FROM RESEARCH TO INDUSTRY

# Abinit on new architectures

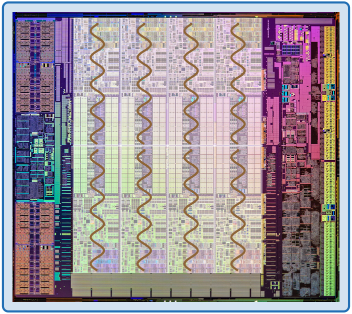
## Example of LOBPCG

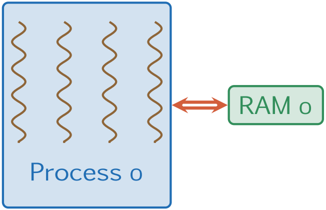
J. Bieder, M. Torrent
CEA-DAM-DIF, Bruyères-le-Châtel

AbiDev 2017

May 10, 2017

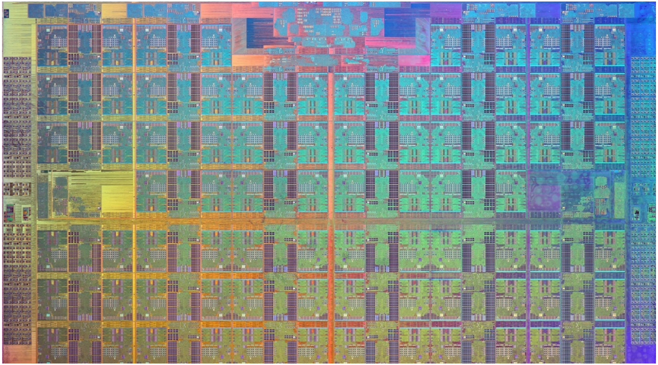
- 1 CPU has several (4 or 6 or 8 or 12 or more) cores
- Each core may have 2 or 4 threads.
- Each core has its own cache memory
- All the cores share the RAM memory





# Target architectures

## In the near future



Intel Many Integrated Cores (MIC) Xeon Phi

→ 64 cores and only 16Go of high speed RAM !

Need Hybrid parallelization

# LOBPCG algorithm

Matrix-free diagonalization procedure $AX = \lambda BX$ Input :

- Procedure to apply the matrices (A,B)
- Procedure to precondition (T)
- $n$ initial linearly independant vectors $X_0$
- Block size $l$

1. Allocate $X, AX, BX, W, AW, BW, P, AP, BP$
2. B-Ortho $X$
3. Rayleigh-Ritz on $\{X\}$
   3.1 Compute $T * W = T * (AX - \lambda BX)$
   3.2 B-Ortho $W$
   3.3 Rayleigh-Ritz on $\{X, W, P\}$

Output : $X$ and $\lambda$

# Concept

Couche utilisateur

Abinit

$$\psi_{\mathbf{k}} = \sum_{\mathbf{g}} c_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}$$

# Concept

Couche utilisateur

Abinit

$\psi_{\mathbf{k}} = \sum_{\mathbf{g}} c_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}$

Couche Physicien

DFT : Solve $\hat{H}\left|\psi\right\rangle = \epsilon\left|\psi\right\rangle$

DFPT : Solve $\left(\hat{H}^{(0)} - \epsilon_n\right)\left|\psi_n^{(\lambda_1)}\right\rangle = -\hat{H}^{(\lambda_1)}\left|\psi_n^{(0)}\right\rangle$

DMFT : Project to local basis with $P_{mn}^{\mathbf{R}}(\mathbf{k}) = \left\langle \chi_{\mathbf{k}m}^{\mathbf{R}} \middle| \psi_{\mathbf{k}n} \right\rangle$

Others

# Concept

Couche utilisateur

Abinit

$\psi_{\mathbf{k}} = \sum_{\mathbf{g}} c_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}$

Couche Physicien

DFT : Solve $\hat{H}\left|\psi\right\rangle = \epsilon\left|\psi\right\rangle$

DFPT : Solve $\left(\hat{H}^{(0)} - \epsilon_n\right)\left|\psi_n^{(\lambda_1)}\right\rangle = -\hat{H}^{(\lambda_1)}\left|\psi_n^{(0)}\right\rangle$

DMFT : Project to local basis with $P_{mn}^{\mathbf{R}}(\mathbf{k}) = \left\langle \chi_{\mathbf{k}m}^{\mathbf{R}} \middle| \psi_{\mathbf{k}n} \right\rangle$

Others

Interface avec la couche d'abastraction

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

BLAS interface : `xgBlock_gemm`, `xgBlock_axpy`, ...

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

# Concept

**Couche utilisateur**

Abinit

$$\psi_{\mathbf{k}} = \sum_{\mathbf{g}} c_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}$$

**Couche Physicien**

DFT : Solve $\hat{H}\left|\psi\right\rangle = \epsilon\left|\psi\right\rangle$

DFPT : Solve $\left(\hat{H}^{(0)} - \epsilon_n\right)\left|\psi_n^{(\lambda_1)}\right\rangle = -\hat{H}^{(\lambda_1)}\left|\psi_n^{(0)}\right\rangle$

DMFT : Project to local basis with $P^{\mathbf{R}}_{mn}(\mathbf{k}) = \left\langle \chi^{\mathbf{R}}_{\mathbf{k}m} \middle| \psi_{\mathbf{k}n}\right\rangle$

Others

**Interface avec la couche d'abstraction**

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

BLAS interface : `xgBlock_gemm`, `xgBlock_axpy`, ...

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

**Couche d'abstraction**

Choix de la cible

- MIC/KNL
  - MKL
  - magma/plasma
- GPU
  - cuBLAS
  - cuFFT
  - magma/plasma
- Haswell
  - MKL
  - ELPA
  - FFTW3

# Memory management notes

- Rayleigh-Ritz on *X* or *XW* or *XWP* allocate only one big continuous array

```
call xg_init(lobpcg%XWP,space,spacedim,3*blockdim,lobpcg%spacecom)
```

- Access to each individual matrix with pointer

```
call xg_setBlock(lobpcg%XWP,lobpcg%X,1,spacedim,blockdim)
call xg_setBlock(lobpcg%XWP,lobpcg%W,blockdim+1,spacedim,blockdim)
call xg_setBlock(lobpcg%XWP,lobpcg%P,2*blockdim+1,spacedim,blockdim)
call xg_setBlock(lobpcg%XWP,lobpcg%XW,1,spacedim,2*blockdim)
call xg_setBlock(lobpcg%XWP,lobpcg%WP,blockdim+1,spacedim,2*blockdim)
```

- No need for any other allocation (except one temporary array in RR procedure)
- Play with abstract layer to reshape, resize memory blocks.
- Real/Complex handled inside the abstract layer (Never see `icplx` or `istwfk` anymore).

# Low level sample

B-orthonormalization : $X \leftarrow X^T BX = 1$

```
call xg_init(buffer,space(X),cols(X),cols(X),lobpcg%spacecom)

! 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)
```

# Higher level Sample

```
do iline = 1, nline
  call lobpcg_getResidu(lobpcg,eigenvaluesN)
  call pcond(lobpcg%W)
  ! Compute residu norm here !
  call xgBlock_colwiseNorm2(lobpcg%W,residuBlock)
  ! Orthonormalize with respect to previous blocks
  ! Apply A and B on W
  call getAX_BX(lobpcg%W,lobpcg%AW,lobpcg%BW)
  ! DO RR in the correct subspace
  if ( iline == 1 ) then
    RR_var = VAR_XW
  else
    RR_var = VAR_XWP
  end if
  call lobpcg_Borthonormalize(lobpcg,RR_var,ierr)
  RR_eig = eigenvalues3N%self
  call lobpcg_rayleighRitz(lobpcg,RR_var,RR_eig,ierr)
end do
```

## Interface with old abinit

Use “map” and “reverse map” technics to reuse already allocated memory

- From abinit to abstract layer

```
call xgBlock_map(xo,cg,space,icplx*npw*nspinor,nband,mpi_comm)
```

- From abstract layer to abinit

```
call xgBlock_reverseMap(X,cg,icplx,spacedim*blockdim)
call xgBlock_reverseMap(AX,ghc,icplx,spacedim*blockdim)
call xgBlock_reverseMap(BX,gsc,icplx,spacedim*blockdim)

call prep_getghc(cg,gs_hamk,gvnlc,ghc,gsc,...)
```

# Old vs. new (1 block)

Test Case : $YNiO_3$ $P21_n$

1 kpt, 408 bands, 1 core → tolwfr $= 10^{-5}$

| Old | New |
|---|---|
| 321.4s | 277.5s |

Test Case : UMo

126 kpt, 140 bands, 340 MPI

| | |
|---|---|
| 736.8s | 707.3s |

# Tests on KNL

Test Case : UMo

126 kpt, 140 bands, 340 MPI (same as before)

| Old | New |
|---|---|
| 163s/iscf | 139s/iscf |

With 272 cores (68 MPI + 4 threadsi/MPI) new version reduced to **120s/iscf**.

Total simulation ≈ 2× slower with 20% less cores.

## More comparaison : node to node

| Test case | cores | threads | Abinit done | Abinit todo | Abinit total |
|---|---|---|---|---|---|
| Au 107 | 32 | 64 | 49.51 | 48.70 | 98.21 |
| | 64 | 64 | 68.87 | 194.61 | 263.49 |
| | 64 | 64 | 64.83 | 103.56 | 174.39 |
| Ti 256 | 32 | 64 | 954.18 | 299.88 | 1254.06 |
| | 64 | 64 | 1120.78 | 420.37 | 1541.15 |
| | 64 | 64 | 882.66 | 418.59 | 1301.25 |
| UO 96 | 32 | 64 | 40.85 | 71.00 | 111.84 |
| | 64 | 64 | 43.68 | 238.41 | 282.09 |
| | 64 | 64 | 40.96 | 160.55 | 201.51 |

Haswell

KNL (DDR)

KNL (MCDRAM)

## What has been done

- Abstract layer
- Reduce memory footprint and cost
- Maximize time spent inside MKL
- Add OpenMP for `getghc` at a very high level → to be changed
- (Optional) remove `abi_xorthonormalize`
- Numerically more stable (?)

# What needs to be done

- Improve getghc:

$$H = \underbrace{1/2\Delta}_{zdot} + \underbrace{V_{loc}}_{\text{FFT batch mode}} + \underbrace{V_{nonloc}}_{zgemm}$$

- Reduce memory manipulations
- Reduce MPI global communication and/or add thread work to increase efficiency
  - prep_getghc
  - prep_symdo
  - prep_symundo
  - prep_nonlop
  - ...

Thank you for your attention