BERTHA and PyBERTHA: state of the art for full four-component Dirac-Kohn-Sham calculations

Loriano Storchi

University of Chieti-Pescara
Istituto di Scienze e Tecnologie Chimiche del CNR, Perugia
INFN (Istituto Nazionale Di Fisica Nucleare) sez. Perugia

Table of contents

Table of contents

Active molecular codes based on Dirac eq.

DIRAC
Saue (FR)
Visscher (NL),
Jensen (DK) et al.

Respect
Repisky
Oslo, Bratislava

Wenjan Liu (CN)

CronusQ
Xiaosong Li
Seattle

PySCF
Q. Sun et al.
(USA)

BERTHA
Quiney (Australia)
Belpassi (IT)
Storchi (IT)

Introduction

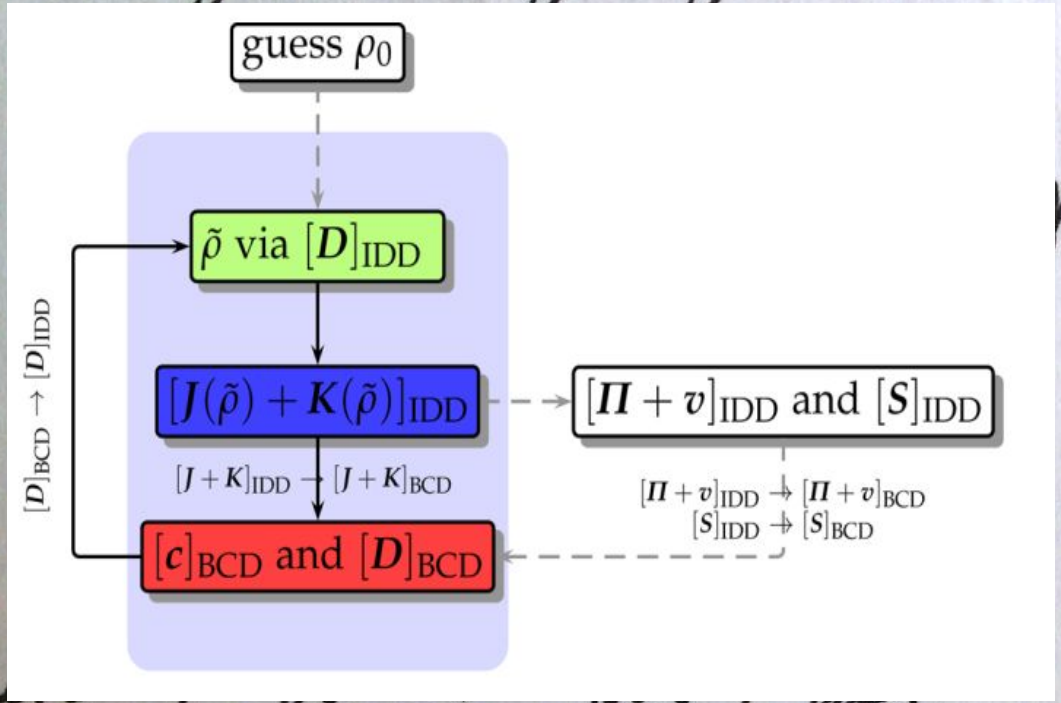
The eigenvalue equation reads:

$$\boldsymbol{H}_{\mathrm{DKS}}\begin{bmatrix}\boldsymbol{c}^{(\mathrm{L})}\\ \boldsymbol{c}^{(\mathrm{S})}\end{bmatrix} = E\begin{bmatrix}\boldsymbol{S}^{(\mathrm{LL})} & 0\\ 0 & \boldsymbol{S}^{(\mathrm{SS})}\end{bmatrix}\begin{bmatrix}\boldsymbol{c}^{(\mathrm{L})}\\ \boldsymbol{c}^{(\mathrm{S})}\end{bmatrix} = 0$$

Spinors expansion vectors

The matrix $\mathbf{H}_{\mathrm{DKS}}$ depends, because of **J** and **K**, on the canonical spinor-orbitals produced by its diagonalization, so that the solution **c** must be obtained recursively to self-consistence.

Parallelization strategies



The matrix H_{DKS} depends, because of J and K, on the canonical spinor-orbitals produced by its diagonalization, so that the solution c must be obtained recursively to self-consistence.

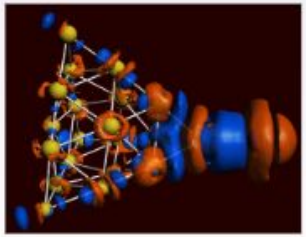
Introduction

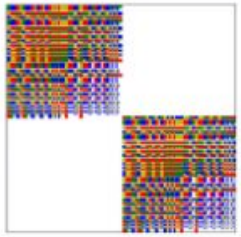
It is universally recognized that relativistic effects play a crucial role in chemistry, especially for heavy elements

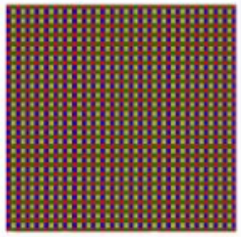
- The challenge clearly arises from the fact that heavy elements have a very large number of electrons, and both relativistic effects and electron correlation play a crucial role
- DKS matrices are big because of Large and Small components
- DKS matrices are inherently complex matrices

Introduction

Small molecules containing one heavy atom ($CsAu$, HgF_2)







OpenMP

2003 — Linux Beowulf Cluster

2009 — SGI (GE) cc-NUMA cache-coherent Non Uniform Memory Access

2014 — IBM BG/Q Cineca (IT)

2019 — Cray x30 EPCC (UK)

2024 >>> CPU & GPU

Introduction

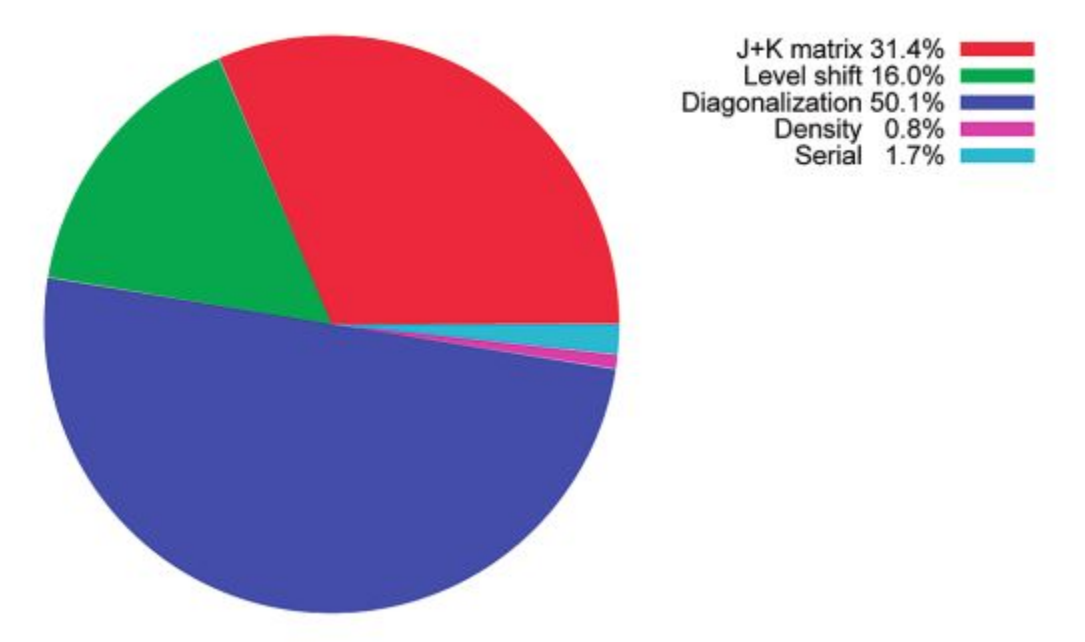
We employed density fitting techniques; what is the efficiency of the density fitting approach?

Cluster	DKS Size	$(J+K)_{conv}$	$(J+K)_{fit}$	Speed-up
Au_2	1560	$1.86 \cdot 10^3$	7.4	251
Au_4	3120	$1.71 \cdot 10^4$	44.1	388
Au_8	6240	$1.71 \cdot 10^5$	296	578
Au_{16}	12480	$1.91 \cdot 10^{6a}$	$2.16 \cdot 10^3$	884

[a] Extrapolated value.

Table of contents

Parallelization strategies

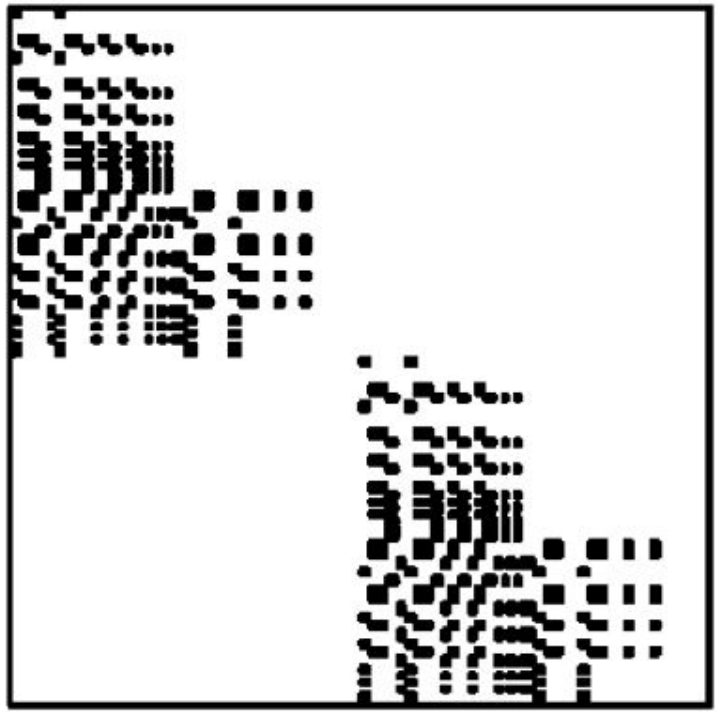


CPU time percentages for the various phases of a serial DKS calculation of the gold cluster Au_{16} . All linear algebra operations are performed with the Intel Math Kernel Library. We are here considering each SCF step

Parallelization strategies

- According to Amdahl's law, serial portion of code limits the speedup, thus **we tried to remove any single portion of serial code**
- During the SCF procedure, in fact, **the "bulk" memory allocation is due to several 2N × 2N complex Hermitian matrices**, we want to share the memory burden
- We will use ScaLAPACK for linear algebra
- The parallelization strategy for the J+K construction, and similarly for all the others matrices is induced by the problem (i.e., by the matrices structure)
- The strategy adopted for all the other matrices is basically tha same as for the J+K

Parallelization strategies



Integral Driven Distribution (IDD): Cyclically assigning to each process the allocation and computation of blocks whose offsets and dimensions depend on the specific structure of the G-spinor matrices.

```
DO block_num = 1, max_nom_of_block
  IF ((my_rank+((num_of_processors)*my_block_num)) == block_num)
    my_block_num = my_block_num + 1
    ALLOCATE block
    COMPUTE block
  END IF
ENDDO
```

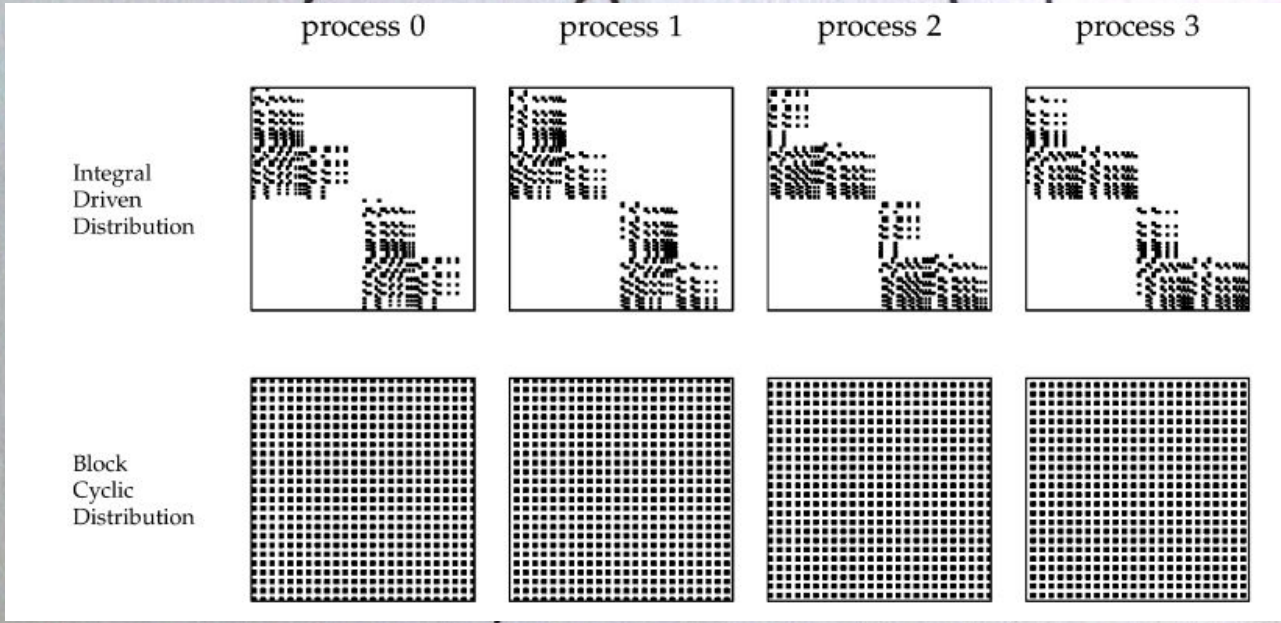
$$[J+K]_{\mathrm{IDD}} \rightarrow [J+K]_{\mathrm{BCD}}$$

Linear algebra using SCALAPACK

(Level Shift, Diagonalization, Density)

Parallelization strategies

$$[J+K]_{\mathrm{IDD}} \rightarrow [J+K]_{\mathrm{BCD}}$$



Parallelization strategies

Set up an “info” array

⇩

Allocates send buffer

⇩

Pack the owned data (along with the related destination
Indices, so local indices) into the send-buffers

⇩

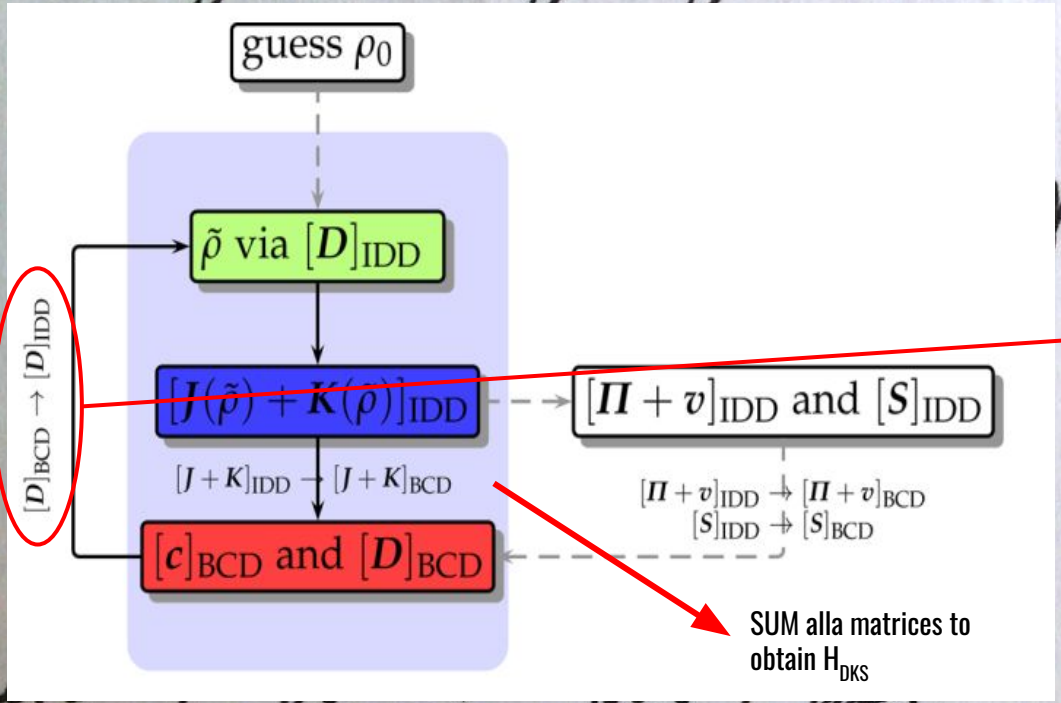
Communicate

⇩

Deallocate send buffer

```
loop on processes
    if my turn then
        make my 'info' array available to all and send packed data
    else
        allocate receive-buffers according to the 'info' array
        made available by the sender, receive and unpack data,
        deallocate receive-buffers
    end if
end loop
```

Parallelization strategies



A similar strategy has been adopted for the one electron and superposition matrices

In the IDD scheme, instead , the distribution is much less regular. A convenient and efficient representation is obtained using a derived data type, composed of a two-dimensional array and some metadata describing its size and placement in the global matrix. On each process, an array of such derived data types is then used to identify each local IDD block

Parallelization strategies

Wall-Clock Time in Seconds (Average over 4 SCF Cycles) Spent in the Distribution Mapping Routines during Calculation

	$[(Ph_3P)Au(C_2H_2)]^+$		Au_8		Au_{16}		Au_{32}	
P	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$
4	0.70	0.57	0.74	0.60	3.52	2.46	20.61	9.68
8	0.39	0.36	0.41	0.38	1.85	1.46	9.09	6.22
16	0.21	0.25	0.22	0.24	0.96	0.98	5.96	3.72
32	0.20	0.24	0.21	0.25	0.73	0.87	2.95	3.32
64	0.35	0.38	0.36	0.40	0.82	1.00	2.55	3.27
128	(2.5%) 0.92	(2.3%) 0.86	(2.9%) 0.90	(2.8%) 0.88	1.50	1.64	3.23	4.02
256	(6.9%) 2.69	(6.6%) 2.56	(6.5%) 1.84	(7.4%) 2.11	(2.6%) 3.84	(2.4%) 3.64	6.10	6.69

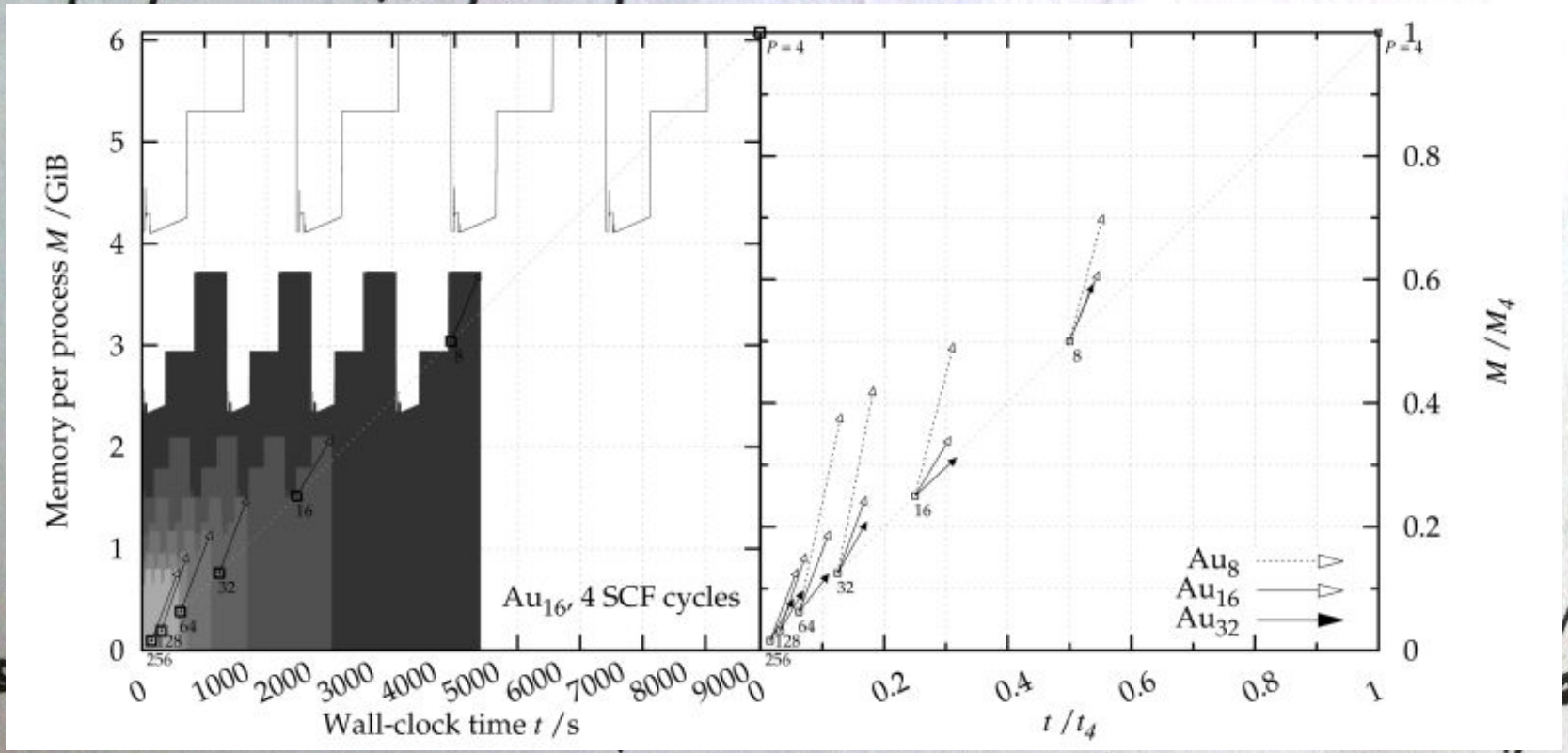
[a]Percentages of the SCF iteration times are smaller than 1% in any case except where otherwise noted in parentheses.

Parallelization strategies

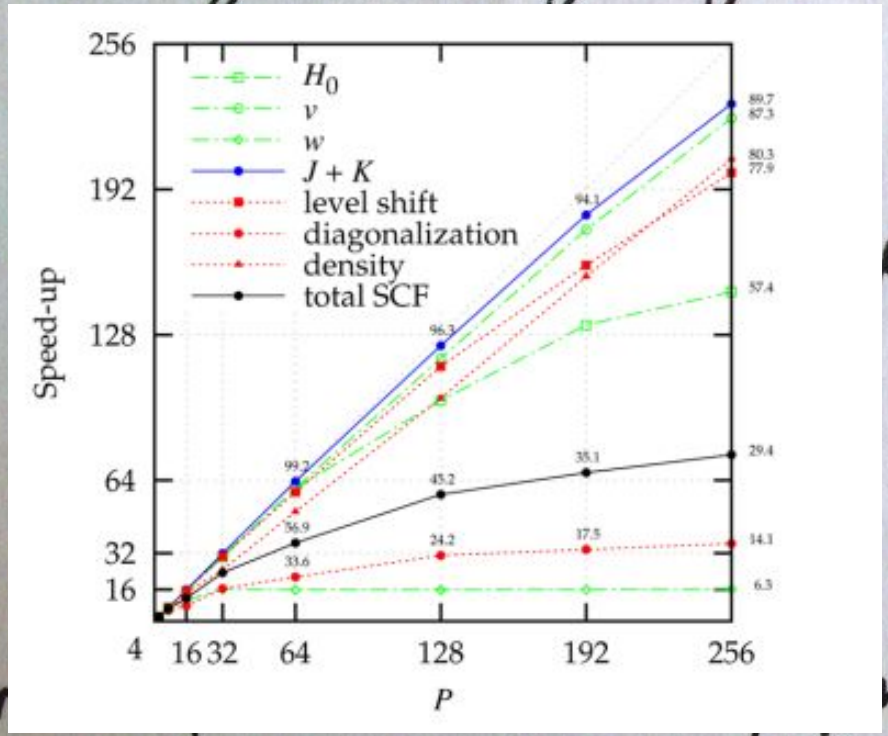
Memory Per Process Peak (Average Value M_{av}, Maximum Positive Δ_+ and Negative Δ_- Deviations) in MiB over P Processes

	$[(Ph_3P)Au(C_2H_2)]^+$			Au_8			Au_{16}			Au_{32}		
P	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+
4	76	1842	37	34	1882	60	81	6214	46	126	23835	187
8	57	1253	59	64	1303	71	54	3770	90	88	14106	232
16	12	884	15	8	888	44	14	2124	23	8	7405	63
32	62	799	84	33	773	96	44	1499	85	53	4827	155
64	47	700	92	37	672	92	80	1167	80	64	2880	83
128	21	631	115	24	620	110	54	961	81	48	2220	101
256	10	599	134	15	586	119	52	866	83	80	1942	92

Parallelization strategies



Parallelization strategies



Speedup for all of the computational kernels of the SCF procedure for Au_{32} as a function of the number of processors P

Mccw cluster equipped with Intel(R) Xeon(R) CPU E5-26700 2.60 GHz (24 nodes, 384 cores with 128 GiB/node, 8 GiB/core) and Infiniband network

Table of contents

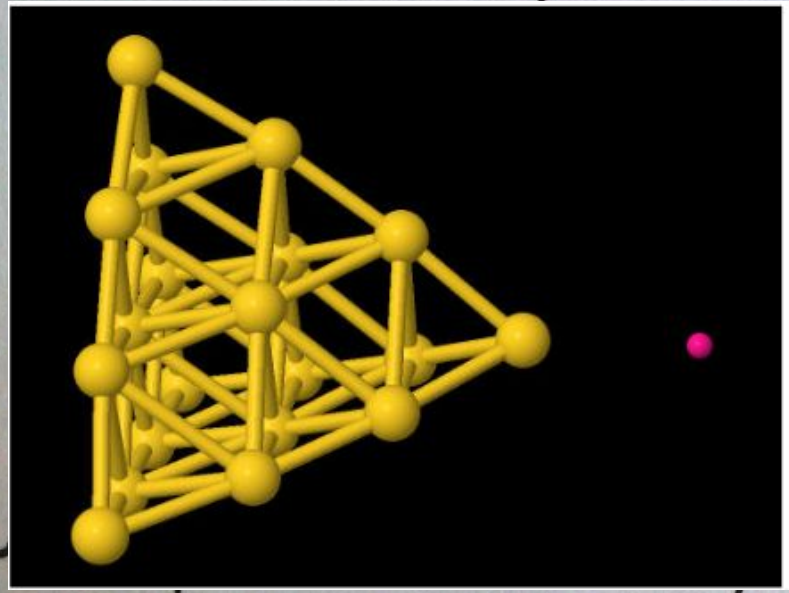
A test application using NOCV/CD

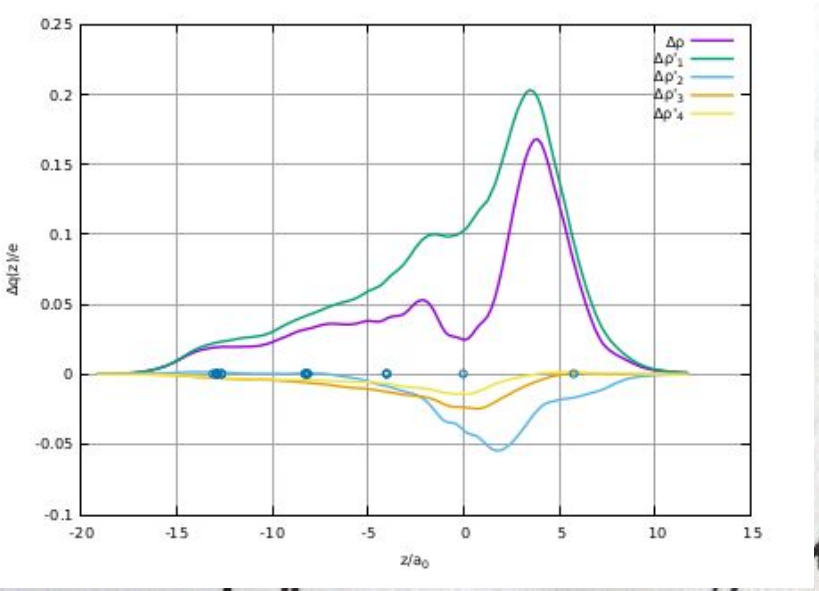
- CD: Charge-Displacement analysis has been successfully employed to describe the nature of intermolecular interactions and various type of controversial chemical bond
- Charge-Displacement function defined as a partial integration along a suitable z axis of the difference $\Delta\rho(x, y, z')$ between the electron density of the adduct and that of its non-interacting fragments placed at the same equilibrium position they occupy in the adduct.
- The core idea of the approach is the decomposition, via natural orbitals for chemical valence (NOCV), of the so-called charge-displacement (CD) function into additive Chemically meaningful components.

$$\Delta q(z) = \int_{-\infty}^{z} dz' \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \Delta\rho(x, y, z')dxdy$$

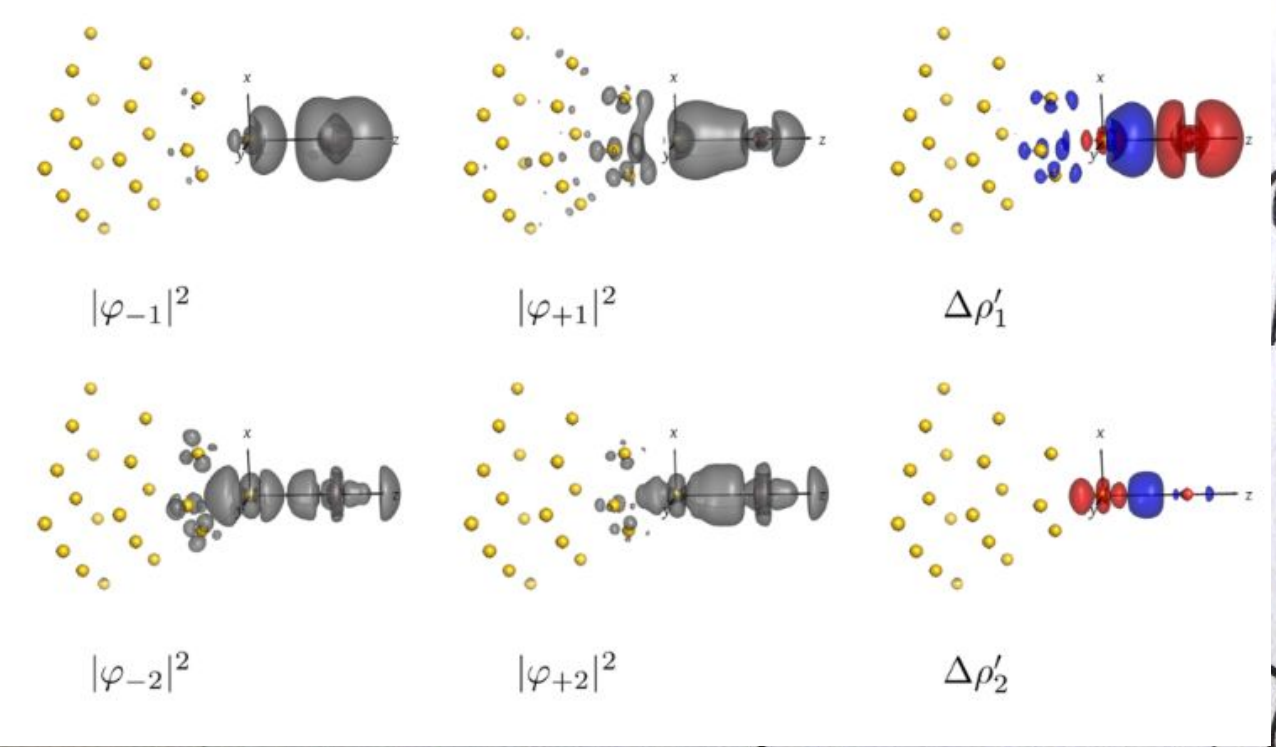
A test application using NOCV/CD

The Au_{20} - Fl complex and the CD analysis for the bond





A test application using NOCV/CD



We are able to split the total CD curve into several chemically meaningful additive components.

isodensity surfaces

red surfaces identify charge depletion areas and blue surfaces identify charge accumulation

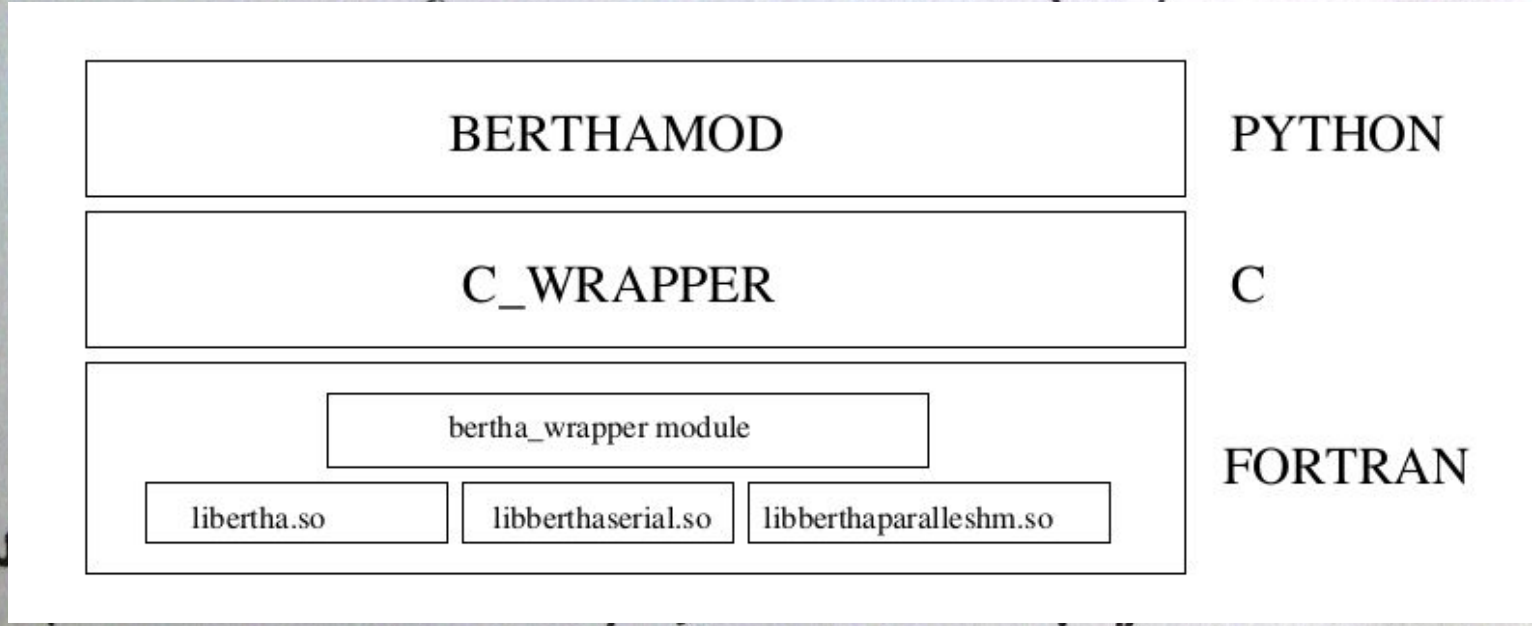
Table of contents

PyBERTHA a Python binding for BERTHA

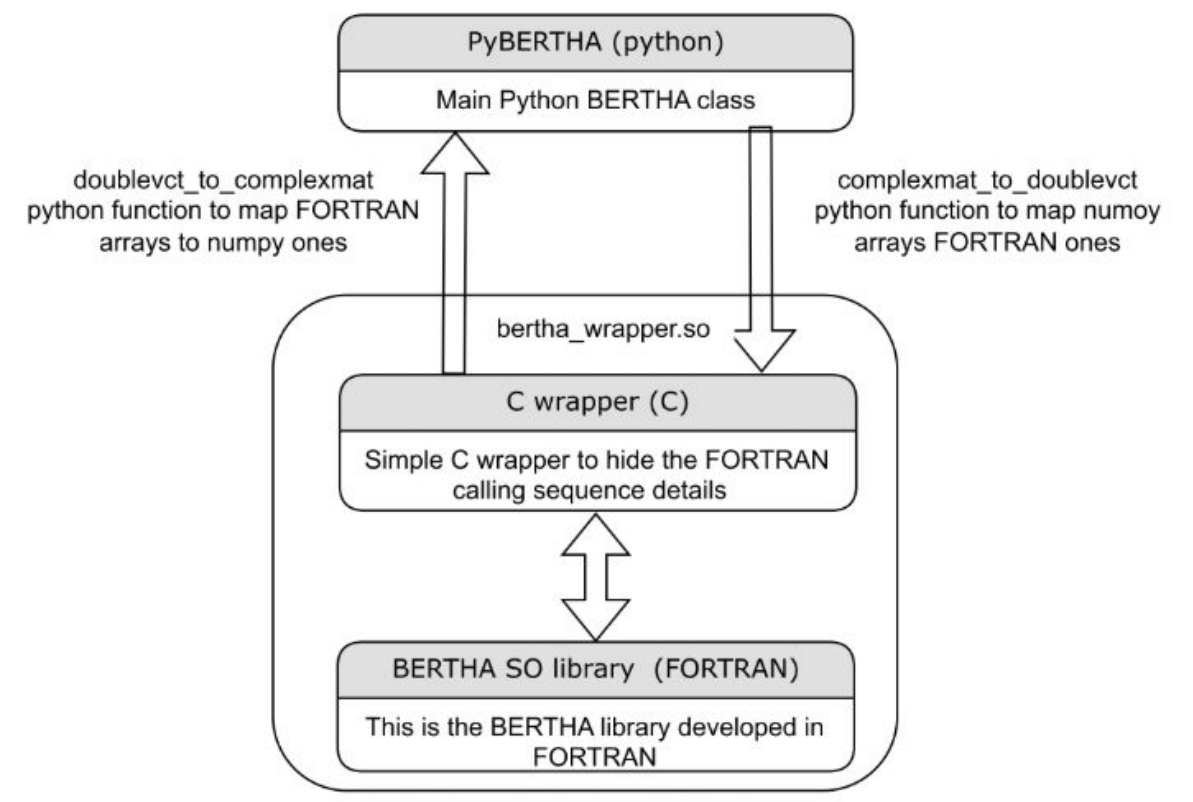
- Undoubtedly the Python programming language is emerging as one of the most important and used HLL also in the field of scientific computing.
- Python HLL, besides providing an extensive range of modules to be used to solve comprehensive set of computational problems, enables for a quick prototyping
- So Python is clearly a natural choice for the BERTHA project.

PyBERTHA a Python binding for BERTHA

An overview of the software and HLL layers.



PyBERTHA a Python binding for BERTHA



A couple of Python functions , and some specific lines of code at the bertha_wrapper/Fortran layer, are need to implement the data movement between Python to/from Fortran

PyBERTHA a Python binding for BERTHA

Algorithm 2 A simple four-component relativistic DFT program implemented using the *berthamod* Python module

```
1: import berthamod
2: Inputs: input options ...
3: bertha = berthamod.pybertha(wrapperso)
4: bertha.set_verbosity(verbosity)
5: bertha.set_fnameinput(inputfilename)
6: bertha.set_fittfname(fittfilename)
7: bertha.set_tresh(tresh)
8: bertha.init()
9: ovapmtx, eigenvectors, fockmtx, eigenvalues = bertha.run()
10: etotal = bertha.get_etotal()
11: bertha.finalize()
12: Output: Total Energy and MO energies ...
```

PyBERTHA a Python binding for BERTHA

Impact of the Python binding in the total execution time using 10 SCF iterations. The code has been executed on a Intel(R) Xeon(R) CPU E3- 1220 compiling the code with the Intel(R) compiler version: 2018.3.222

System	Matrix Dimension	Wall-time 10 SCF iterations with Python (s)	Wall-time 10 SCF iterations without Python (s)	Python overhead 10 SCF iterations
H_2O	140	3.910	3.906	0.09 %
Zn	624	20.471	20.455	0.07 %
Cd	916	41.595	41.556	0.09 %
Hg	1240	97.046	96.975	0.07 %
Au_2	1560	104.458	104.354	0.99 %
Au_4	3152	613.912	613.483	0.07 %
Au_8	6304	3965.911	3964.078	0.05 %

PyBERTHA a Python binding for BERTHA

Impact of the Python binding in the berthamod.get_realtime_fock method.

System	Matrix Dimension	Wall-time 10 SCF iterations with Python (s)	Wall-time 10 SCF iterations without Python (s)	Python overhead 10 SCF iterations
H_2O	140	0.383	0.382	0.19 %
Zn	624	1.257	1.250	0.52 %
Cd	916	2.592	2.575	0.64 %
Hg	1240	6.388	6.358	0.48 %
Au_2	1560	6.395	6.294	1.59 %
Au_4	3152	38.050	37.479	1.50 %
Au_8	6304	244.447	241.999	1.00 %

Table of contents

PyBERTHA a Python binding for BERTHA

- Parallel PyBERTHA
  - First option load the liberrthaparallelshm.so and after we can manage MPI at python level using, for instance, mpi4py or similar modules
  - Other option is to use OpenMP, especially when one is interested in improving the performances for small molecular systems
    - we adopted this strategies and the parallelization strategy adopted for the J+k and other matrices construction is the same as the MPI version, but within shared memory model

PyBERTHA a Python binding for BERTHA

Small Systems OpenMP (multithreading) better or equivalent to MPI

System	Step	Serial	OpenMP				MPI			
			4	9	16	24	2X2	3X3	4X4	6X4
Zn	**J+K** matrix	0.67	0.17	0.12	0.09	0.09	0.28	0.26	0.24	0.26
	Linear algebra	0.21	0.15	0.14	0.15	0.13	0.15	0.16	0.14	0.18
	Total iteration	2.33	0.60	0.44	0.38	0.36	0.52	0.50	0.45	0.50
Hg	**J+K** matrix	2.29	0.65	0.41	0.31	0.33	0.88	0.64	0.50	0.53
	Linear algebra	1.61	0.81	0.71	0.67	0.75	0.76	0.66	0.59	0.70
	Total iteration	7.88	2.66	1.89	1.58	1.67	2.00	1.58	1.32	1.41
Au_2	**J+K** matrix	3.53	1.08	0.61	0.46	0.39	1.15	0.85	0.71	0.69
	Linear algebra	3.36	1.63	1.31	1.12	0.84	1.45	1.22	1.03	1.22
	Total iteration	9.83	3.81	2.74	2.32	1.95	2.97	2.45	2.12	2.23

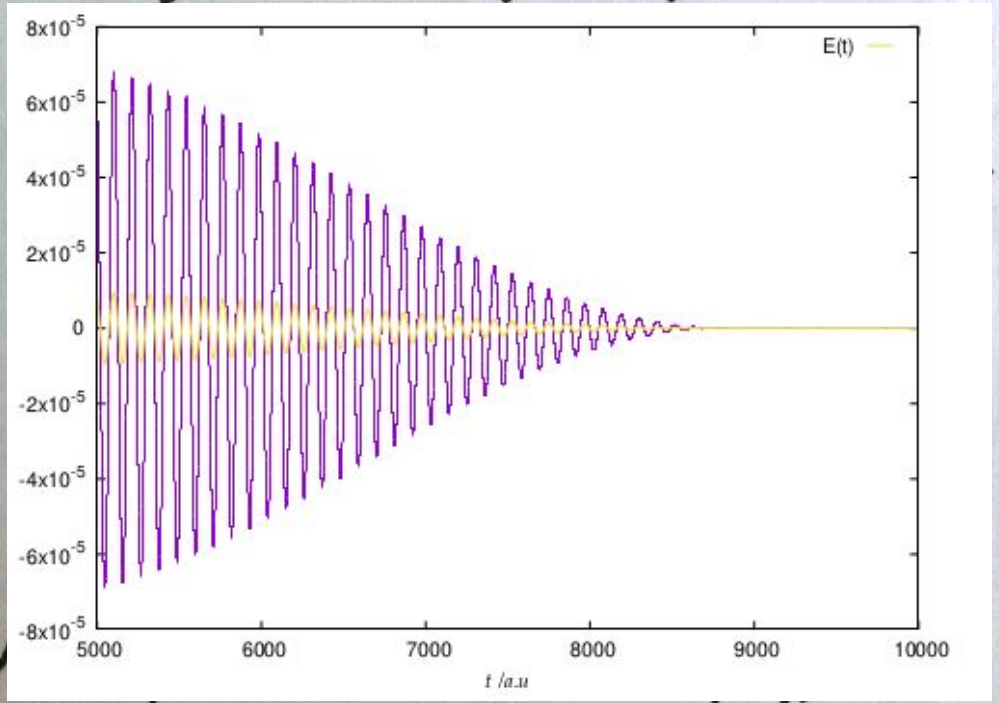
PyBERTHA a Python binding for BERTHA

big Systems MPI better then OpenMP

System	Step	Serial	OpenMP				MPI			
			4	9	16	24	2X2	3X3	4X4	6X4
Au_4	**J+K** matrix	23.05	6.16	3.33	2.19	1.83	6.10	3.43	2.35	2.00
	Linear algebra	31.14	10.70	8.55	7.30	7.89	9.69	7.21	5.59	6.85
	Total iteration	64.60	20.96	14.87	12.12	12.32	16.48	11.55	9.08	9.75
Au_8	**J+K** matrix	158.93	42.17	21.67	13.26	10.45	42.56	20.94	12.30	9.30
	Linear algebra	265.60	90.66	66.05	60.95	65.22	39.57	54.55	37.23	42.51
	Total iteration	469.76	149.84	99.96	84.74	86.08	86.55	75.83	52.05	53.78
Au_{16}	**J+K** matrix	1185.56	314.07	153.99	91.73	70.50	308.19	149.13	85.90	65.78
	Linear algebra	2303.85	679.51	498.07	424.99	414.06	540.20	388.93	270.34	327.72
	Total iteration	3687.16	1069.80	703.74	561.42	528.04	797.21	516.11	352.81	390.62

Table of contents

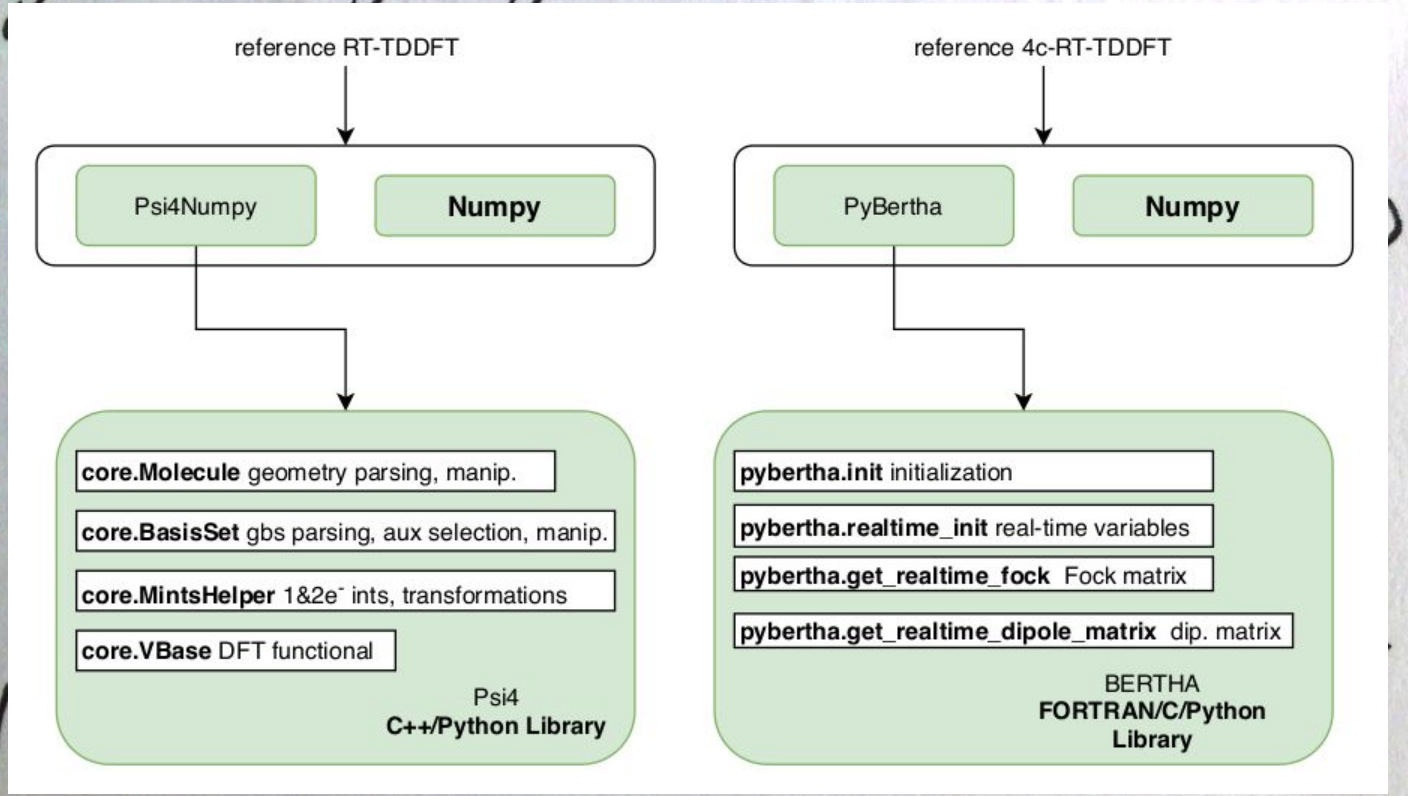
PyBERTHART a real-time TDDFT implementation



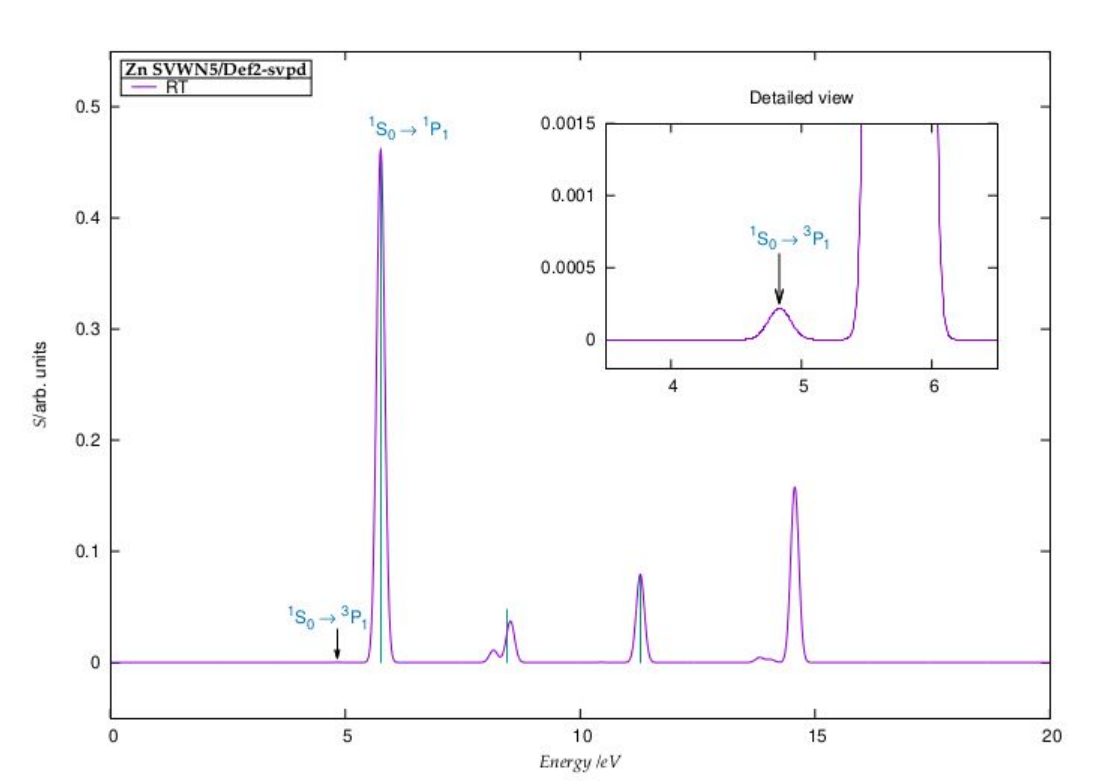
We implemented a real-time time-dependent four-component Dirac-Kohn-Sham (RT-TDDKS) implementation based on the BERTHA code.

Induced dipole moment in H_2 molecule. The representation of the external field is also reported as a yellow line.

PyBERTHART a real-time TDDFT implementation



PyBERTHART a real-time TDDFT implementation



The absorption spectrum of group 12 atoms (Zn,Cd,Hg) features spin-forbidden transitions.

A proper relativistic framework is needed in order to reproduce forbidden transitions

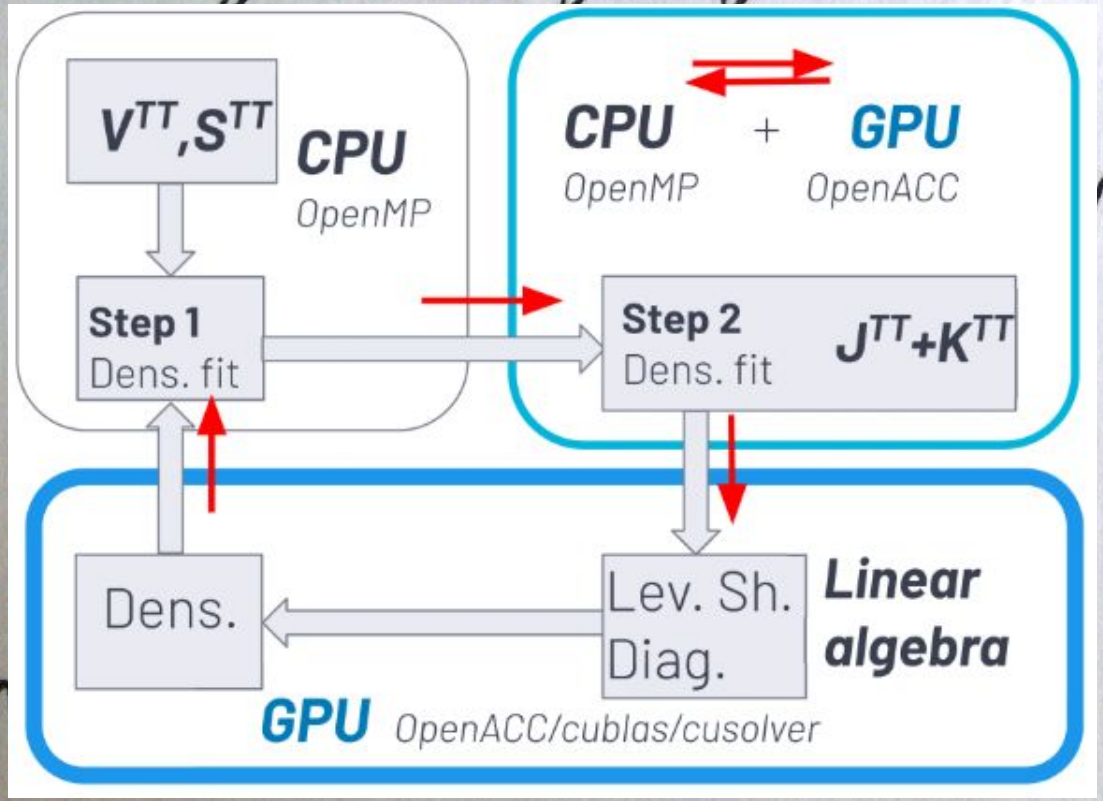
We are here reporting the Zn spectrum

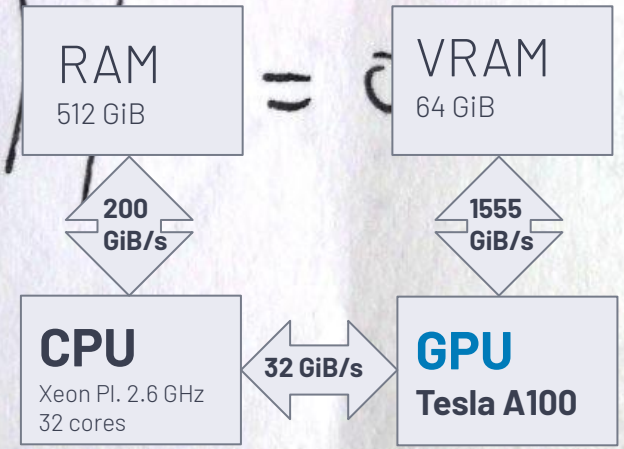
Table of contents

GPU porting of the code

OPEN HACKATHONS

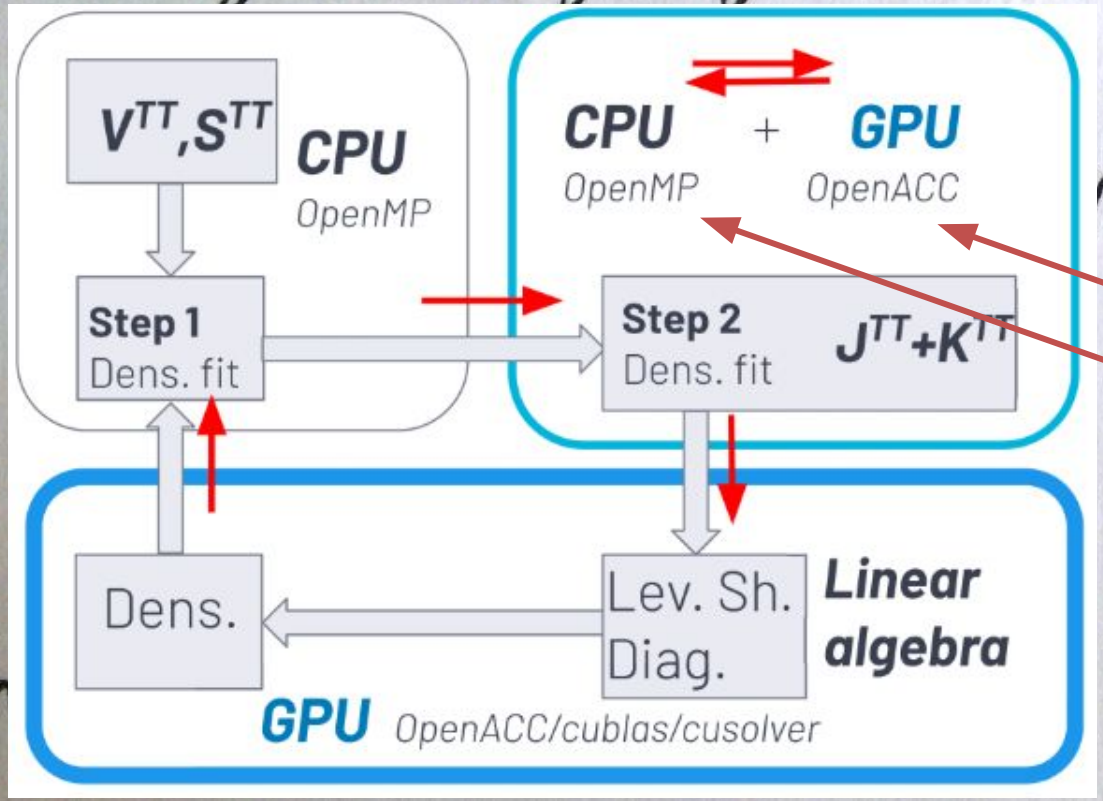
CINECA





Nvfortran -O3 -acc=gpu
Cublas,cusolver for GPU
Openblas for CPU

GPU porting of the code



Step 1: create and solve two linear systems: Ad = v and Az = w

Step2: evaluate

$$\tilde{J}_{\mu\nu}^{(TT)} + \tilde{K}_{\mu\nu}^{(TT)} =$$

$$= \sum_{a=1}^{N_{aux}} (d_a + z_a) \sum_{i,j,k} E_0^{(TT)}[\mu, \nu; i, j, k] \langle f_a \| \overline{\alpha}_{\mu\nu}; i, j, k \rangle$$

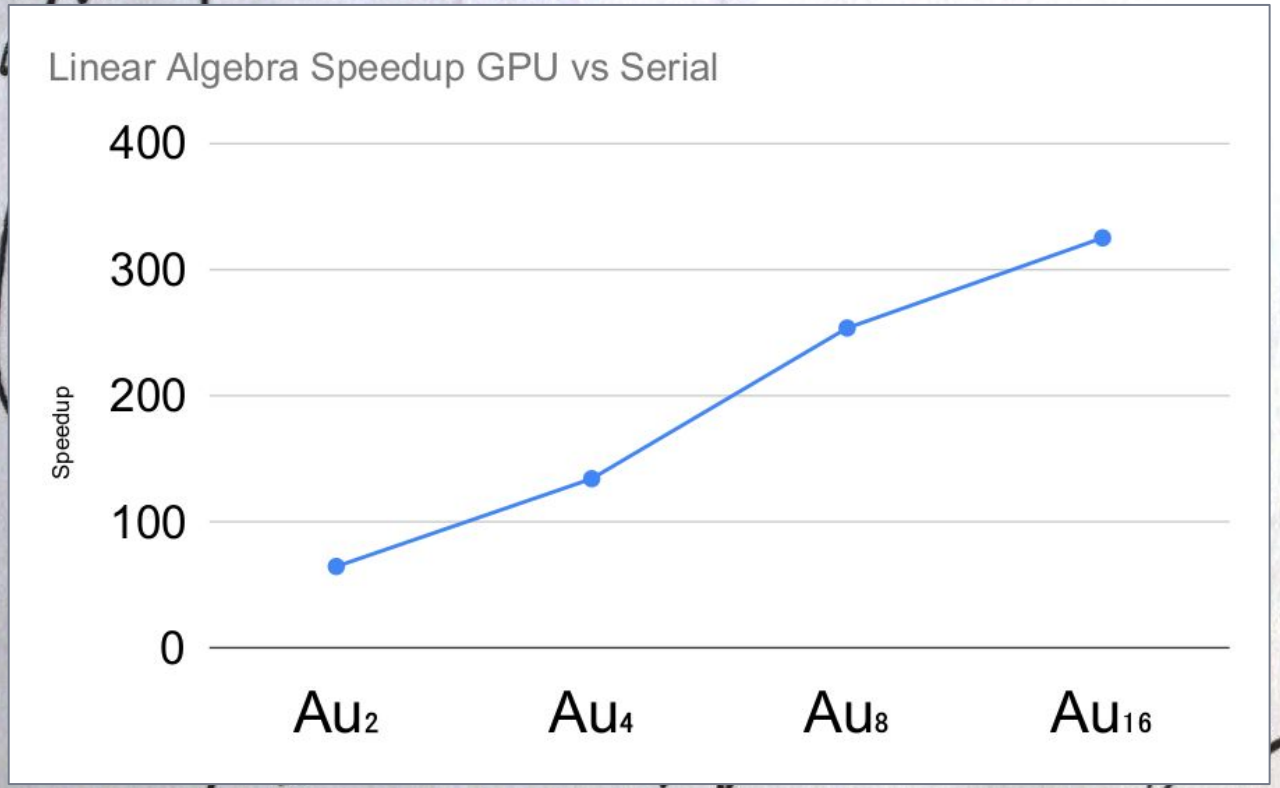
GPU porting of the code

Linear algebra is about 70% of the total cost in a serial run.

Au:Dyall VDZ
Openblas for CPU (serial)
Cublas for GPU
nvfortran compiler

Au_{16} (dim 12608):

Serial CPU 4761.1 sec
GPU 14.7 sec



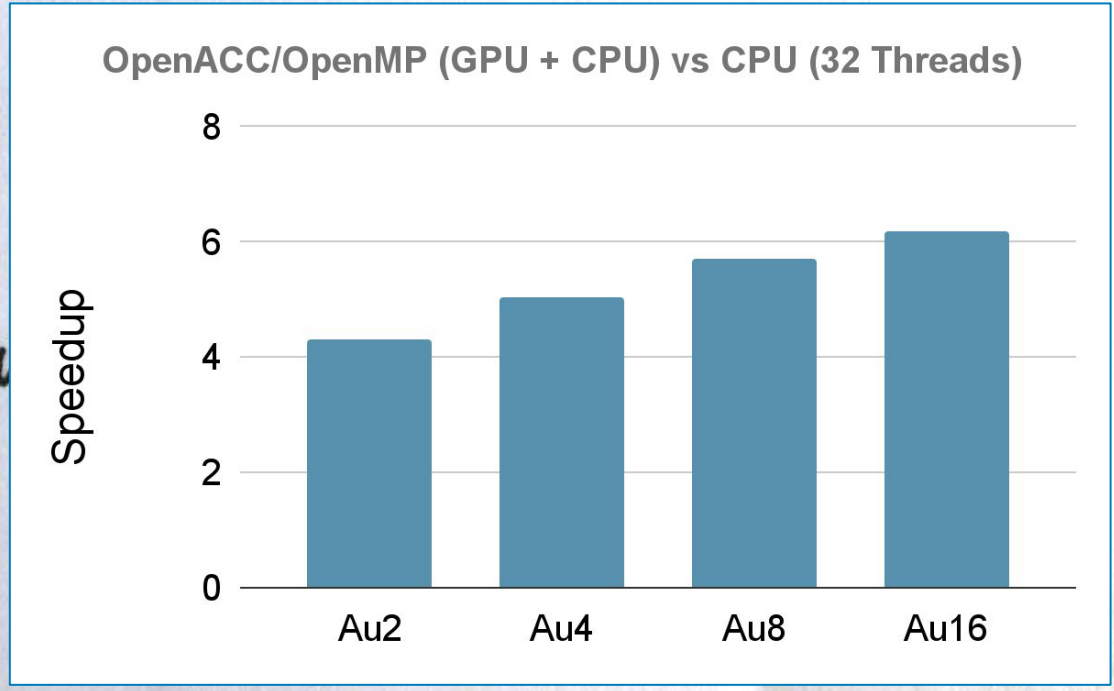
GPU porting of the code

CPU + **GPU**

Openmp *OpenACC*

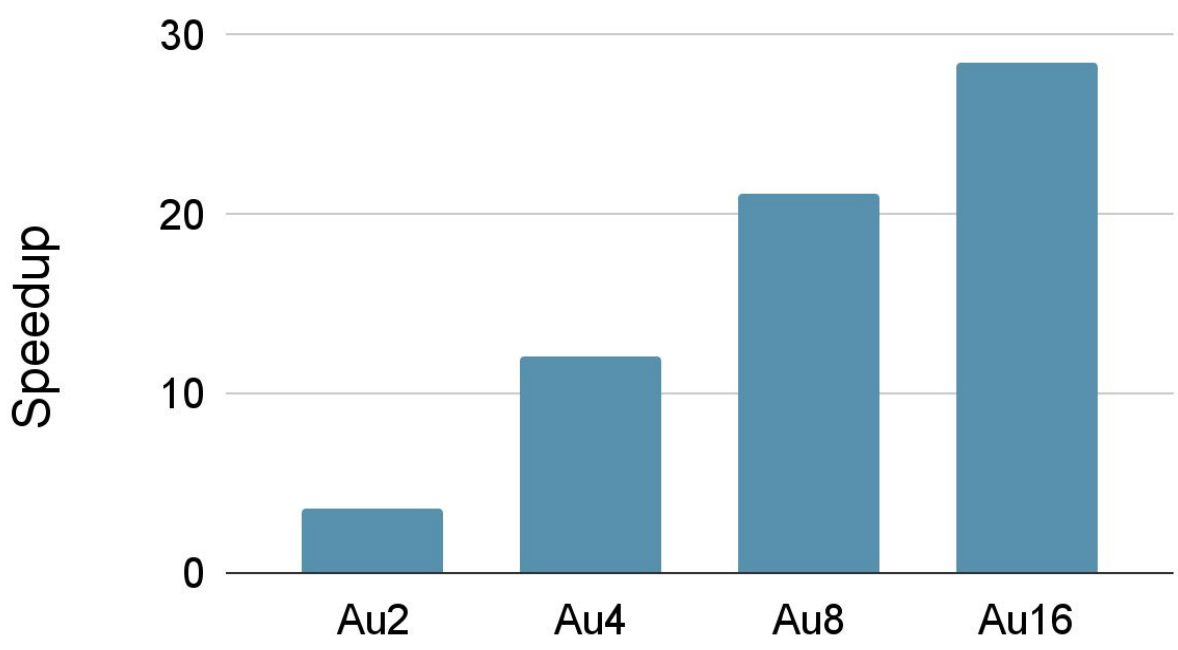
Step 2
Dens. fit $J^{TT}+K^{TT}$

This step is about 20% of the total cost in a serial run.



Au_{16} (dim 12608): Serial CPU :1180 sec
CPU (32 threads) : 62 sec
GPU + CPU (32 threads) : 10 sec

GPU porting of the code



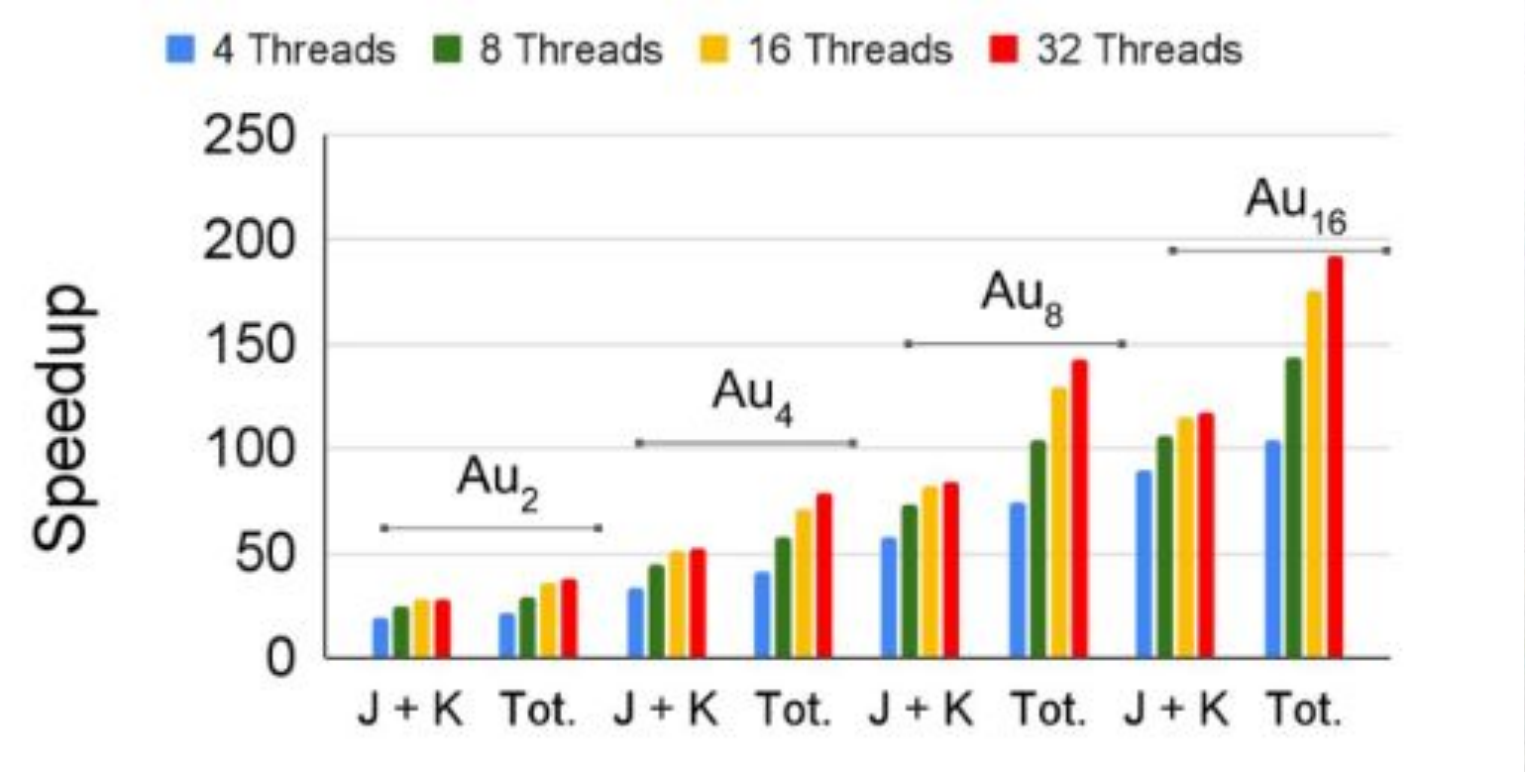
Au_{16} (dim 12608):

Serial CPU :6045 sec
CPU (#32) : 893 sec
GPU+CPU(#32): 31 sec

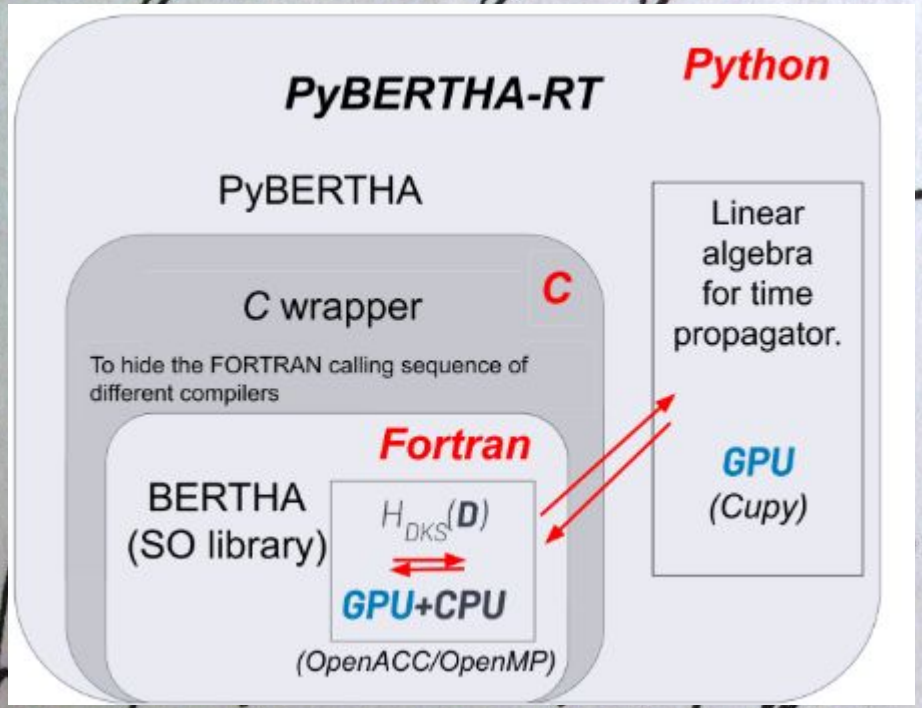
Numerical stability

#1 :-38089.88730963261
#32 :-38089.88730963165
GPU : -38089.88730963576

GPU porting of the code

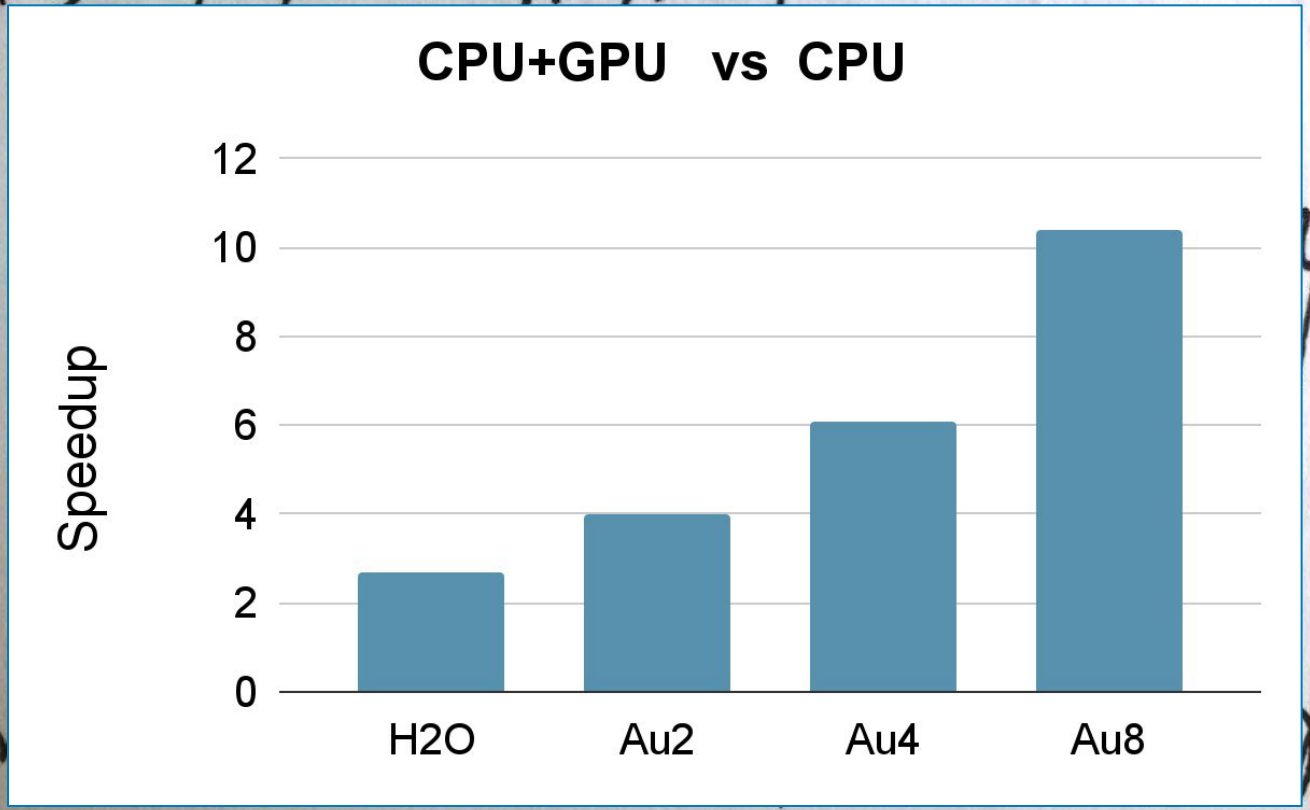


GPU porting of the code



- BERTHA SOs are using OpenACC and OpenMP, thay can be “almost easily” called by th Python Layer
- C_WRAPPER particularly useful to hide various Fortran compilers differences
- NUMPY to CuPY quite easy porting of the RT-TDDKS
  - Almost all the python module is GPU only, clearly tharere are several data movement involved when calling BERTHA SOs

GPU porting of the code



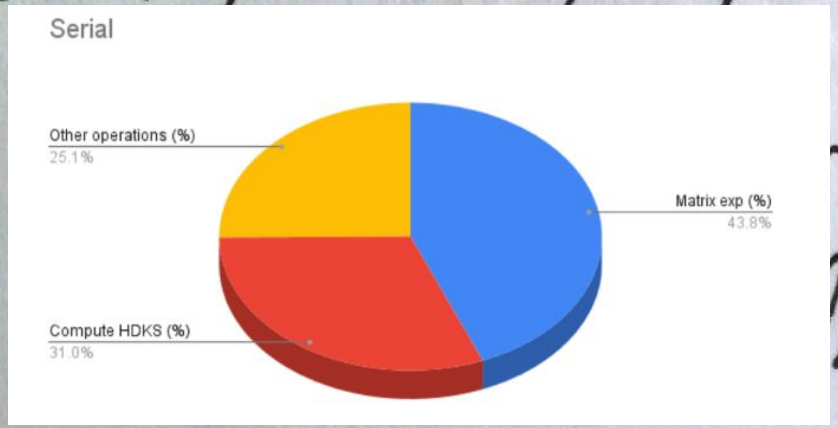
Au_8 (dim 6304):

Openmp/Numpy
CPU (#32) : 835 sec

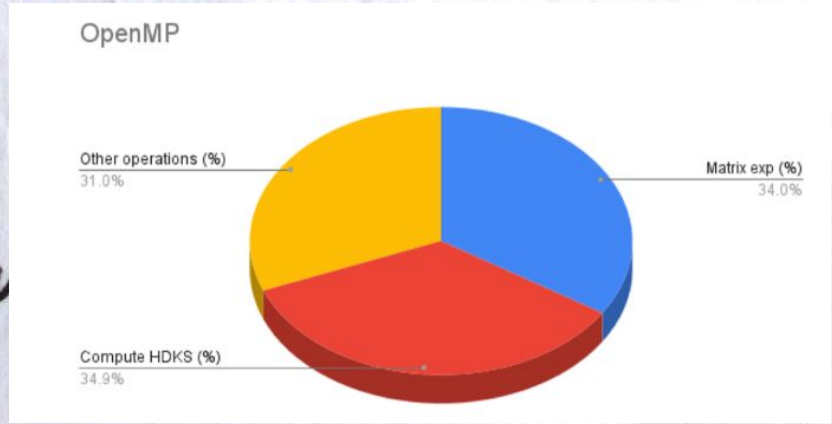
Openmp/OpenACC/Cupy
GPU+CPU(#32): 80 sec

RT-TDDKS for Au_4 (15 sec per time step) can be done in days instead of weeks!!

GPU porting of the code



$J^{TT}+K^{TT}$



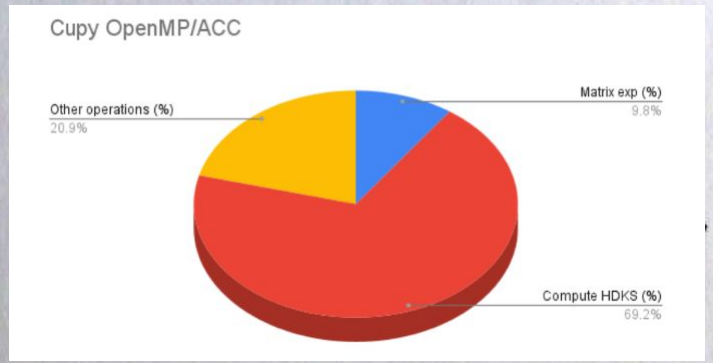


Table of contents

Conclusions

https://github.com/BERTHA-4c-DKS/

- Improve the data locality on GPU
- MultiGPUs in combination with our open-ended parallel implementation (use of ELPA library).

THANK YOU

Special thanks to:

- **Leonardo Belpassi** : collaboration on the BERTHA project
- **Matteo De Santis**
- Laura Bellentani, Sergio Orlandini (CINECA)
- Jeff Hammond (NVIDIA)

- PaGUSci - Parallelization and GPU Porting of Scientific Codes within the Cascading Call issued by ICSC , Spoke 3 Astrophysics and Cosmos Observations.
- Italian Research Center on High-Performance Computing, Big Data and Quantum Computing
- GPU Hackathon event for technical support and for awarding this project accessto the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy) and

Some references

- Loriano Storchi, Laura Bellentani, Jeff Hammond, Sergio Orlandini, Leonardo Pacifici, Nicolo' Antonini, Leonardo Belpassi, "Acceleration of the Relativistic Diracâ€“Kohnâ€“Sham Method with GPU: A Pre-Exascale Implementation of BERTHA and PyBERTHA", Journal of Chemical Theory Computation, DOI: 10.1021/acs.jctc.4c01759 (2025)
- Leonardo Belpassi, Matteo De Santis, Harry M. Quiney, Francesco Tarantelli, Loriano Storchi, "BERTHA: Implementation of a four-component Dirac-Kohn-Sham relativistic framework", The Journal of Chemical Physics, DOI: 10.1063/5.0002831 (2020).
- M. De Santis, L. Storchi, L. Belpassi, H. M. Quiney, F. Tarantelli, "PyBERTHART: A Relativistic Real-Time Four-Component TDDFT Implementation Using Prototyping Techniques Based on Python", Journal of Chemical Theory and Computation, DOI: 10.1021/acs.jctc.0c00053 (2020).
- Matteo De Santis, Leonardo Belpassi, Christoph R. Jacob, Andre' Severo Pereira Gomes, Francesco Tarantelli, Lucas Visscher, and Loriano Storchi, "Environmental effects with Frozen Density Embedding in Real-Time Time-Dependent Density Functional Theory using localized basis functions", Journal of Chemical Theory and Computation, DOI: 10.1021/acs.jctc.0c00603 (2020).
- Loriano Storchi, Matteo De Santis, Leonardo Belpassi, "BERTHA and PyBERTHA: State of the Art for Full Four-Component Dirac-Kohn-Sham Calculations", Advances in Parallel Computing, Parallel Computing: Technology Trends, DOI: 10.3233/APC200060 (2019)
- De Santis M., Rampino S., Quiney H.M., Belpassi L., Storchi L. "Charge-displacement analysis via natural orbitals for chemical valence in the four-component relativistic framework", Journal of Chemical Theory and Computation, DOI: 10.1021/acs.jctc.7b01077 (2018).
- S. Rampino, L. Belpassi, F. Tarantelli, and L. Storchi, "Full Parallel Implementation of an All-Electron Four-Component Dirac-Kohn-Sham Program", Journal of Chemical Theory and Computation, DOI: 10.1021/ct500498m (2014).
- Loriano Storchi , Sergio Rampino , Leonardo Belpassi , Francesco Tarantelli , and Harry M. Quiney, "Efficient parallel all-electron four-component Dirac-Kohn-Sham program using a distributed matrix approach.II", JCTC Journal of Chemical Theory and Computation, DOI: 10.1021/ct400752s (2013).
- L. Storchi, L. Belpassi, F. Tarantelli, A. Sgamellotti, H. M. Quiney, "An efficient parallel all-electron 4-component Dirac-Kohn-Sham program using a distributed matrix approach". Journal of Chemical Theory and Computation, DOI: 10.1021/ct900539m (2010).
- L. Belpassi, L. Storchi, F. Tarantelli, H. M. Quiney, "Recent advances and perspectives in 4-component Dirac-Kohn-Sham calculations", Physical Chemistry Chemical Physics, DOI: 10.1039/c1cp20569b (2011).