm:
- Application of $H = \text{transposition} + \text{computation}$
- Rayleigh-Ritz procedure = diago in subspace + orthogonalization

New multi-threaded implementation greatly reduces the cost of orthogonalizations

New multi-threaded implementation greatly reduces the size of transpositions ($\text{alltoall}$ MPI calls)

FFT parallelization level fully done with openMP (apart batch mode)

New implementation is usable with only one block, or a few blocks

- The cost of global orthogonalization is reduced
- The algorithm is numerically more stable
- The number of SCF iterations is strongly reduced
### Old implem. vs new implem.

<table>
<thead>
<tr>
<th></th>
<th>Nehalem</th>
<th>Haswell</th>
<th>KNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process x threads</td>
<td>500 x 1</td>
<td>500 x 1</td>
<td>50 x 4</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>63</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>CPU time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old implementation</td>
<td>45 min.</td>
<td>27 min.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>36 it.</td>
<td>36 it.</td>
<td></td>
</tr>
<tr>
<td>CPU time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New implementation</td>
<td>10 min.</td>
<td>5 min.</td>
<td>8 min.</td>
</tr>
<tr>
<td></td>
<td>8 it.</td>
<td>8 it.</td>
<td>8 it.</td>
</tr>
</tbody>
</table>

**Test case**
- 64 Pu atoms
- 1200 electronic bands
- Tera1000 CEA supercomputer

### Performances – 1 KNL node

<table>
<thead>
<tr>
<th># MPI proc.</th>
<th># threads</th>
<th># blocks</th>
<th>CPU (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>32</td>
<td>1</td>
<td>188</td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>2</td>
<td>205</td>
</tr>
<tr>
<td>36</td>
<td>8</td>
<td>4</td>
<td>231</td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>8</td>
<td>360</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># MPI proc.</th>
<th># threads</th>
<th># blocks</th>
<th>CPU (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>570</td>
<td>2</td>
<td>1</td>
<td>1091</td>
</tr>
<tr>
<td>285</td>
<td>4</td>
<td>1</td>
<td>918</td>
</tr>
<tr>
<td>143</td>
<td>8</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>72</td>
<td>16</td>
<td>1</td>
<td>248</td>
</tr>
<tr>
<td>36</td>
<td>32</td>
<td>1</td>
<td>188</td>
</tr>
<tr>
<td>18</td>
<td>64</td>
<td>1</td>
<td>197</td>
</tr>
</tbody>
</table>
ABSTRACTION LAYER - CONCEPT

**User layer**

\[ \psi_k = \sum_g c_g e^{i(k+G) \cdot r} \]

**Physicist layer**

DFT: Solve \( \hat{H} \ket{\psi} = \epsilon \ket{\psi} \)

DFPT: Solve \( (\hat{H}^{(0)} - \epsilon_n) \ket{\psi_n^{(\lambda)}} = -\hat{H}^{(\lambda)} \ket{\psi_n^{(0)}} \)

DMFT: Project to local basis with \( p_{mn}(k) = \langle \chi^R_{km} | \psi_{kn} \rangle \)

Others

**Interface with low-level**

Memory: xgBlock_init, xgBlock_free, xgBlock_map, xgBlock_reverseMap, ...

BLAS interface: xgBlock_gemm, xgBlock_axpy, ...

LAPACK interface: xgBlock_potrf, xgBlock_trsm, xgBlock_heev, ...

**Low level**

Mkl
- mkl/magma/plasma
- MIC/KNL
- GPU

cuBLAS
- cuBLAS
- cuFFT
- magma/plasma

MKL
- MKL
- ELPA
- FFTW
- Haswell
ABSTRACTION LAYER

- All wave functions operations have been encapsulated and are now hidden for the developers.

- ABINIT specificities are taken into account:
  - Real and complex wave functions (according to physical context)
  - Compressed storage according to symmetries

- Low level operations are implemented in different langages according to computing architectures.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>MIC</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory allocation</td>
<td>malloc()</td>
<td>mbw_malloc()</td>
<td>cudaMalloc()</td>
</tr>
<tr>
<td>Initialization</td>
<td>memcpy()</td>
<td>memcpy()</td>
<td>cudaMemcpy()</td>
</tr>
<tr>
<td>GEMM</td>
<td>XGEMM</td>
<td>MAGMA/XGEMM</td>
<td>Cublas&lt;T&gt;::gemm()</td>
</tr>
</tbody>
</table>
**ABSTRACTION LAYER**

- Memory management: 1 unique contiguous memory space allocated
  No need for any other allocation
  
  ```
  call xg_init(XWP, space, spacedim, 3*blockdim, spacecom)
  ```

- Easy direct access via pointers: access to wave functions individually (vectors) or collectively (matrix)
  
  ```
  call xg_setBlock(XWP, X, 3, spacedim, blockdim)
  ```

- Storage mode hidden: complex, compressed, ...
  Easy memory resizing (memory blocks)

- Linear algebra:
  - Use of external libraries (mkl) maximized
  - ScaLAPACK hidden

- Fast Fourier Transforms:
  - Handle individual or batch modes
  - Handle ABINIT specific storage
  - FFT automatic selection (mkl, dfti, fftw, ...)

018
subroutine Borthonormalize(lobpcg,X,AX,BX)
call xg_init(buffer,space(X),cols(X),cols(X),space(X))

! Compute X^TBX
call xgBlock_gemm(X%trans,BX%normal,1.do,X,BX,0.do,buffer%self)

! Compute Cholesky decomposition (Upper part)
call xgBlock_potrf(buffer%self,'u',info)

! Solve YU=X
call xgBlock_trsm('r','u',buffer%normal,'n',1.do,buffer%self,X)
! Solve BYU=BX
call xgBlock_trsm('r','u',buffer%normal,'n',1.do,buffer%self,BX)
! Solve AYU=AX
call xgBlock_trsm('r','u',buffer%normal,'n',1.do,buffer%self,AX)

call xg_free(buffer)
end subroutine
MANY-CORE VECTOR ARCHITECTURES: NUMERICAL STABILITY
Issue: on modern many-core architectures
ABINIT shows numerical instabilities
- Continuous numerical deviations
- Isolated steps unconverged

2 causes identified – 2 fixes
- Vectorization leads to unpredictable MPI processes desynchronization
  Fix: synchronize MPI processes
- Vectorization+multithreading leads to unpredictable floating point operations precision
  Fix: use of verificarlo/verifitracer
Open Source Project under GPL licence, developed by University of Versailles and ENS Paris-Saclay

- Automatically analyses the numerical stability of applications
- Introduces a noise on each floating-point operation
\[ \text{inexact}(x) = x + \beta^{e_x-t} \xi \]

- \(e_x\) is the magnitude of \(x\)
- \(t\) the virtual precision
- \(\xi\) a random variable between \([ -\frac{1}{2}, \frac{1}{2} ]\)

**Random Rounding mode**

\[ \text{RR}(x \circ y) = \text{inexact}(x \circ y) \]

\( \epsilon \in [+, -, x, \cap, \cap] \)

### Diagram

- \( z = x \circ y \)
- \( 2^{e_x-t} \xi \)
- \( \text{inexact}(z) \)
- \( t = 21 \)
- Trace the precision of FP variables over time
- Provide contextual information on traced variables
  - Current version implements Context Analysis
  - Inserts call to `backtrace()` (GNU C library)
  - Analyzes backtraces to detect flow divergences
- All in an automatic way
ABINIT+VERIFITRACER

Very simple test case: hydrogen molecule
Test case: BaTiO₃ perovskite

Functions below 23bits
- are more resistant
- can potentially be changed to single precision

Functions above 23bits
- are less resistant
- require further analyses

Interesting function among the top 5
__m_pawrad_MOD_simp_gen
A simple Simpson integration...

- A simple dot product
- Several regimes

**Figure 4:** Evolution of the value returned by simp_gen with $t = 53$ in Random Rounding mode with 24 samples in parallel on the CINES Occigen cluster (2106 nodes x2 Intel 12-Cores (E5-2690V3@2.6 GHz))
A simple Simpson integration...

- Replace the dot product by a compensated version
- Implemented with libeft
- Precision improved
- 1 Call Site Path (CSP) has a low precision due to reentrance of the error
ABINIT: PERFORMANCES
Uranium
128 atoms
1600 bands (3200 electrons)

Old implementation
MPI only
CEA-TGCC Curie
Nehalem
16 cores/node

Hamitonian

New implementation
MPI x 24 threads
CEA Tera1000-2
Intel KNL
64 cores/node
Plutonium
108 atoms
2300 bands \(4600\) electrons

**CEA Tera1000-1 ➤ Intel Haswell (32 cores/nodes)**

**CEA Tera1000-2 ➤ Intel KNL (64 cores/node)**
PERFORMANCES

Gallium oxide $\text{Ga}_2\text{O}_3$
1960 atoms
8700 bands (**17400 electrons**)
Time per SCF iteration

Bands + FFT parallelism only
*Can be mixed to other parallelism levels*

**TGCC –Joliot-Curie**

**Intel Skylake**

48 cores/node – 2 sockets

PASC 2018 – July 2, 2018
Gallium oxide Ga$_2$O$_3$
4160 atoms
18400 bands (**36800 electrons**)
Time per SCF iteration

Bands + FFT parallelism only
*Can be mixed to other parallelism levels*

**Diagram:**
- Chebyshev filtering - Pure MPI
- Block CG (LOBPCG) - 24 threads

**TGCC – Joliot-Curie**
**Intel Skylake**
48 cores/node – 2 sockets
CONCLUSION
**KEY POINTS**

- Porting strategy (4 items) on Many-core architectures
  
  *Paid off*

- Beyond the simple « arithmetic » gain
  
  *Numerical stability of the algorithm improved*
  
  *Time to solution decreased*

- Numerical precision issues
  
  *Efficient use of Monte-Carlo FP arithmetic*

*Towards exaScale:*

*Band+FFT parallelism mixed with other parallelism(s)*