

HPC generation of the Hamiltonian and Overlap matrices in DFT methods based on linearized and augmented plane waves

JLESC, Kobe, December 1st 2016 | Edoardo di Napoli

Motivation

Code modernization

- Legacy codes in Materials Science have grown with focus on functionality
- Frequent problems:
 - Codes lacks modularity, encapsulation, code reuse, ...
 - Codes are often a direct translation of mathematical formulas
- Problems get exacerbated with recent shift to heterogeneous architectures
- “Modernize or perish”
 - Yes, it is costly
 - Yes, it is necessary but benefits are substantial

Goal

Performance portability

In general

- Re-engineering the software by re-thinking the algorithms;
- Modular design, clear layering and interfaces;
- Bottom layers: standardized and highly optimized libraries.

In this talk:

- the FLEUR code as use case
- Modernize a portion of the code that takes about 40% of the computation
- Required an important initial effort
- We now give evidence of performance portability to heterogeneous CPU + GPU architectures

Outline

The FLAPW method & FLEUR code

An exercise in performance portability

Experimental results

Conclusions

Topic

The FLAPW method & FLEUR code

An exercise in performance portability

Experimental results

Conclusions

Density Functional Theory (DFT)

- 1 $\Phi(x_1; s_1, x_2; s_2, \dots, x_n; s_n) \implies \Lambda_{i,a} \psi_a(x_i; s_i)$
- 2 **density of states** $n(\mathbf{r}) = \sum_a f_a |\psi_a(\mathbf{r})|^2$
- 3 In the Schrödinger equation the exact Coulomb interaction is substituted with an effective potential $V_0(\mathbf{r}) = V_I(\mathbf{r}) + V_H(\mathbf{r}) + V_{xc}(\mathbf{r})$

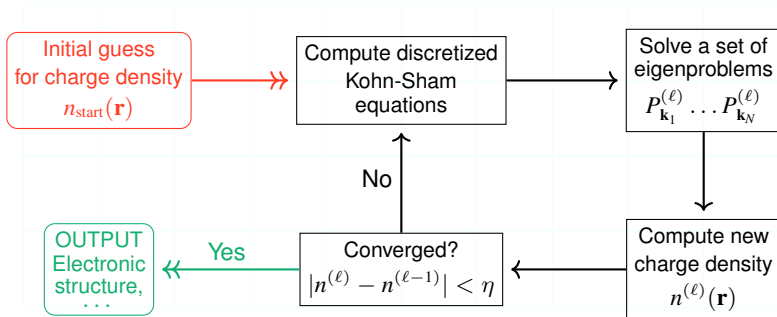
Hohenberg-Kohn theorem

- \exists one-to-one correspondence $n(\mathbf{r}) \leftrightarrow V_0(\mathbf{r}) \implies V_0(\mathbf{r}) = V_0(\mathbf{r})[n]$
- $\exists!$ a functional $E[n] : E_0 = \min_n E[n]$

The high-dimensional Schrödinger equation translates into a set of coupled non-linear low-dimensional self-consistent Kohn-Sham (KS) equation

$$\forall a \quad \text{solve} \quad \hat{H}_{\text{KS}} \psi_a(\mathbf{r}) = \left(-\frac{\hbar^2}{2m} \nabla^2 + V_0(\mathbf{r}) \right) \psi_a(\mathbf{r}) = \epsilon_a \psi_a(\mathbf{r})$$

DFT self-consistent field cycle



Zoo of methods

LDA
 GGA
 LDA + U
 Hybrid functionals
 GW-approximation

Plane waves
 Localized basis set
 Real space grids
 Green functions

$$\left(-\frac{\hbar^2}{2m} \nabla^2 + V_0(\mathbf{r}) \right) \psi_a(\mathbf{r}) = \epsilon_a \psi_a(\mathbf{r})$$

Finite differences
 Non-relativistic eqs.
 Scalar-relativistic approx,
 Spin-orbit coupling
 Dirac equation

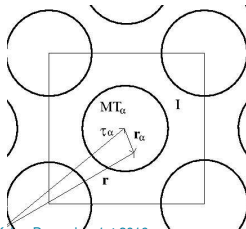
All-electron
 Pseudo-potential
 Shape approximations
 Full-potential
 Spin polarized p calculations

Introduction to FLAPW

LAPW basis set

$$\psi_a(\mathbf{r}) = \sum_G^{N_G} c_{G,i} \varphi_G(\mathbf{r}) \quad i = (\mathbf{k}, \nu) \quad \begin{array}{l} \mathbf{k} \text{ Bloch vector} \\ \nu \text{ band index} \end{array}$$

$$\varphi_G(\mathbf{r}) = \begin{cases} e^{i(\mathbf{k} + \mathbf{G}_t)\mathbf{r}} & \text{INT} \\ \sum_{\ell,m} \left[A_{(l,m)}^{a,G} u_{l,a}(r) + B_{(l,m)}^{a,G} \dot{u}_{l,a}(r) \right] Y_{l,m}(\hat{\mathbf{r}}_a) & a^{\text{th}} \text{ MT} \end{cases}$$



boundary conditions

Continuity of wavefunction and its derivative at MT boundary

$$A_{(l,m)}^{a,G} \quad \downarrow \quad \text{and} \quad B_{(l,m)}^{a,G}$$

Hamiltonian and Overlap matrices

Operatorial form

$$(H)_{G',G} = \sum_a \iint \varphi_{G'}^*(\mathbf{r}) \hat{H}_{\text{KS}} \varphi_G(\mathbf{r}) d\mathbf{r}, \quad (S)_{G',G} = \sum_a \iint \varphi_{G'}^*(\mathbf{r}) \varphi_G(\mathbf{r}) d\mathbf{r}.$$

Entrywise form

$$(S)_{G',G} = \sum_a \sum_{L=(l,m)} \left(A_L^{a,G'} \right)^* A_L^{a,G} + \left(B_L^{a,G'} \right)^* B_L^{a,G} \|\dot{u}_{l,a}\|^2$$

$$\begin{aligned} (H)_{G',G} = \sum_a \sum_{L',L} & \left(\left(A_{L'}^{a,G'} \right)^* T_{L',L;a}^{[AA]} A_L^{a,G} \right) + \left(\left(A_{L'}^{a,G'} \right)^* T_{L',L;a}^{[AB]} B_L^{a,G} \right) \\ & + \left(\left(B_{L'}^{a,G'} \right)^* T_{L',L;a}^{[BA]} A_L^{a,G} \right) + \left(\left(B_{L'}^{a,G'} \right)^* T_{L',L;a}^{[BB]} B_L^{a,G} \right). \end{aligned}$$

Hamiltonian and Overlap matrices

Matrix form

$$H = \sum_{a=1}^{N_A} \underbrace{A_a^H T_a^{[AA]} A_a}_{H_{AA}} + \underbrace{A_a^H T_a^{[AB]} B_a + B_a^H T_a^{[BA]} A_a + B_a^H T_a^{[BB]} B_a}_{H_{AB+BA+BB}}$$

$$S = \underbrace{\sum_{a=1}^{N_A} A_a^H A_a}_{S_{AA}} + \underbrace{\sum_{a=1}^{N_A} B_a^H \dot{U}_a^H \dot{U}_a B_a}_{S_{BB}}$$

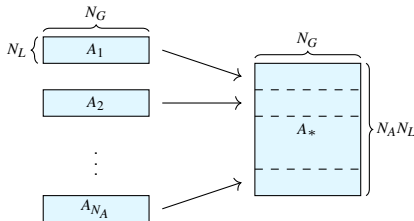
Constructing S_{AA}

An example of memory layout re-structuring

$$S_{AA} = \sum_{a=1}^{N_A} A_a^H A_a.$$

```
1: for  $a := 1 \rightarrow N_A$  do
2:    $S_{AA} = A_a^H A_a$ 
3: end for
```

▷ (zherk: $4N_L N_G^2$ Flops)



```
1:  $S_{AA} = A_*^H A_*$ 
```

▷ (zherk: $4N_A N_L N_G^2$ Flops)

Constructing $H_{AB+BA+BB}$

An example of algorithm re-structuring

$$\begin{aligned}
 H_{AB+BA+BB} &= \sum_{a=1}^{N_A} B_a^H (T_a^{[BA]} A_a) + (A_a^H T_a^{[AB]}) B_a + \\
 &\quad \frac{1}{2} B_a^H (T_a^{[BB]} B_a) + \frac{1}{2} (B_a^H T_a^{[BB]}) B_a \\
 &= \sum_{a=1}^{N_A} B_a^H (T_a^{[BA]} A_a + \frac{1}{2} T_a^{[BB]} B_a) + \\
 &\quad (A_a^H T_a^{[AB]} + \frac{1}{2} B_a^H T_a^{[BB]}) B_a
 \end{aligned}$$

-
- 1: **for** $a := 1 \rightarrow N_A$ **do**
 - 2: $Z_a = T_a^{[BA]} A_a$ ▷ (zgemm: $8N_L^2 N_G$ Flops)
 - 3: $Z_a = Z_a + \frac{1}{2} T_a^{[BB]} B_a$ ▷ (zhemm: $8N_L^2 N_G$ Flops)
 - 4: Stack Z_a to Z_\star and B_a to B_\star
 - 5: **end for**
 - 6: $H = Z_\star^H B_\star + B_\star^H Z_\star$ ▷ (zher2k: $8N_A N_L N_G^2$ Flops)

Stripped HSDLA algorithm for H and S

```

1: Create  $A, B$ 
2: //  $H_{AB+BA+BB}$ 
3: for  $a := 1 \rightarrow N_A$  do
4:    $Z_a = T_a^{[BA]} A_a$  ▷ (zgemm:  $8N_L^2 N_G$  Flops)
5:    $Z_a = Z_a + \frac{1}{2} T_a^{[BB]} B_a$  ▷ (zhemm:  $8N_L^2 N_G$  Flops)
6: end for
7:  $H = Z^H B + B^H Z$  ▷ (zher2k:  $8N_A N_L N_G^2$  Flops)
8: //  $S$ 
9:  $S = A^H A$  ▷ (zherk:  $4N_A N_L N_G^2$  Flops)
10:  $B = UB$  ▷ (scaling:  $2N_A N_L N_G$  Flops)
11:  $S = S + B^H B$  ▷ (zherk:  $4N_A N_L N_G^2$  Flops)
12: //  $H_{AA}$ 
13: for  $a := 1 \rightarrow N_A$  do
14:   try:
15:      $C_a = \text{Cholesky}(T_a^{[AA]})$  ▷ (zpotrf:  $\frac{4}{3} N_L^3$  Flops)
16:   success:
17:      $Y_a = C_a^H A_a$  ▷ (ztrmm:  $4N_L^2 N_G$  Flops)
18:   failure:
19:      $X_a = T_a^{[AA]} A_a$  ▷ (zhemm:  $8N_L^2 N_G$  Flops)
20: end for
21:  $H = H + A_{\text{HPD}}^H X_{\text{HPD}}$  ▷ (zgemm:  $8N_{A\text{HPD}} N_L N_G^2$  Flops)
22:  $H = H + Y_{\text{HPD}}^H Y_{\text{HPD}}$  ▷ (zherk:  $4N_{A\text{HPD}} N_L N_G^2$  Flops)

```

Previous multi-core results

NaCl ($K_{\max} = 4.0$)						
	IvyBridge				Haswell	
	HSDLA	FLEUR	×	HSDLA	FLEUR	×
1 core	31.53	48.31	1.53	19.00	47.41	2.50
2 cores	16.10	24.58	1.53	9.98	24.95	2.50
1 CPU	3.90	6.21	1.59	2.25	5.00	2.22
2 CPUs	2.61	5.20	1.99	1.93	4.03	2.09

TiO ₂ ($K_{\max} = 3.6$)						
	IvyBridge				Haswell	
	HSDLA	FLEUR	×	HSDLA	FLEUR	×
1 core	175.53	256.15	1.46	106.56	259.91	2.44
2 cores	86.68	127.90	1.48	53.48	131.21	2.45
1 CPU	19.63	29.35	1.50	10.63	25.95	2.44
2 CPUs	12.25	21.50	1.76	7.55	16.76	2.22

Table: Scalability of HSDLA and FLEUR: execution times in minutes on Haswell (12 cores / CPU) and IvyBridge (10 cores / CPU); speedups of HSDLA over FLEUR in **bold**.

Topic

The FLAPW method & FLEUR code

An exercise in performance portability

Experimental results

Conclusions

Porting to heterogeneous architectures

- Re-wrote the generation of H and S in terms of standardized libraries
- Reached a speedup of around $2\times$
- Can one exploit this useful exercise in code modernization beyond the speed obtained?

Porting to heterogeneous architectures

- Re-wrote the generation of H and S in terms of standardized libraries
- Reached a speedup of around $2\times$
- Can one exploit this useful exercise in code modernization beyond the speed obtained?

Kernels-based algorithms go a long way

- BLAS is the first numerical library ported to every new architecture
- On paper: quick and easy port to other architectures
- The natural questions are:
 - Can one port to CPU+GPU with minimal modifications?
 - How far can one get in terms of performance improvements?

Porting to heterogeneous architectures

Back-of-the-envelope analysis

- 5 lines of the algorithm constitute 97% of flops
- Correspond to BLAS-3 operations (`gemm`, `herk`, `her2k`)
- High arithmetic intensity and should fit GPUs well

Porting to heterogeneous architectures

Back-of-the-envelope analysis

- 5 lines of the algorithm constitute 97% of flops
- Correspond to BLAS-3 operations (`gemm`, `herk`, `her2k`)
- High arithmetic intensity and should fit GPUs well
- First step: offload these routine calls
- All 5 are BLAS kernels. Can we use some library?
 - cuBLAS
 - cuBLAS-XT
 - MAGMA
 - BLASX

Porting to heterogeneous architectures

Back-of-the-envelope analysis

- 5 lines of the algorithm constitute 97% of flops
- Correspond to BLAS-3 operations (`gemm`, `herk`, `her2k`)
- High arithmetic intensity and should fit GPUs well
- First step: offload these routine calls
- All 5 are BLAS kernels. Can we use some library?
 - cuBLAS
 - cuBLAS-XT
 - MAGMA
 - BLASX

Porting to heterogeneous architectures

Back-of-the-envelope analysis

- 5 lines of the algorithm constitute 97% of flops
- Correspond to BLAS-3 operations (`gemm`, `herk`, `her2k`)
- High arithmetic intensity and should fit GPUs well
- First step: offload these routine calls
- All 5 are BLAS kernels. Can we use some library?
 - cuBLAS
 - cuBLAS-XT
 - MAGMA
 - BLASX

Porting to heterogeneous architectures

Additional code?

3 x wrappers around the BLAS calls:

```
void gpu_zgemm_( char *transa, char *transb, int *m, int *n, int *k,
                std::complex<double> *alpha, std::complex<double> *A, int *lda,
                std::complex<double> *B, int *ldb,
                std::complex<double> *beta, std::complex<double> *C, int *ldc )
{
    cublasOperation_t cu_transa = transa[0] == 'N' ? CUBLAS_OP_N :
                                     transa[0] == 'T' ? CUBLAS_OP_T : CUBLAS_OP_C;
    cublasOperation_t cu_transb = transb[0] == 'N' ? CUBLAS_OP_N :
                                     transb[0] == 'T' ? CUBLAS_OP_T : CUBLAS_OP_C;
    cublasXtZgemm( handle, cu_transa, cu_transb, *m, *n, *k,
                  (cuDoubleComplex *)alpha, (cuDoubleComplex *)A, *lda,
                  (cuDoubleComplex *)B, *ldb,
                  (cuDoubleComplex *)beta, (cuDoubleComplex *)C, *ldc );
}
```

Porting to heterogeneous architectures

Additional code?

- 3x wrappers (zgemm, zherk, zher2k)
- Init and cleanup of cuda runtime and devices
 - Get #devices, initialize devices, create handlers, ...
 - Destroy handlers, free devices, ...
- Allocate data in page-locked memory
 - Avoid “hidden” copies
 - Fast data transfer

Porting to heterogeneous architectures

Additional code?

- 3x wrappers (zgemm, zherk, zher2k)
- Init and cleanup of cuda runtime and devices
 - Get #devices, initialize devices, create handlers, ...
 - Destroy handlers, free devices, ...
- Allocate data in page-locked memory
 - Avoid “hidden” copies
 - Fast data transfer

Only around 100 lines of additional code

Topic

The FLAPW method & FLEUR code

An exercise in performance portability

Experimental results

Conclusions

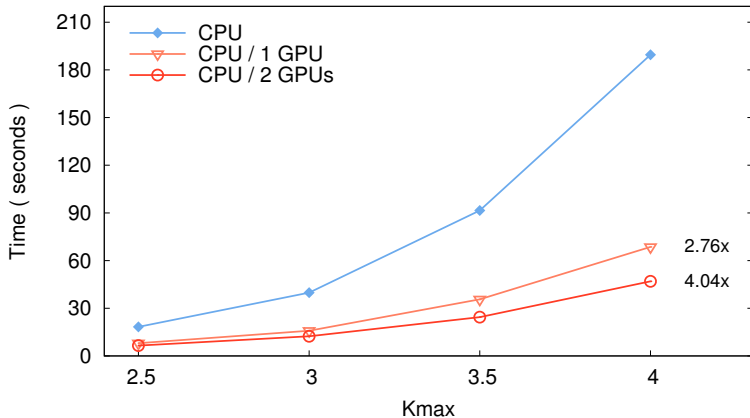
Experimental results

Sandy Bridge:

- CPU: E5-2650, 2 x 8core, 2.0GHz, 64GBs RAM
- 2 Nvidia Tesla K20X
- Peak performance: 256 GFs/s + 2 x 1.3 TFs/s

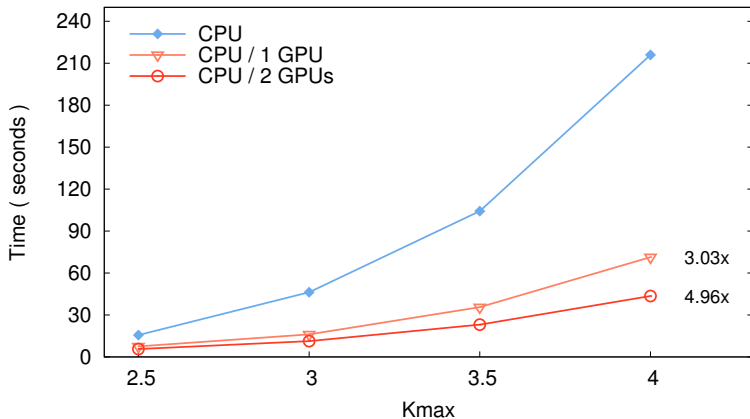
Experimental results

Test case 1: NaCl ($N_A = 512, N_L = 49, N_G = [2256 - 9273]$)



Experimental results

Test case 2: AuAg ($N_A = 108, N_L = 121, N_G = [3275 - 13379]$)



Topic

The FLAPW method & FLEUR code

An exercise in performance portability

Experimental results

Conclusions

Conclusions and Future Work

Conclusions

- **Modernizing** algorithm structure of legacy code is critical
- Layered design built on top of standardized libraries
- Increase in performance
- (Almost) free lunch \Rightarrow **performance portability**
- Case of FLEUR: up to $12 \times$ speedup

Conclusions and Future Work

Conclusions

- **Modernizing** algorithm structure of legacy code is critical
- Layered design built on top of standardized libraries
- Increase in performance
- (Almost) free lunch \Rightarrow **performance portability**
- Case of FLEUR: up to $12 \times$ speedup

Future Work:

- Hybrid for `zgemm`, `zherk`, `zher2k`
- Experiments on Jureca (4 GPU devices)
- **Apply the same methodology** to other Materials Science codes

Thank you for your attention!

Details on the original HSDLA:

- “High-performance generation of the Hamiltonian and Overlap matrices in FLAPW methods.” *Edoardo Di Napoli, Elmar Peise, Markus Hrywniak and Paolo Bientinesi*. Accepted for publication in Comp. Phys. Comm. [arXiv:1602.06589]
- “Hybrid CPU-GPU generation of the Hamiltonian and Overlap matrices in FLAPW methods.” *Diego Fabregat-Traver, Davor Davidović, Markus Höhnertbach, Edoardo Di Napoli*. Accepted for publication in LNCS. [arXiv:1611.00606]