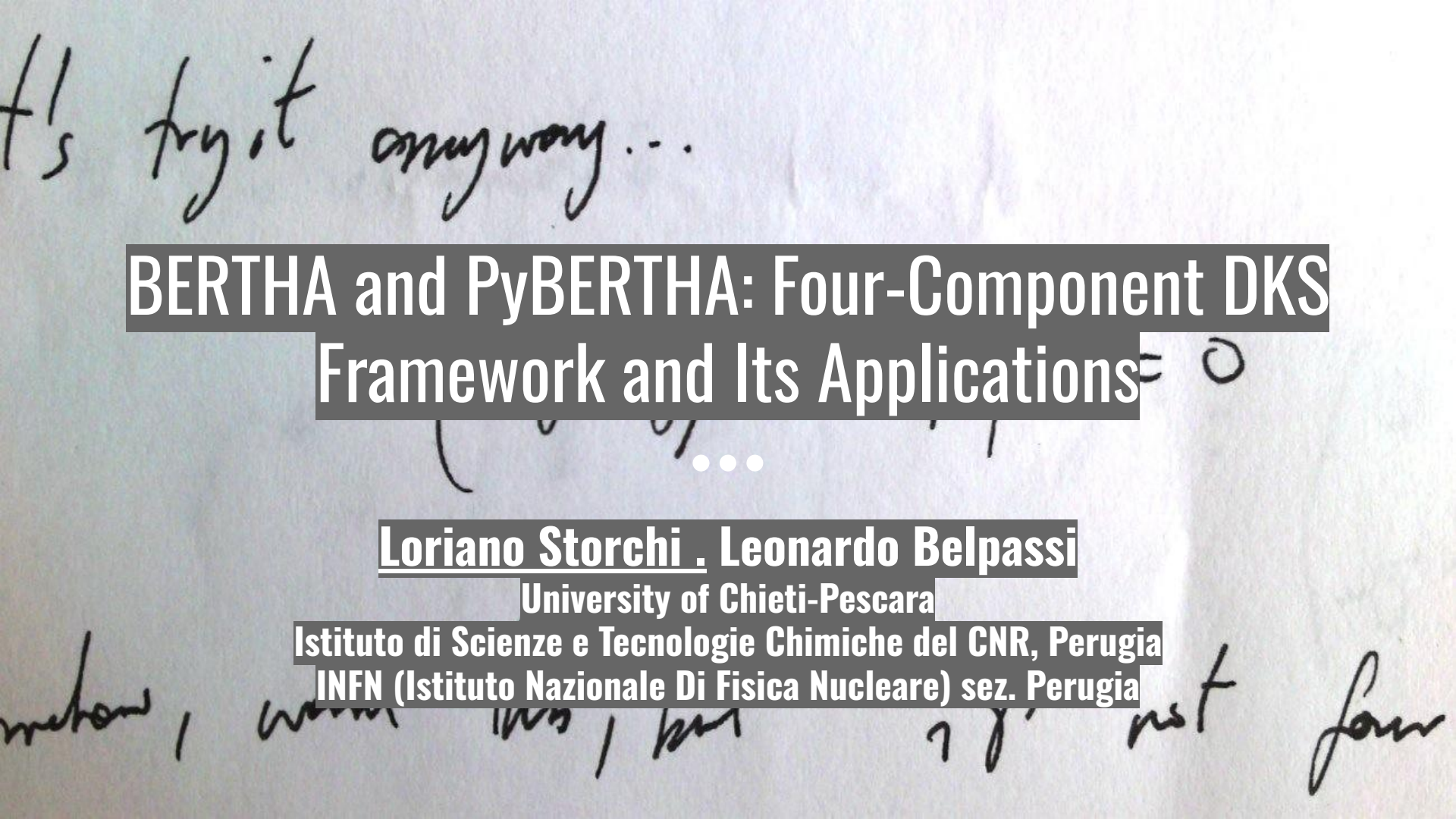


BERTHA and PyBERTHA: Four-Component DKS Framework and Its Applications

Loriano Storchi . Leonardo Belpassi

University of Chieti-Pescara

Istituto di Scienze e Tecnologie Chimiche del CNR, Perugia

INFN (Istituto Nazionale Di Fisica Nucleare) sez. Perugia

Where I Come From

Molecular Electronic Structure 2024

Società Chimica Italiana
Divisione di Chimica Teorica e Computazionale

Società Chimica Italiana
Sezione Abruzzo

HPC CINECA

My Research Background

01 / THEORY

Four Component Dirac-Kohn-Sham Theory (BERTHA code)

CNR & UNIPG

02 / ML & CHEMO

Machine Learning & Chemoinformatics

UNIPG & MolDiscovery

03 / PHYSICS & CLOUD

HEP (High Energy Physics) ML techniques, FPGA & Cloud Computing,

INFN & CERN

04 / BIO-INFO

Bio & Chemoinformatics

UNICH

My Research Background

01 / THEORY

Four Component Dirac-Kohn-Sham Theory (BERTHA code)

CNR & UNIPG

02 / ML & CHEMO

Machine Learning & Chemoinformatics

UNIPG & MolDiscovery

03 / PHYSICS & CLOUD

HEP (High Energy Physics) ML techniques, FPGA & Cloud Computing

INFN & CERN

04 / BIO-INFO

Bio & Chemoinformatics

UNICH

ML & CHEMO: Interpretable ML for DFT reaction energies

Collaboration with G. Bistoni • first work published in JCTC 2025 • update in progress

RF + LR correction pipeline

Low-cost quantum chemistry in → transparent correction out

1 — LOW-COST DFT

PBE / PBE0
reaction energy

2 — ENERGY DESCRIPTORS

ΔT_e ΔV_{NN} ΔV_{eN}
ΔE_X ΔE_C ΔE_J
ΔE_{disp}

3 — RF SELECTOR

chooses the most suitable local model

4 — LR CORRECTION

$\Delta E^{LR} = w^T \Delta E + b$
global / specialized

REFERENCE / LABELS: CCSD(T)/CBS

INTERPRETABLE

LIGHTWEIGHT

MODULAR

Goal: bridge inexpensive DFT calculations toward high-level reaction-energy accuracy without another QM calculation.

WHY IT MATTERS

Highly interpretable and computationally inexpensive correction of DFT reaction energies.

PUBLISHED PROOF-OF-CONCEPT

207.3% → 84.4%

PBE/MINIX MAPE DFT → RF/LR

PROJECT STATUS

PUBLISHED • 2025

Update: new data + methodological refinements + broader validation

Jacinto-Mejía • Storchi • Bistoni | JCTC 2025, 21, 10853–10862 | DOI 10.1021/acs.jctc.5c01184

From Different Fields to BERTHA

01 / THEORY

Four Component Dirac-Kohn-Sham Theory (BERTHA code)

CNR & UNIPG

02 / ML & CHEMO

Machine Learning & Chemoinformatics

UNIPG & MolDiscovery

03 / PHYSICS & CLOUD

HEP (High Energy Physics) ML techniques, FPGA & Cloud Computing

INFN & CERN

04 / BIO-INFO

Bio & Chemoinformatics

UNICH

Table of contents

Table of contents

Active molecular codes based on Dirac eq.

DIRAC
Sbae (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

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**
- **A particularly suitable and promising theoretical framework to appropriately treat these systems is the Dirac-Kohn-Sham model (DKS)**

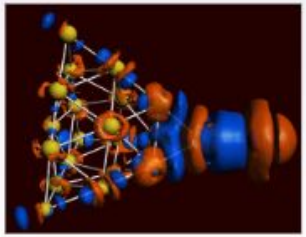
Relativistic DKS: Computational Challenges

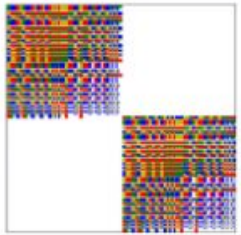
The BERTHA Choice: All-Electron Four-Component DKS

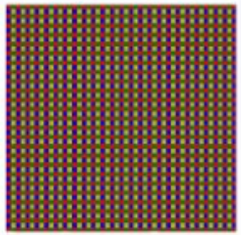
- Core and valence electrons are both treated explicitly
  - Relativistic effects are particularly important for inner-shell electrons near heavy nuclei, where the strong nuclear electric field “accelerates them to velocities” that are a significant fraction of the speed of light.
- Heavy nuclei
  - Heavy and superheavy atoms contain many electrons.
- Matrices become So Large
  - From a Real N×N Problem to a Complex 2N×2N Problem

Relativistic DKS: Computational Challenges

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







OpenMP

2003 ① Linux Beowulf Cluster

2009 ② SGI (GE) cc-NUMA cache-coherent Non Uniform Memory Access

2014 ③ IBM BG/Q Cineca (IT)

2019 ④ Cray x30 EPCC (UK)

2024 >>> CPU & GPU

Computational bottleneck & density fitting

THE BOTTLENECK

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

Because **H**DKS depends on the density through **J** and **K**, the expensive **J** + **K** matrix must be rebuilt at every SCF iteration.

Repeated integral evaluation + large complex matrices

OUR STRATEGY

$$\rho(\mathbf{r}) \approx \tilde{\rho}(\mathbf{r}) = \Sigma \square \, \mathrm{d}\square \, \mathrm{f}\square(\mathbf{r})$$

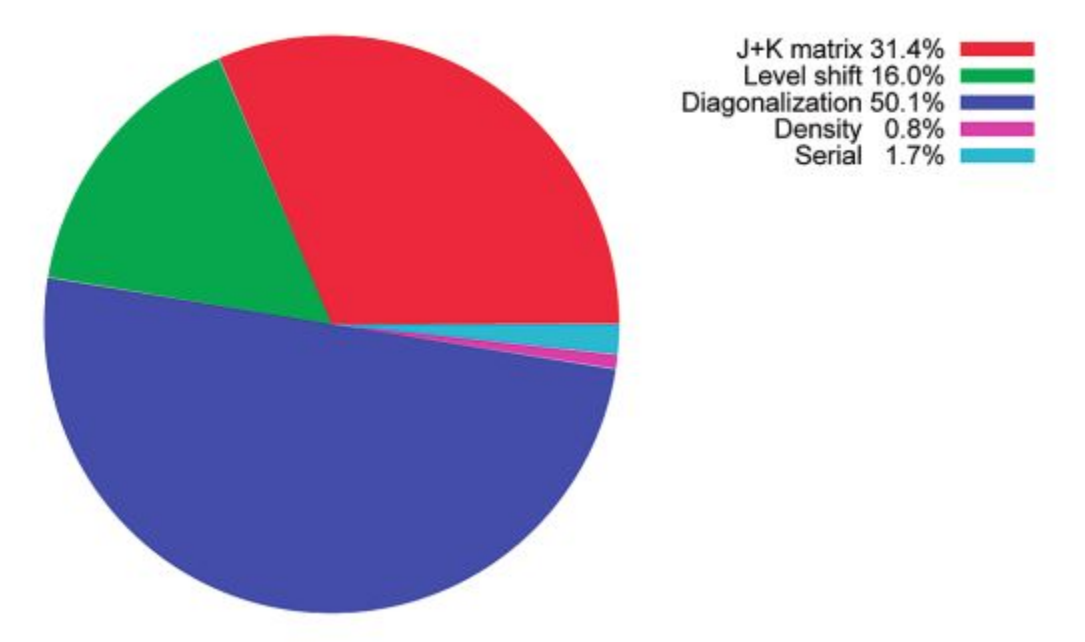
Density fitting expands the electronic density in an auxiliary basis, reducing the cost of **J** + **K** construction.

$$O(N^4) \rightarrow O(N^3)$$

Au□ benchmark: 251×–884× speed-up for J + K construction

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

01

Fully parallel SCF workflow

Remove all serial sections to prevent an early speedup ceiling imposed by Amdahl's law.

02

Distributed-memory design

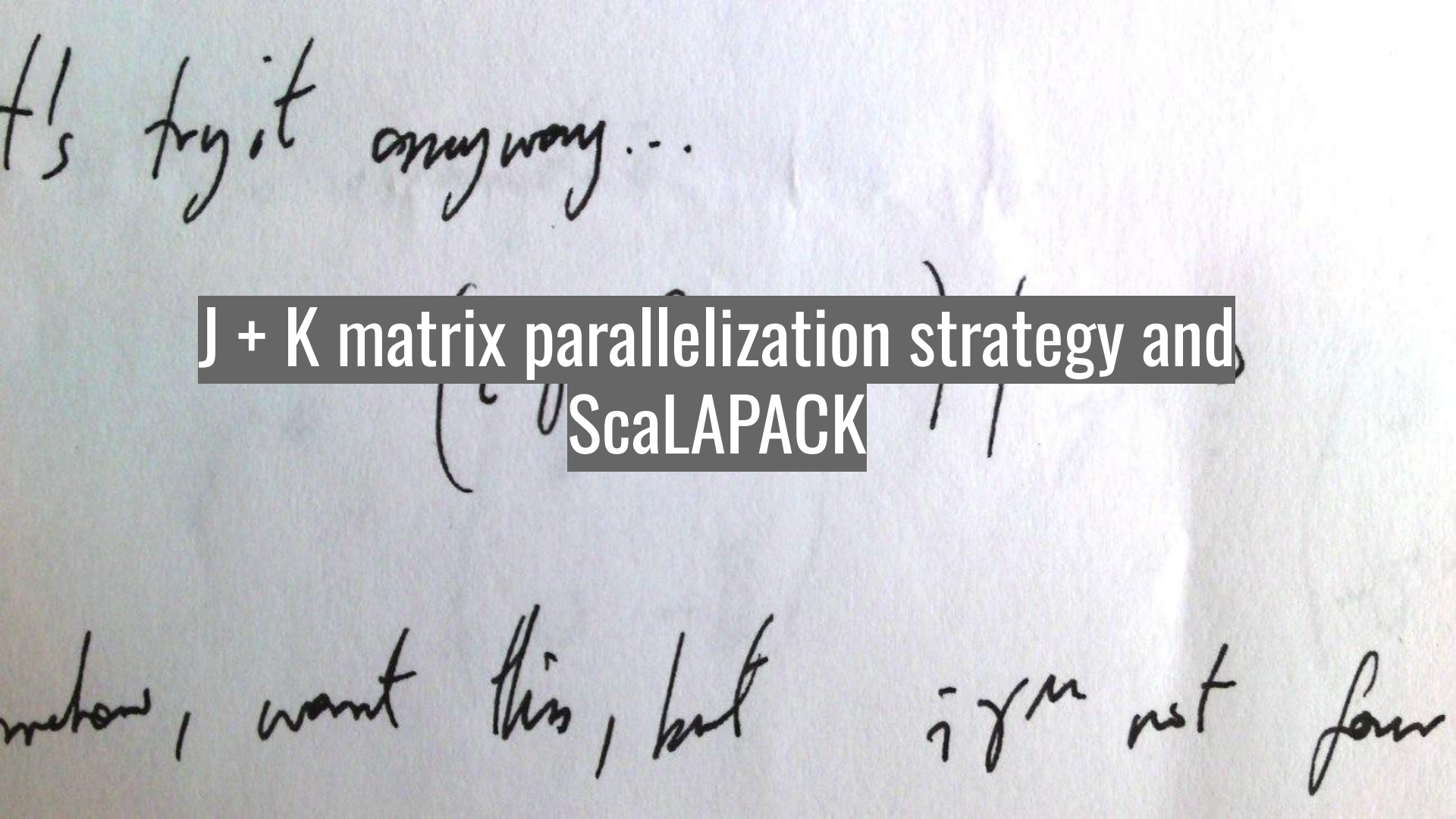
Partition the large 2N × 2N (or 4N * 4N...) complex matrices across MPI processes, reducing the memory burden per process.

03

MPI + ScaLAPACK

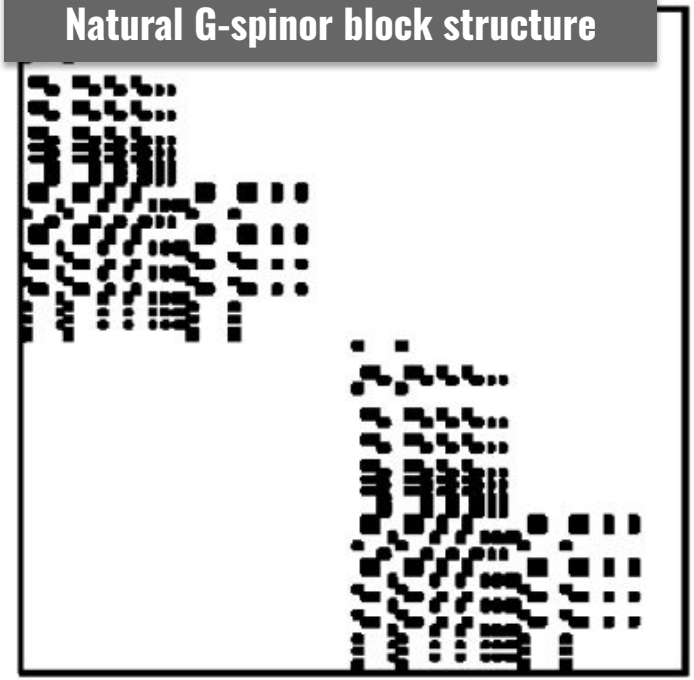
MPI manages distributed data and communication; ScaLAPACK replaces serial LAPACK for dense linear algebra using a 2D block-cyclic layout.

Target: CPU time and memory scale with the number of MPI processes.


J + K matrix parallelization strategy and ScaLAPACK

The matrix structure dictates the parallelization

Natural G-spinor block structure



1 Matrix shape

G-spinor functions grouped by common origin and angular momentum generate an irregular, problem-specific block pattern.

2 Integral-Driven Distribution (IDD)

Each natural block becomes a work unit and is assigned cyclically to MPI processes for balanced J + K construction.

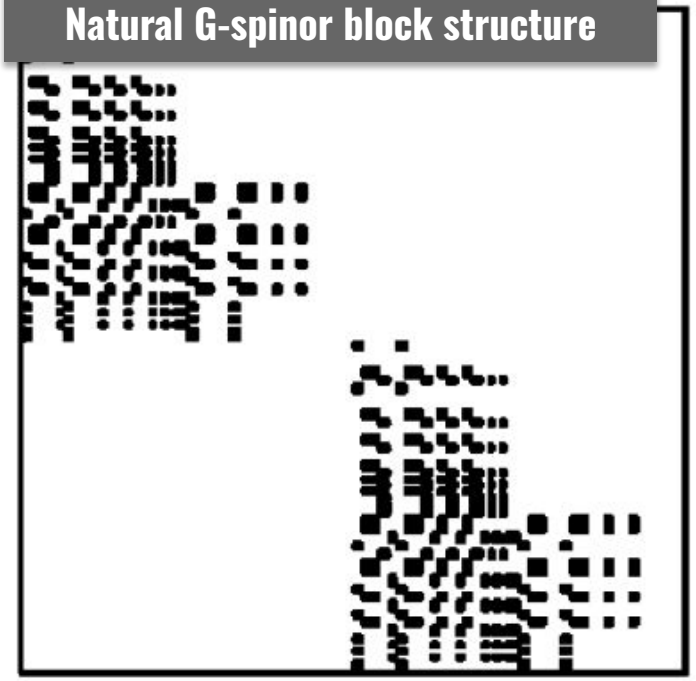
3 Interface to parallel linear algebra

The assembled matrix is then remapped from IDD to ScaLAPACK's block-cyclic distribution for the linear-algebra phase.

Take-home: the decomposition is dictated by the mathematical structure of the problem—not chosen arbitrarily.

The matrix structure dictates the parallelization

Natural G-spinor block structure



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.

PSEUDOCODE

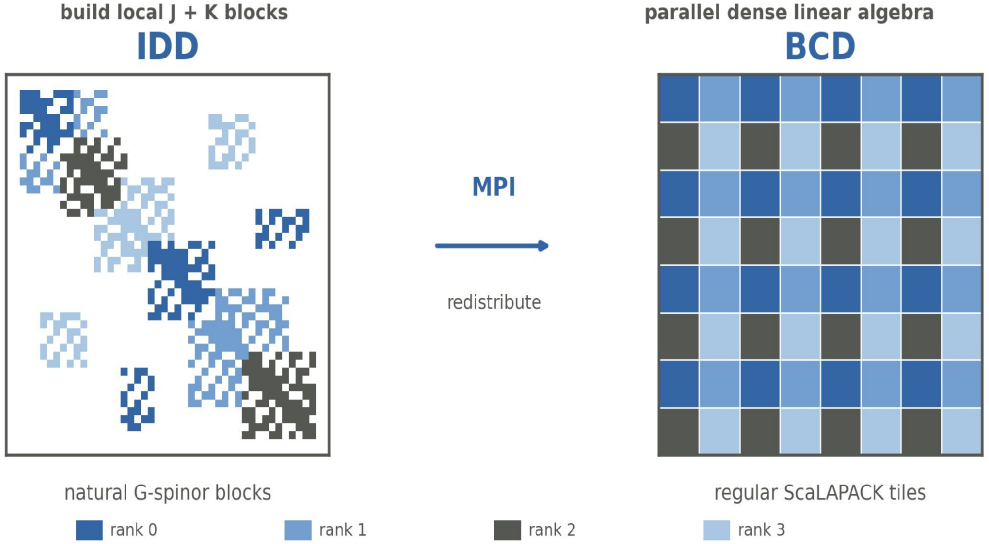
```
for each block b with r = block_id(b) mod P do
    → evaluate density-fitting integrals for b
    → build local (J + K)[b] in the IDD layout
end for
```

`b` = block | `r` = rank assigned | `P` = number of MPI processes | `IDD` = Integral-Driven Distribution | `(J + K)[b]` = Coulomb + Exchange matrices for block `b`

Take-home: the decomposition is dictated by the mathematical structure of the problem—not chosen arbitrarily.

From IDD matrix construction to ScaLAPACK

Two layouts optimized for different kernels



1 Build in IDD
Each MPI rank evaluates its assigned G-spinor blocks and constructs its local part of J + K.

2 Redistribute with MPI
The distributed matrix is remapped from the irregular IDD layout to a regular block-cyclic layout.

3 Compute with ScaLAPACK
Level shift, complex diagonalization, and density construction run on the BCD matrix.

Take-home: IDD follows the integral structure; BCD enables scalable ScaLAPACK operations.

How the IDD → BCD remapping works

1. **Describe destinations**
   Build a compact info array containing destination ranks and local BCD indices.

2. **Pack local data**
   Allocate send buffers and pack each owned IDD block with its destination metadata.

3. **Exchange with MPI**
   Process senders in turn; communicate only the packed data required by each rank.

4. **Unpack and release**
   Insert received data into local BCD tiles, then free temporary receive buffers.

</> MPI redistribution pseudocode

```
1  for sender = 0 ... P - 1 do
2    broadcast info[sender]
3    if rank == sender then
4      pack local IDD blocks
5      send packed data
6    else
7      allocate buffers from info[sender]
8      receive + unpack into BCD tiles
9      free receive buffers
10   end if
11 end for
```

[J + K]IDD → [J + K]BCD

Take-home: metadata converts irregular IDD blocks into ScaLAPACK's regular BCD tiles.

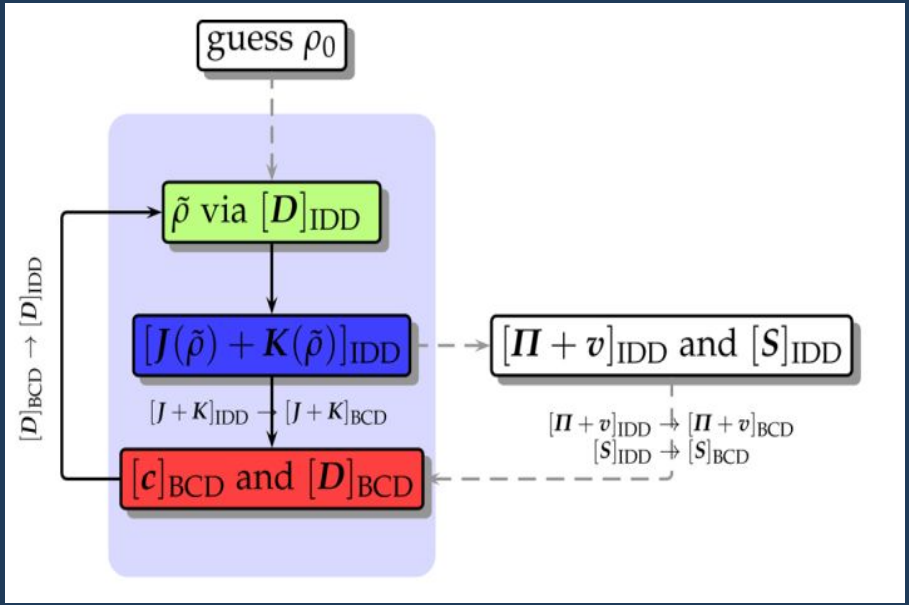
t's try it anyway...

$$(i\gamma^\mu \partial_\mu - m)\psi = 0$$

Distributed-Memory Parallelization

motion, want this, but $i\gamma^\mu$ not four

Distributed-Memory Parallelization



Matrix Integration Strategy

A similar strategy has been adopted for the one-electron and superposition matrices to ensure consistent parallel execution.

IDD Scheme & Representation

In the IDD scheme, distribution is much less regular. An efficient representation is obtained using a derived data type, composed of a 2D array and metadata describing its size and global placement.

An array of these derived data types is then used on each process to identify each local IDD block.

Workflow Assembly

Sum all matrices to obtain the final Hamiltonian matrix (HDKS).

Parallelization Strategies

Wall-Clock Time in Seconds Spent in the Distribution Mapping Routines

Average value over 4 Self-Consistent Field (SCF) cycles 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.

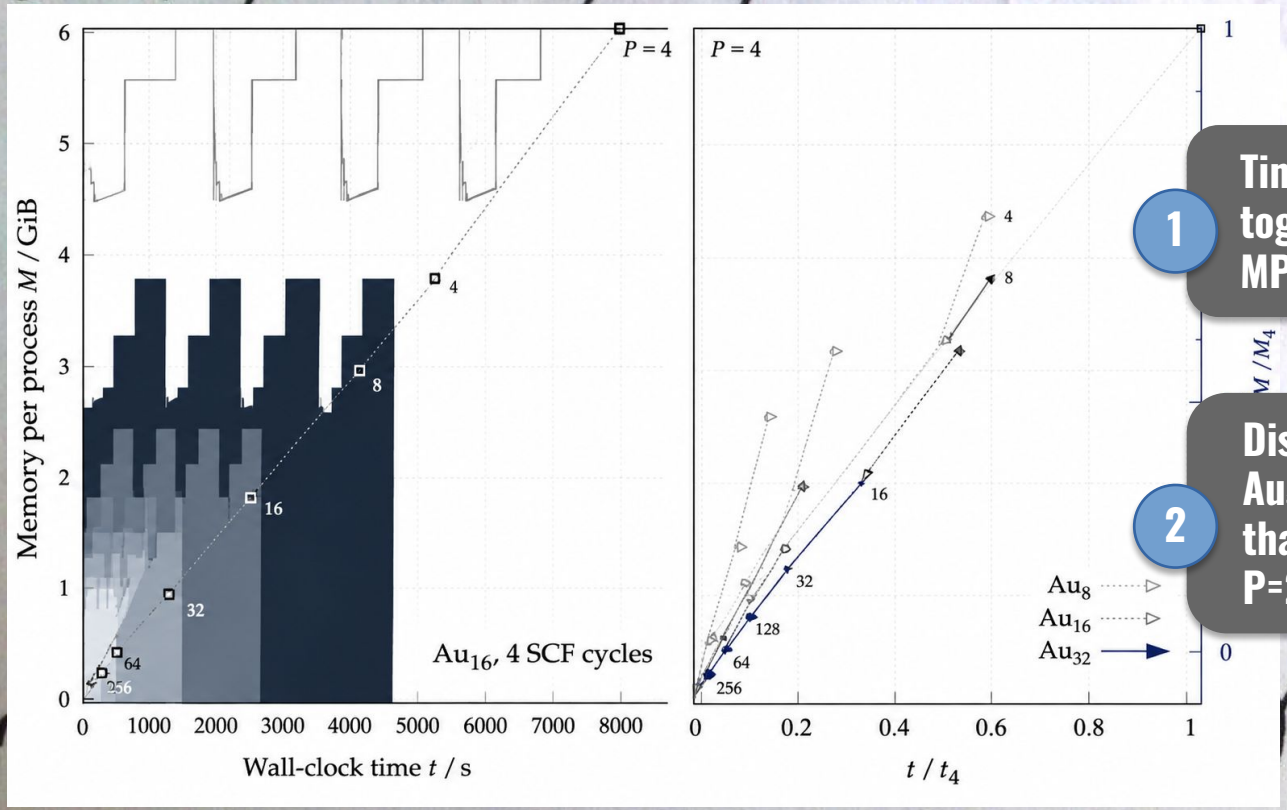
Parallelization Strategies

Memory Per Process Peak in MiB over P Processes

Average Value (Mav), Maximum Positive (Δ+) and Negative (Δ-) Deviations

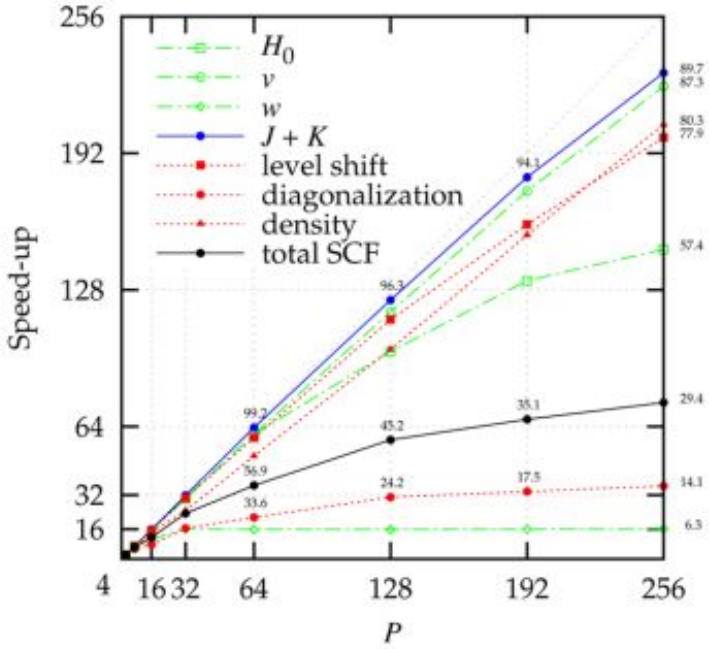
	$[(Ph_3P)Au(C_2H_2)]^+$			Au_8			Au_{16}			Au_{32}		
P	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+
4	76	1842	37	34	1882	60	81	6214	46	126	23835	187
8	57	1253	59	64	1303	71	54	3770	90	88	14106	232
16	12	884	15	8	888	44	14	2124	23	8	7405	63
32	62	799	84	33	773	96	44	1499	85	53	4827	155
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

Distributed-Memory Parallelization



1. Time and memory scale together with the number of MPI processes.
2. Distributed matrices reduce the Au_{32} memory footprint to less than 2 GiB per process at P=256

Distributed-Memory Parallelization



SPEEDUP PERFORMANCE

Speedup for all of the computational kernels of the SCF procedure for Au32 as a function of the number of processors P.

SYSTEM CONFIGURATION

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

Table of contents

PyBERTHA: A Python Binding for BERTHA

01

Scientific Standard

Undoubtedly the Python programming language is emerging as one of the most important and used HLL also in the field of scientific computing.

02

Rapid Prototyping

Python HLL, besides providing an extensive range of modules to solve a comprehensive set of computational problems, enables quick prototyping.

03

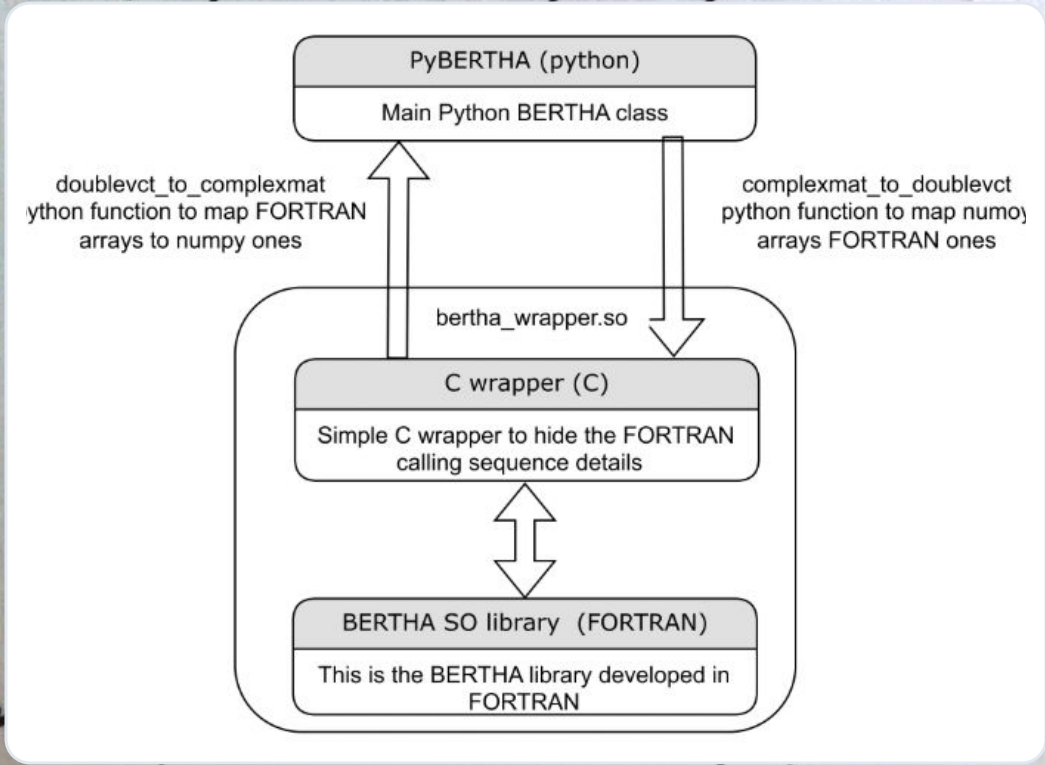
Natural Choice

With its combination of power, modules, and accessibility, Python is clearly a natural choice for the BERTHA project.

PyBERTHA: A Python Binding for BERTHA

An overview of the software and High-Level Language (HLL) layers

PyBERTHA: A Python Binding for BERTHA



Bridge Implementation

To facilitate seamless data movement between **Python** and **Fortran**, the architecture utilizes:

- A couple of specialized Python functions.
- Specific lines of mapping code at the **bertha_wrapper/Fortran** layer.

PyBERTHA a Python binding for BERTHA

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

```
import berthamod
Inputs: input options ...
bertha = berthamod.pybertha(wrapperso)
bertha.set_verbosity(verbosity)
bertha.set_fnameinput(inputfilename)
bertha.set_fittfname(fittfilename)
bertha.set_tresh(tresh)
bertha.init()
ovapmtx, eigenvectors, fockmtx, eigenvalues = bertha.run()
etotal = bertha.get_etotal()
bertha.finalize()
Output: Total Energy and MO energies ...
```

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.
10: etotal = bertha.get_etotal()
11: bertha.finalize()
12: Output: Total Energy and MO energies ...
```

NEW • AUTOMATED WORKFLOW

Automatic basis + fitting sets

- **fitt_gen**: automatic auxiliary basis from the selected G-spinor basis.
- **berthainputbasis + berthaingen**: basis/fitting library + BERTHA input directly from XYZ.

H → Og | full periodic table

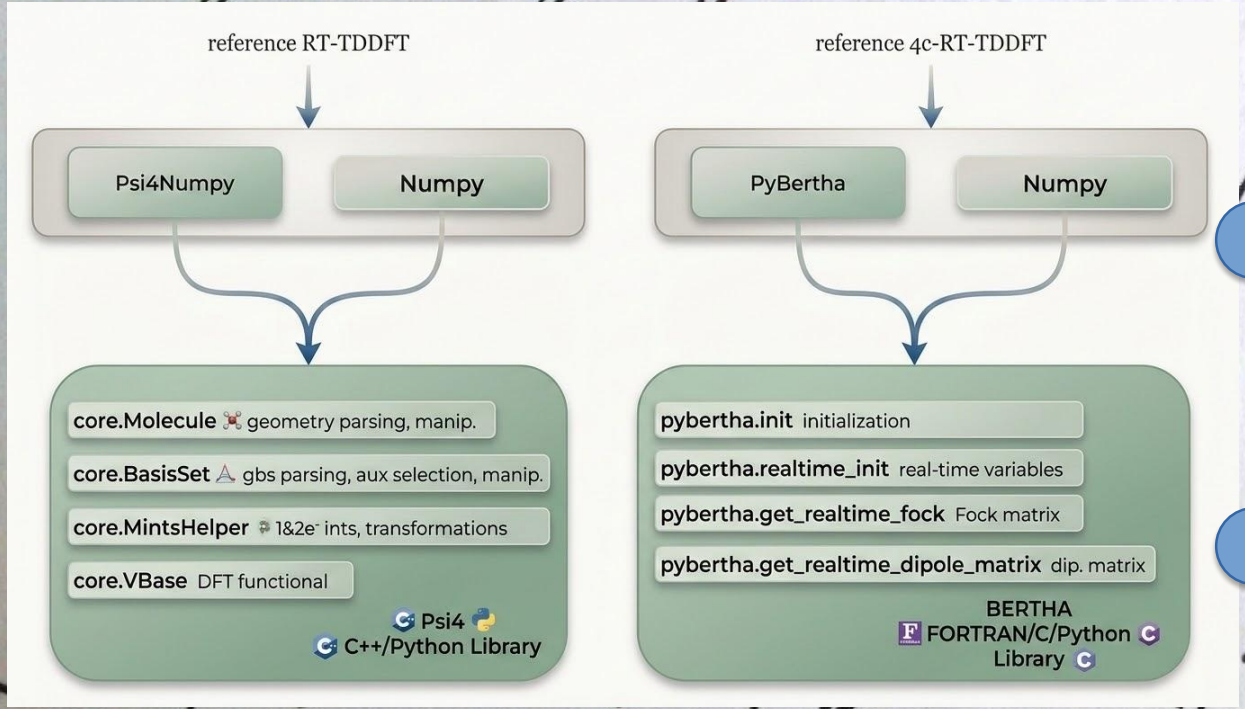
Antonini et al., JPCA 2025

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 %

PyBERTHART a real-time TDDFT implementation



FIRST AUTHOR
M. De Santis et al. • JCTC 2020
16, 2410–2429 • DOI: 10.1021/acs.jctc.0c00053

Porting

1 The RT-TDDFT propagation is first validated in Psi4NumPy and then transferred to the four-component PyBERTHA framework.

Python with negligible overhead

2 PyBERTHA exposes the DKS and dipole matrices directly; the Python interface adds <1% SCF overhead and does not affect propagation stability.

Table of contents

PyBERTHA: A Python Binding for BERTHA

PARALLEL PyBERTHA

1 MPI-Based Parallelization

First option is to load the `liberrthaparallelshm.so` library.

This allows you to manage MPI at the Python level using modules such as `mpi4py` or similar alternatives.

2 OpenMP Shared-Memory

An alternative option is to utilize OpenMP shared-memory multithreading.

This is especially useful and highly effective when interested in improving performance for smaller molecular systems.

PyBERTHA: Python Binding for BERTHA

Performance Analysis: OpenMP (Multithreading) is better or equivalent to MPI for small systems

			OpenMP				MPI			
System	Step	Serial	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

Comparative execution times (seconds) for Zn, Hg, and Au_2 molecular systems

PyBERTHA: Python Binding for BERTHA

Performance Analysis: big Systems MPI better than 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

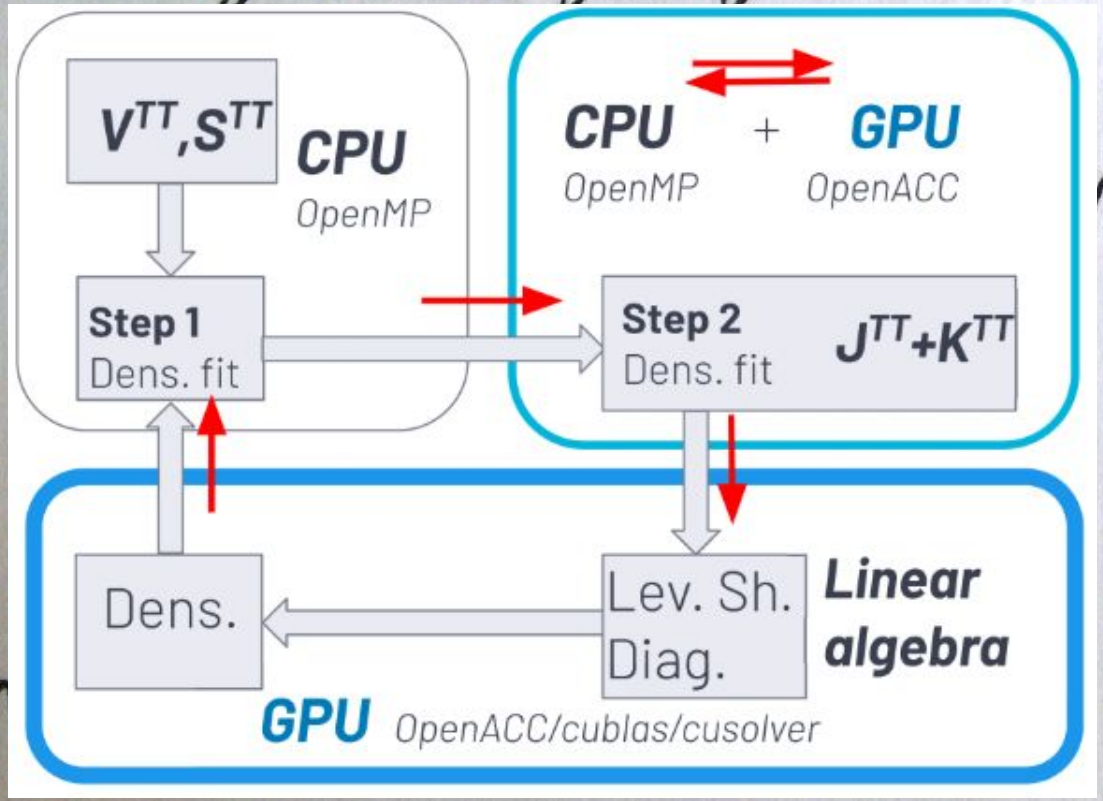
Comparative execution times (seconds) for Au_4 , Au_8 , Au_{16} molecular systems

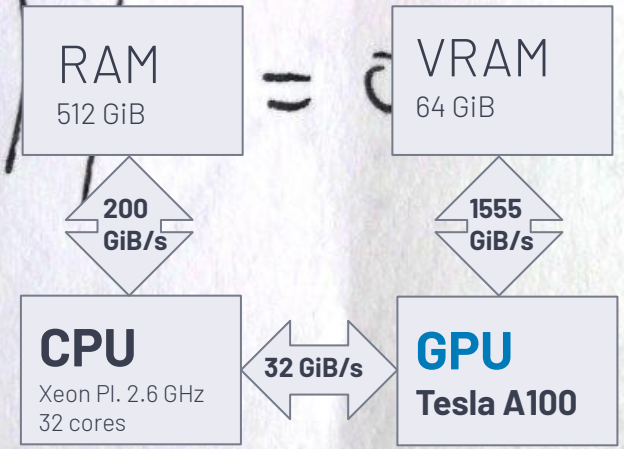
Table of contents

GPU porting of the code

OPEN HACKATHONS

CINECA





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

GPU Porting of the Code

Key Insight:
Linear algebra accounts for **70% of the total cost** in a serial run.

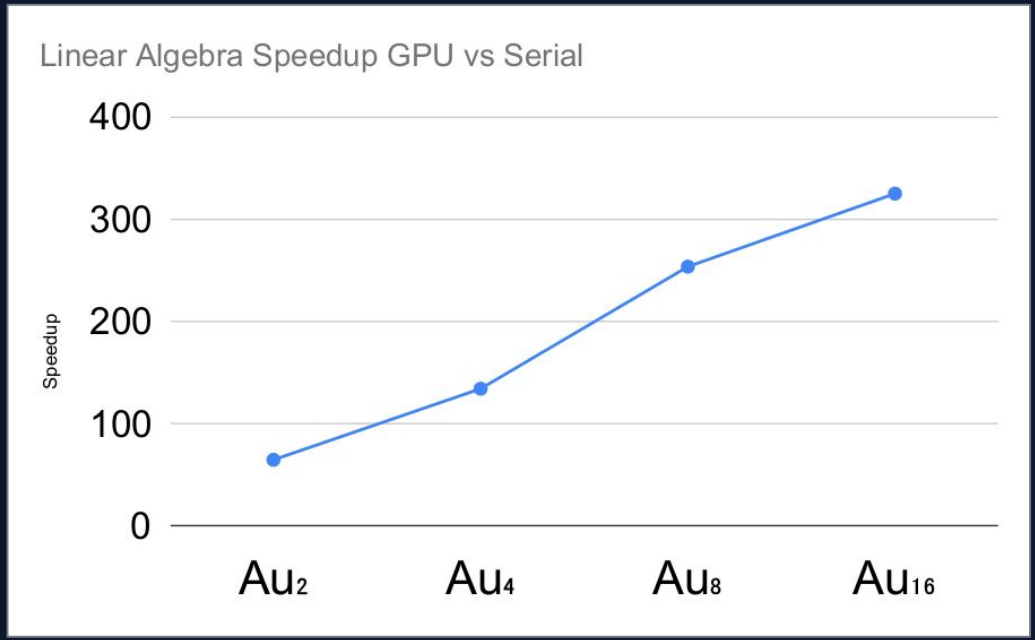
Environment Specifications

- **Au:** Dyall VDZ
- **CPU (Serial):** OpenBLAS
- **GPU:** cuBLAS
- **Compiler:** nvfortran

Au16 Benchmark (dim 12608)

Serial CPU: `4761.1 sec`
GPU Accelerated: `14.7 sec`

324x Speedup



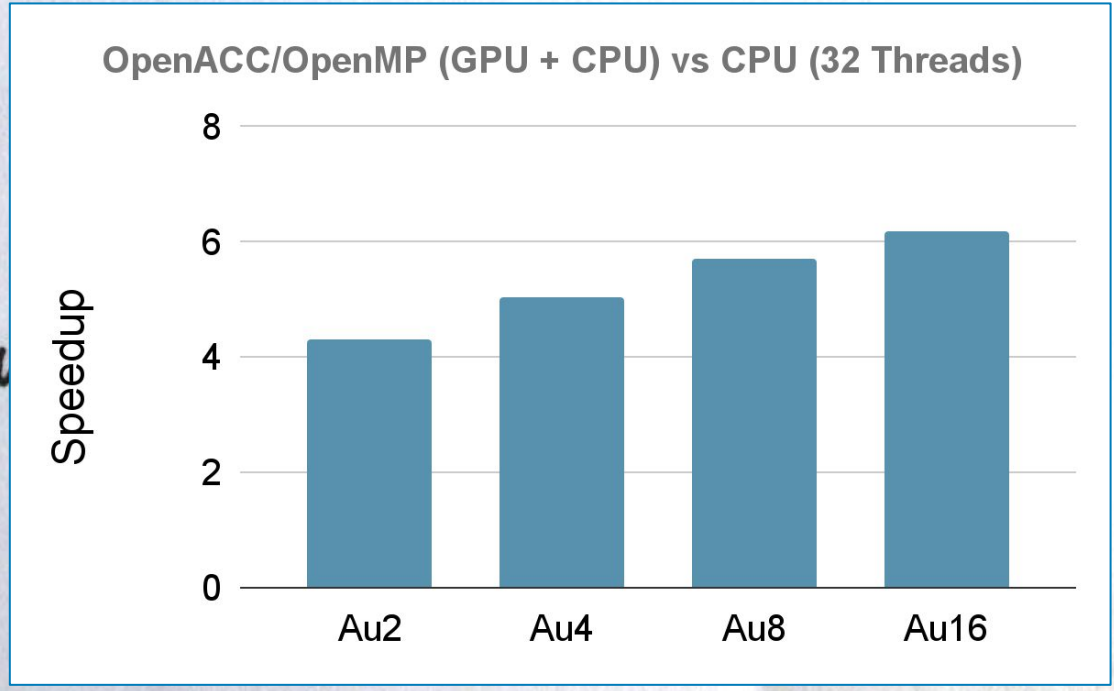
GPU porting of the code

CPU + **GPU**

Openmp *OpenACC*

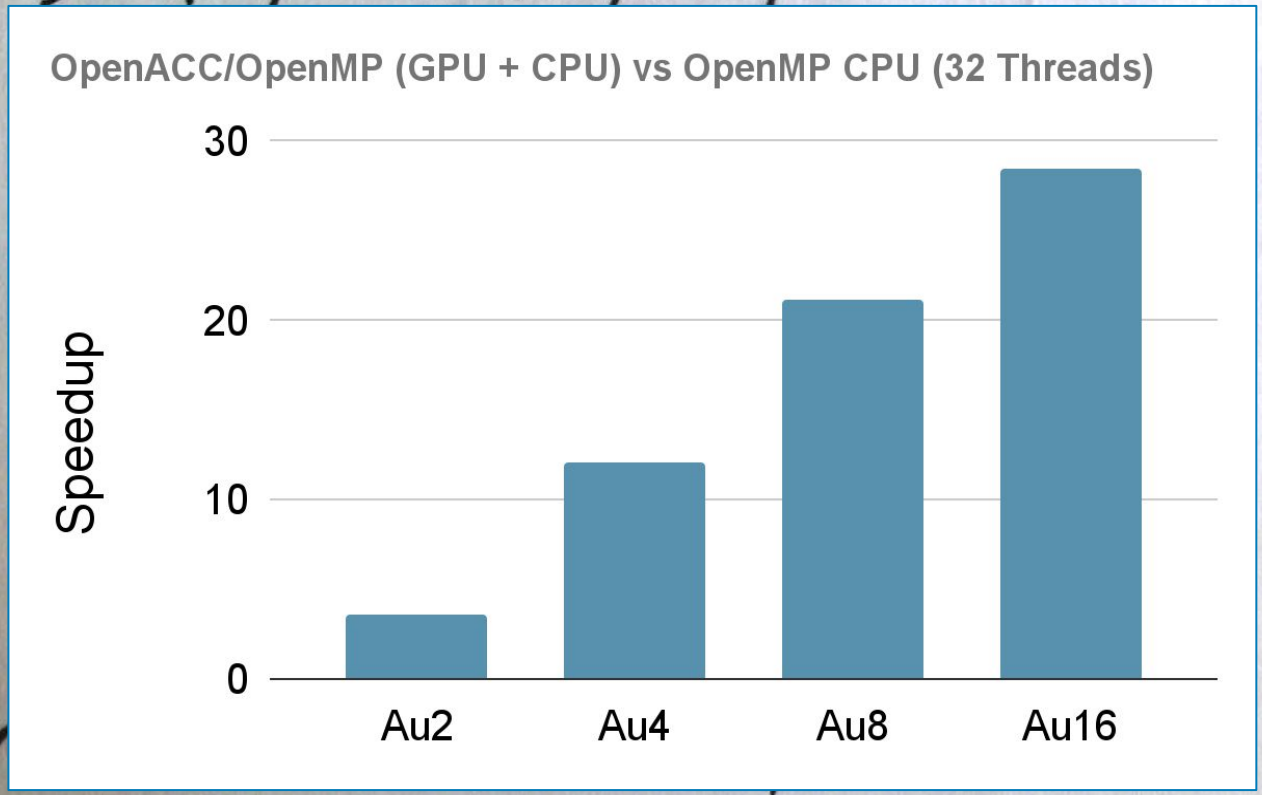
Step 2
Dens. fit $J^{TT}+K^{TT}$

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

RT-TDDKS: Python / PyBERTHA workflow

Python drives the real-time loop; PyBERTHA calls BERTHA to rebuild H_DKS from the instantaneous density at every step.

PYTHON • PyBERTHA-RT

- **Owns the time loop:** Magnus + predictor/corrector.
- **Linear algebra:** NumPy / CuPy propagation.
- **Advance:** D(t) → D(t+Δt).

D(t)

PyBERTHA API

H_DKS(t)

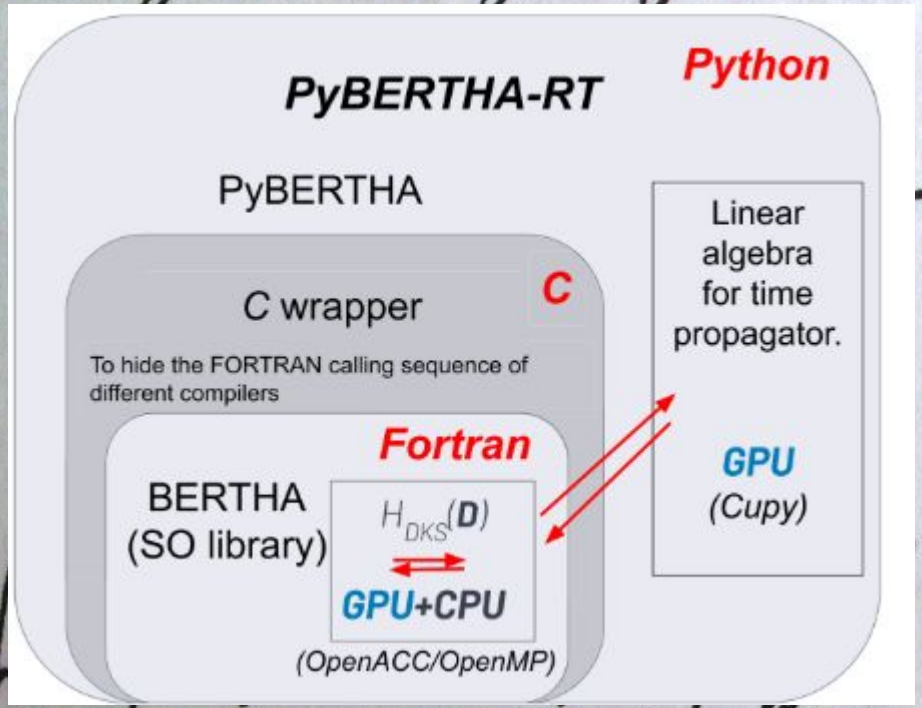
PyBERTHA → BERTHA

- **Receives D(t):** instantaneous density from Python.
- **Rebuilds H_DKS[D(t)]:** updates J + K and the full 4c Hamiltonian.
- **Backend:** Fortran; OpenMP / OpenACC.

D(t) → PyBERTHA / BERTHA → H_DKS[D(t)] → Python propagator → D(t+Δt) → repeat

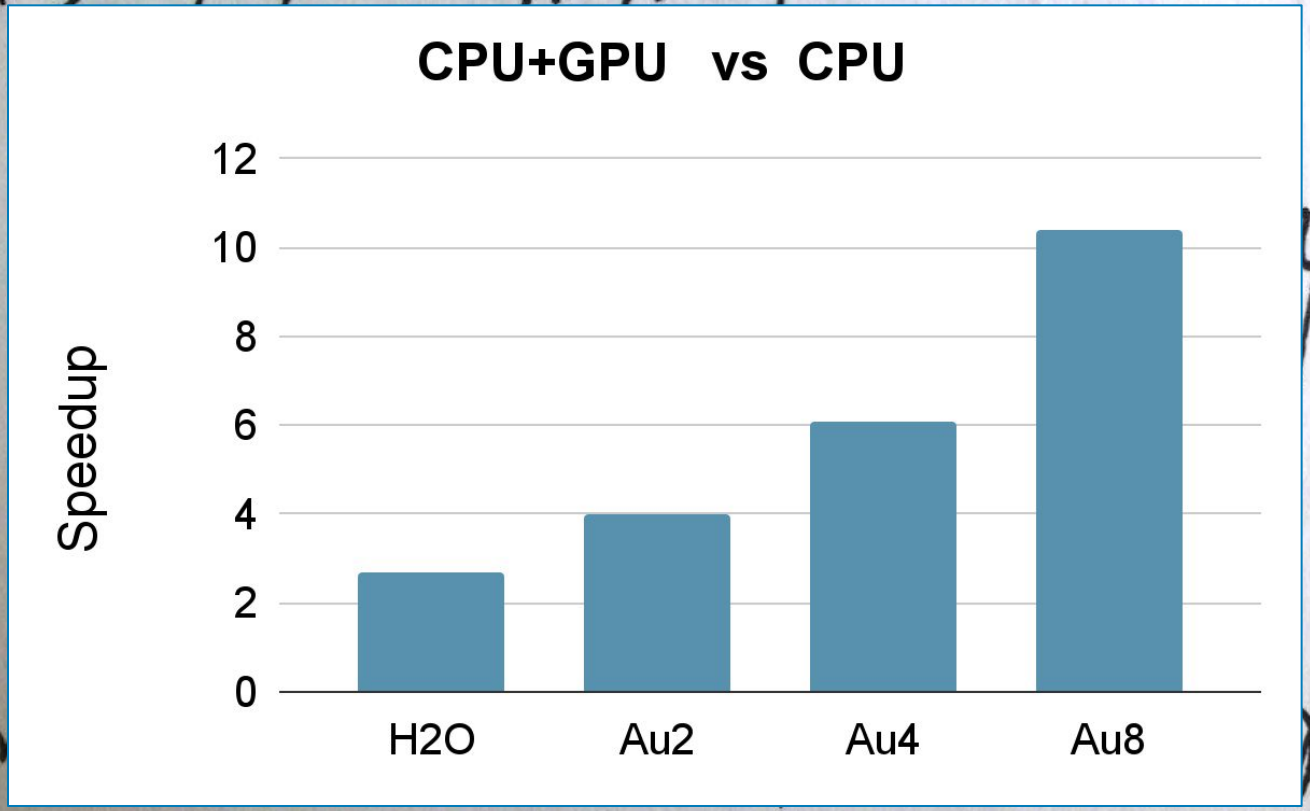
Python drives the propagation; BERTHA owns the expensive H_DKS construction.

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 poreting 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!!

Table of contents

t's try it anyway...

$$(i\gamma^\mu \partial_\mu - m)\psi = 0$$

motion, want this, but $i\gamma^\mu$ not four

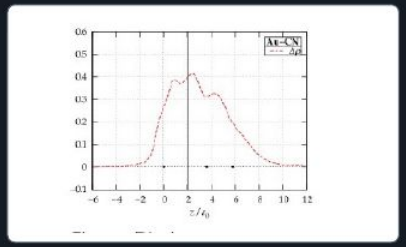
NOCV/CD analysis: from charge flow to bonding channels

METHOD

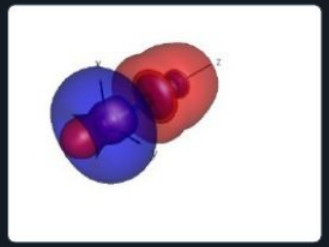
1 Charge Displacement (CD)

Integrate the density difference across planes perpendicular to the bond axis to obtain the amount and direction of charge flow.

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



(a) Charge-displacement function for Au–CN



(b) Δρ = ρ(AB) − ρ(A) − ρ(B)

CD value at an isodensity boundary → quantitative charge transfer

2 NOCV decomposition

Diagonalize the density rearrangement into a small set of additive, chemically meaningful channels.

$$\Delta\rho' = \sum_k \nu_k \left(|\phi_k|^2 - |\phi_{-k}|^2\right) = \sum_k \Delta\rho'_k$$

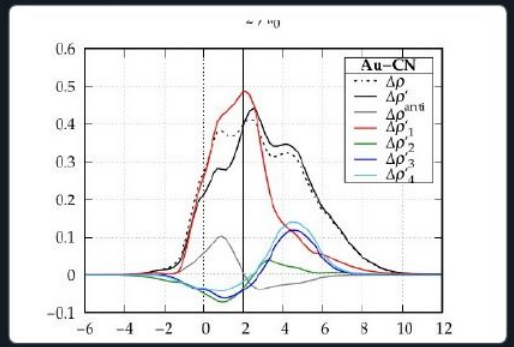
1. **A few NOCV pairs dominate**
   Only the leading pairs carry chemically significant density rearrangement.
2. **Donation / back-donation become explicit**
   Each Δρ′k is integrated with CD and assigned a quantitative charge-transfer value.
3. **Naturally suited to relativistic bonding**
   The analysis remains meaningful when spin–orbit coupling reduces symmetry.

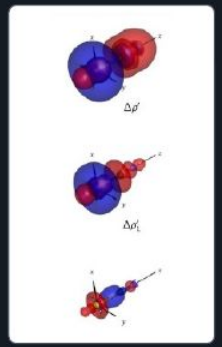
Take-home: CD tells how much charge moves; NOCV tells which bonding channel moves it.

Relativistic NOCV/CD in practice: chemistry and scalability

APPLICATION

AuCN — relativistic bond decomposition





0.484 e

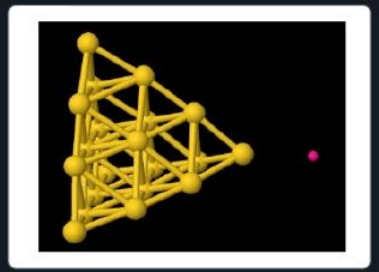
$CN^- \rightarrow Au^+$ donation

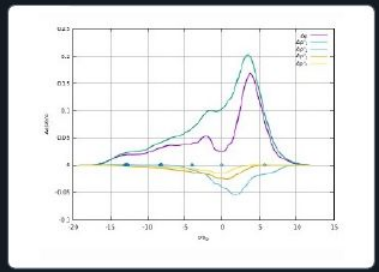
−0.041 / −0.017 e

Au → CN back-donation

Spin–orbit coupling lifts the Au 5d degeneracy and splits the back-donation channels; gold shows the strongest donation and back-donation of Cu/Ag/Au.

Au_{20}–Og — NOCV/CD at large scale





1. **1698 electrons; DKS matrix > 10 GB**

   Workload and memory distributed over 64 MPI processes.

2. **Net charge transfer ≈ 0.178 e**

   The dominant NOCV channel contributes ≈ 0.199 e and reveals the major Og → Au_{20} charge-flow pathway.

Take-home: NOCV/CD turns a four-component density into chemically interpretable charge-flow channels—even for very large heavy-element systems.

t's try it anyway...

$$(i\gamma^\mu \partial_\mu - m)\psi = 0$$

motion, want this, but $i\gamma^\mu$ not four

RT-TDDKS in PyBERTHA: from time propagation to spectra

METHOD

Four-component time evolution

$$i\,\partial\psi_i(r,t)/\partial t = \hat{H}\_DKS[\rho(t),t]\;\psi_i(r,t)$$

The DKS matrix is rebuilt self-consistently from the instantaneous density at every time step.

1 Perturb
Weak δ-kick for linear response or shaped strong field for nonlinear dynamics.

2 Propagate
Second-order midpoint Magnus + self-consistent predictor/corrector.

3 Observe
Record induced dipole μ(t) at every time step.

LINEAR RESPONSE

$\mu(t) \rightarrow$ Fourier transform $\rightarrow S(\omega)$

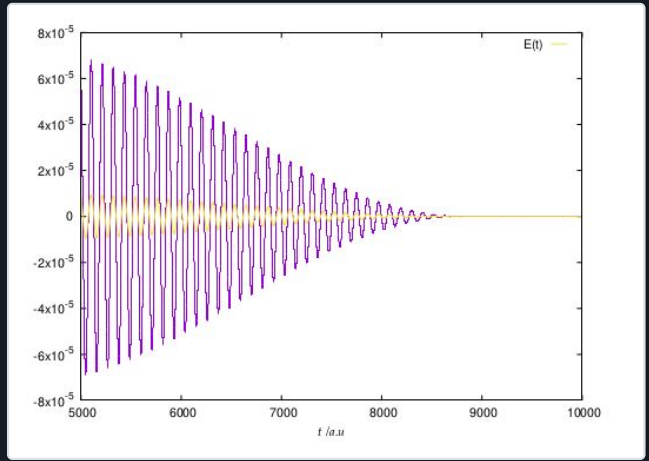
A δ-pulse contains all frequencies, so one propagation probes the full spectrum.

NONLINEAR RESPONSE

Strong field $\rightarrow \mu(t) \rightarrow$ HHG

The same four-component propagator remains stable in the strong-field regime.

H_2: a direct stability check



Weak field
dipole follows E(t)

Strong-field capability
same propagator → nonlinear dynamics / HHG

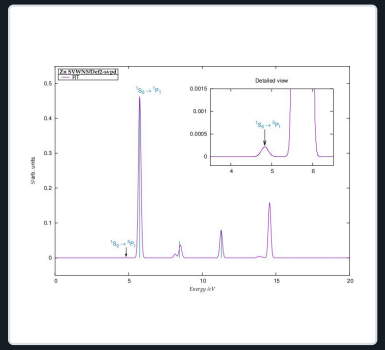
Take-home: real-time propagation converts the four-component DKS Hamiltonian directly into spectra and electron dynamics.

Relativistic spectroscopy in real time: Zn → Cd → Hg

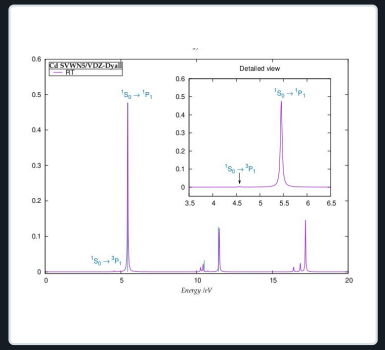
APPLICATION

Spin-forbidden intensity is a sensitive spin–orbit probe

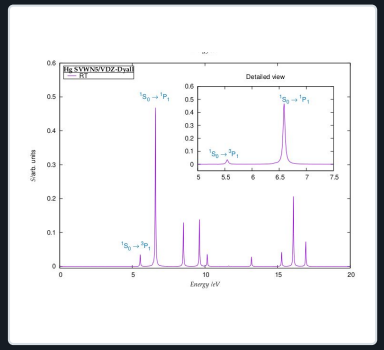
The nominally forbidden $^1S_0 \rightarrow {}^3P_1$ line gains oscillator strength as relativistic spin–orbit mixing increases down group 12.



Zn $I(^3P_1) / I(^1P_1)$ **4.7×10^{-4}**



Cd $I(^3P_1) / I(^1P_1)$ **5.6×10^{-3}**



Hg $I(^3P_1) / I(^1P_1)$ **7.4×10^{-2}**

spin-forbidden intensity increases by ~150× from Zn to Hg • nearly Z^5 trend

What the benchmark shows

1 SOC becomes observable

The $^1S_0 \rightarrow {}^3P_1$ transition is tiny for Zn but clearly visible for Hg.

2 A stringent 4c test

The rapidly growing intensity directly probes the relativistic spin–orbit description.

3 Current XC limitation

$^1P_1$ energies agree very well with experiment; $^3P_1$ is overestimated by ≈0.4–0.5 eV with the density-only XC approximation.

Relativity is not a small spectral correction here—it changes transition intensities qualitatively.

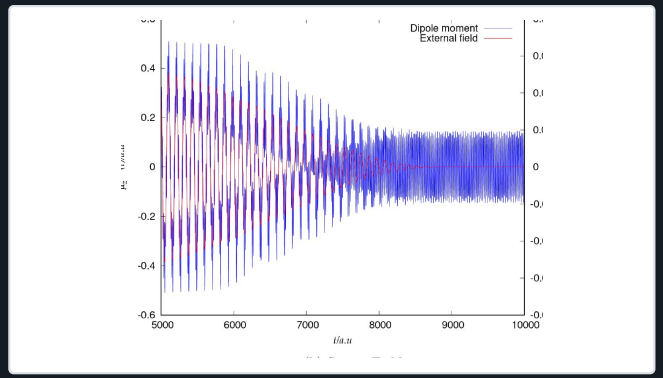
Take-home: spin–orbit coupling activates nominally forbidden transitions, and RT-TDDKS captures their rapid growth from Zn to Hg.

RT-TDDKS in the nonlinear regime: high-harmonic generation in H_2

STRONG FIELD

Strong-field polarization in the time domain

The induced dipole no longer follows the laser field adiabatically.



1.55 eV
carrier

1.02×10^{14}
W cm^{-2}

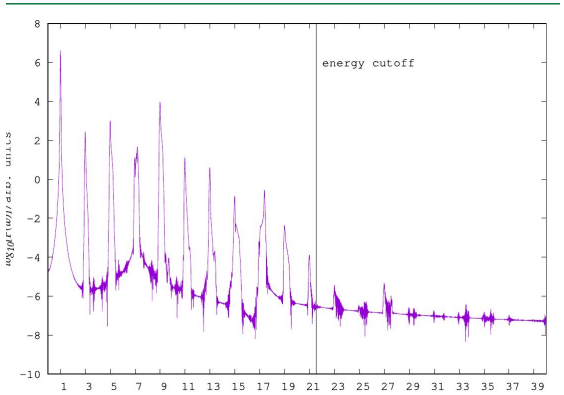
80 cycles
≈223 fs

$\mu(t)$

Fourier transform

High-harmonic spectrum

Fourier analysis of the laser-driven dipole reveals the nonlinear harmonic content.



Take-home: the same four-component real-time framework remains stable in intense fields and captures high-harmonic generation in H_2

RT-TDDKS in the nonlinear regime: high-harmonic generation in Au_2

HEAVY-ELEMENT TEST

From H_2 benchmark to heavy-element dynamics

Same intense pulse — now propagated all-electron in Au_2.

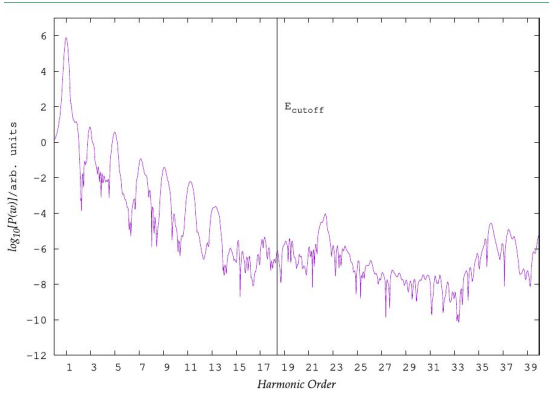
R_e = 4.67 a.u.

1.55 eV	1.02×10^{14}	80 cycles
carrier	W cm^{-2}	≈223 fs

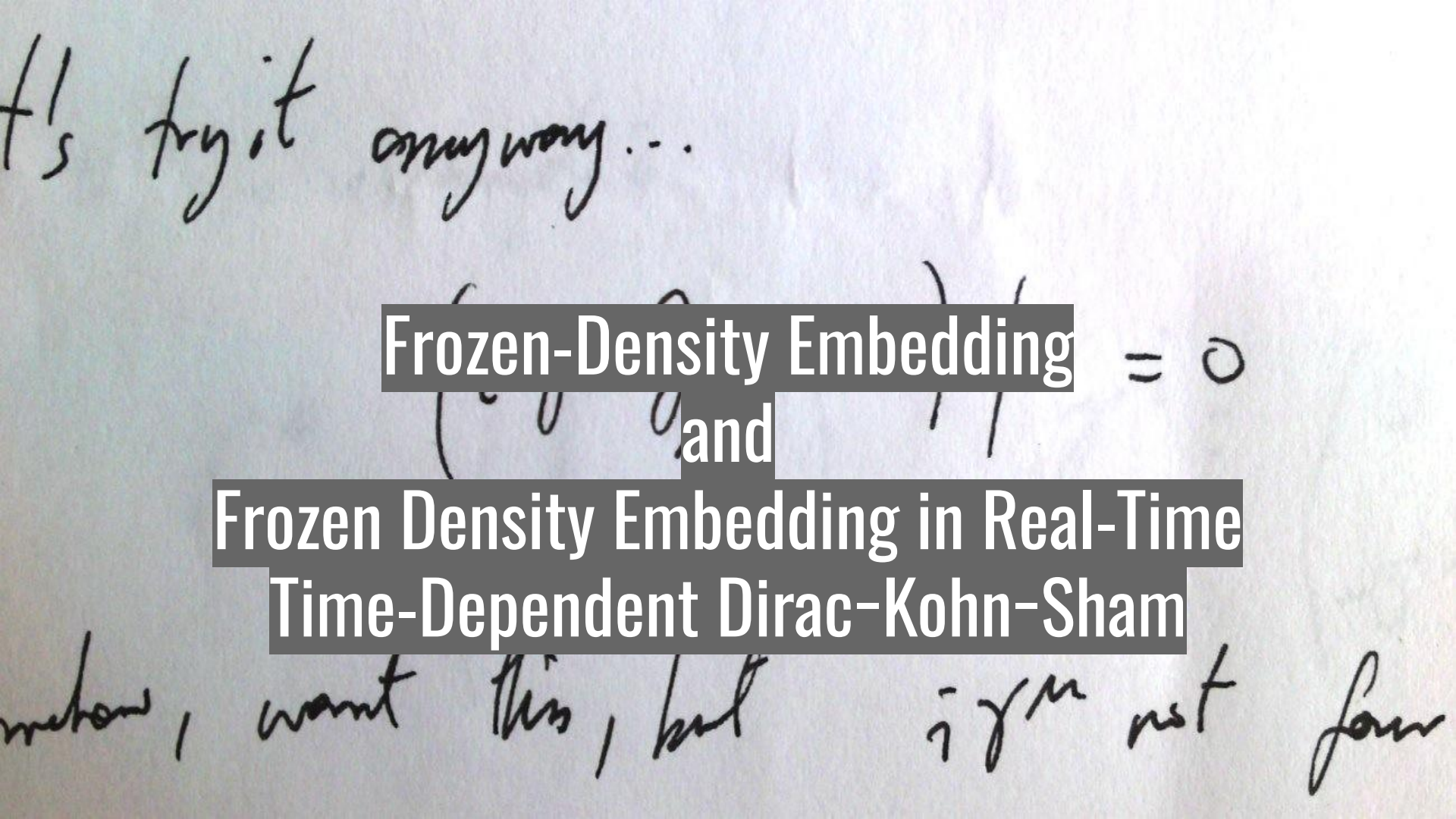
- All-electron 4c propagation remains numerically stable.
- **Relatively well-defined harmonics are obtained up to H_{13}.**

Au_2 high-harmonic spectrum

Fourier components of the laser-driven dipole; vertical line marks the semiclassical cutoff estimate.



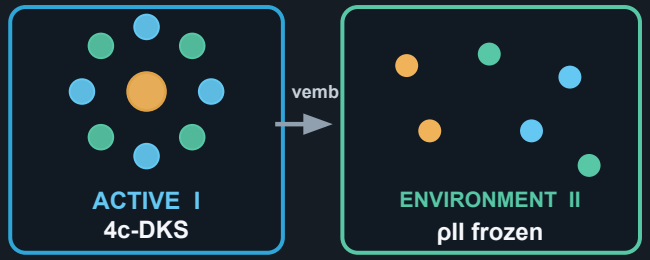
Take-home: Au_2 demonstrates stable four-component strong-field dynamics in a heavy-element molecule and extends HHG beyond light-system benchmarks.


Frozen-Density Embedding
and
Frozen Density Embedding in Real-Time
Time-Dependent Dirac-Kohn-Sham

Frozen-Density Embedding: 4c-DKS active subsystem in a quantum environment

THEORY

Partition the density, not the physics



$$\rho_{tot}(r) = \rho_I(r) + \rho_{II}(r)$$

Only the active density is optimized (or propagated); the environment enters explicitly through its frozen electron density.

uFDE → no mutual polarization of the frozen environment

Embedding potential added to the active Hamiltonian

$$[\,\hat{H}_{DKS}[\rho_I] + V_{emb}I[\rho_I,\rho_{II}]\,]\,C = S\,C\,\varepsilon$$

Vembl contains

- **Environment nuclei** — external nuclear potential
- **Frozen ρII** — Coulomb potential
- **Nonadditive XC** — δExc nadd / δρI
- **Nonadditive kinetic** — δTs nadd / δρI

The result is a one-electron operator that can be added directly to the four-component DKS matrix of subsystem I.

Keep full 4c-DKS accuracy for the heavy subsystem while the surrounding environment enters through a first-principles embedding potential.

PyBERTHA-Embed → PyBERTHA-Embed-RT: implementation strategy

IMPLEMENT

Python interoperability glues specialized engines together

BERTHA / PyBERTHA
4c-DKS active subsystem
G-spinors + density fitting

→

Shared grid + auxiliary fit
$\tilde{\rho}_I(r)$ on grid ↔ projected Vemb
three-center integrals reused

→

PyADF / PyEmbed / XCFun
environment density + Coulomb
nonadditive XC and kinetic terms

Ground-state DKS-in-DFT FDE

1. **Initialize environment once** — ρ_{II} and its electrostatic potential → global grid
2. **Map fitted active density** — $\tilde{\rho}_I$ → numerical grid
3. **Build embedding potential** — PyEmbed: Vemb[$\tilde{\rho}_I$,ρ_{II}] on the grid
4. **Project + add to DKS** — Vemb → auxiliary fit → J + K + Vemb → SCF

Real-time extension: same machinery at every time step

$$\rho_I(t) \rightarrow \mathrm{Vemb}(t) \rightarrow \mathrm{HDKS}(t)+\mathrm{Vemb}(t) \rightarrow \text{Magnus step}$$

A **ρ_{II} remains frozen** — but Vemb(t) changes because the active density changes.

B **Propagator unchanged** — The midpoint-Magnus / predictor-corrector scheme is used exactly as in PyBERTHA-RT.

Density fitting: $\tilde{V}^{emb}_{\mu\nu} = \Sigma_t I_{t,\mu\nu}\, c_t$ with $A\,c = g$ → avoids direct grid integration over principal G-spinor pair amplitudes.

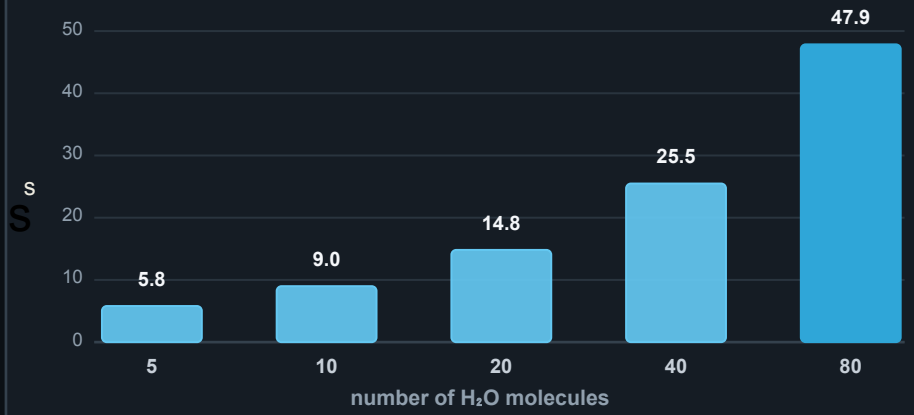
The same density-fitting / Python-interoperability strategy supports both SCF embedding and real-time embedding with minimal changes to the core BERTHA engines.

Ground-state FDE results: scalable environments and confinement

RESULTS

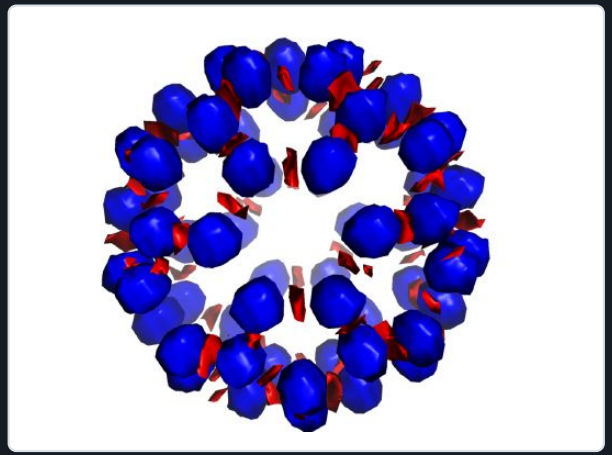
Au_4 in an expanding water environment

FDE work per SCF iteration grows approximately linearly with the frozen environment.



FDE overhead 2.2% → 16.1%

Confinement: Rn / Cn / Fl / Og inside C_{60}



Embedding potential around Rn@C_{60}

- The embedding potential is transferable across Rn, Cn, Fl and Og.
- A Rn-based spherical-average potential reproduces FDE HOMO–LUMO gaps of the other confined atoms with <4% error.

FDE adds realistic solvent/confinement effects while preserving the favorable density-fitted scaling of the BERTHA workflow.

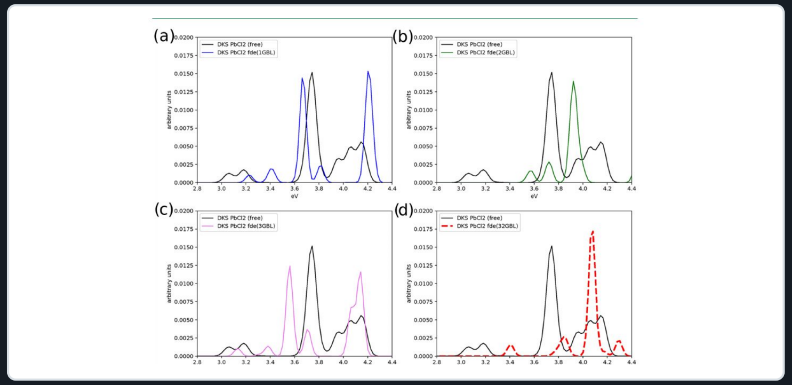
Real-time FDE results: solvent reshapes PbX_2 absorption spectra

RESULTS

Active subsystem: 4c-DKS $PbCl_2$ / PbI_2 • frozen environment: 1, 2, 3 or 32 γ-butyrolactone (GBL) molecules

stable & converged

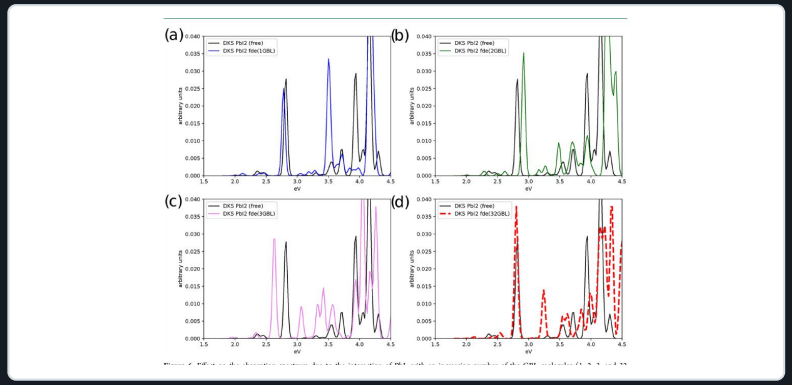
$PbCl_2$ + GBL



Lowest transition: blue shift vs free = 0.20 / 0.60 / 0.15 / 0.35 eV for 1 / 2 / 3 / 32 GBL.

Higher-energy peaks are progressively reshaped; the 32-GBL environment shows that outer solvation shells matter.

PbI_2 + GBL



First excitation ≈2.0 eV (vs ≈3.0 eV for $PbCl_2$); solvation produces an overall shift of about 0.3 eV.

Above 3 eV, both peak positions and intensities depend strongly on the solvent environment.

$PbCl_2$: 5.4 s/step free → 9.9 s/step with 32 GBL

uFDE limitation: excitations delocalized over solute + solvent require a larger active region or a coupled-FDE treatment.

The solvent is not a small correction: first and outer solvation shells can shift, split and redistribute relativistic absorption features.

Table of contents

Conclusions

01 GPU data locality

Improve data locality on the GPU to reduce transfers and communication overhead.

02 Multi-GPU scalability

Combine multi-GPU execution with our open-ended MPI implementation; evaluate ELPA for distributed eigensolvers.

03 Density-only → collinear → non-collinear

Extends BERTHA to open-shell/SOC systems and provides the foundation for explicit spin dynamics in RT-TDDKS.

04 Density fitting for exact exchange

Ongoing: density fitting for Hartree–Fock exact exchange, targeting efficient four-component hybrid functionals.

GitHub *github.com/BERTHA-4c-DKS/*

References

Full author lists • papers used in this presentation

2014 — **Full Parallel Implementation of an All-Electron Four-Component Dirac–Kohn–Sham Program**
Sergio Rampino; Leonardo Belpassi; Francesco Tarantelli; Loriano Storchi
J. Chem. Theory Comput. 10 (9), 3766–3776 • DOI: 10.1021/ct500498m

2019 — **The Chemical Bond and s–d Hybridization in Coinage Metal(I) Cyanides**
Matteo De Santis; Sergio Rampino; Loriano Storchi; Leonardo Belpassi; Francesco Tarantelli
Inorg. Chem. 58 (17), 11716–11729 • DOI: 10.1021/acs.inorgchem.9b01694

2020 — **BERTHA: Implementation of a Four-Component Dirac–Kohn–Sham Relativistic Framework**
Leonardo Belpassi; Matteo De Santis; Harry M. Quiney; Francesco Tarantelli; Loriano Storchi
J. Chem. Phys. 152, 164118 • DOI: 10.1063/5.0002831

2020 — **PyBERTHART: A Relativistic Real-Time Four-Component TDDFT Implementation Using Prototyping Techniques Based on Python**
Matteo De Santis; Loriano Storchi; Leonardo Belpassi; Harry M. Quiney; Francesco Tarantelli
J. Chem. Theory Comput. 16 (4), 2410–2429 • DOI: 10.1021/acs.jctc.0c00053

2022 — **Frozen-Density Embedding for Including Environmental Effects in the Dirac–Kohn–Sham Theory: An Implementation Based on Density Fitting and Prototyping Techniques**
Matteo De Santis; Diego Sorbelli; Valérie Vallet; André Severo Pereira Gomes; Loriano Storchi; Leonardo Belpassi
J. Chem. Theory Comput. 18 (10), 5992–6009 • DOI: 10.1021/acs.jctc.2c00499

2025 — **Acceleration of the Relativistic Dirac–Kohn–Sham Method with GPU: A Pre-Exascale Implementation of BERTHA and PyBERTHA**
Loriano Storchi; Laura Bellentani; Jeff Hammond; Sergio Orlandini; Leonardo Pacifici; Nicoló Antonini; Leonardo Belpassi
J. Chem. Theory Comput. 21 (7), 3460–3475 • DOI: 10.1021/acs.jctc.4c01759

2025 — **Automatic Generation of Density-Fitting Auxiliary Basis Sets for All-Electron Dirac–Kohn–Sham Calculations**
Nicoló Antonini; Enrico Ronca; Loriano Storchi; Leonardo Belpassi
J. Phys. Chem. A 129 (30), 6930–6941 • DOI: 10.1021/acs.jpca.5c02772

2025 — **Transferable and Transparent Energy Decomposition-Based Machine Learning Models for Computing Accurate Reaction Energetics**
Carlos R. Jacinto-Mejía; Loriano Storchi; Giovanni Bistoni
J. Chem. Theory Comput. 21 (21), 10853–10862 • DOI: 10.1021/acs.jctc.5c01184

2026 — **Environmental Effects via Frozen Density Embedding in Real-Time Time-Dependent Dirac–Kohn–Sham Theory: Solvation of Lead Halides**
Matteo De Santis; Edoardo Mosconi; Leonardo Pacifici; Valérie Vallet; André Severo Pereira Gomes; Loriano Storchi; Leonardo Belpassi
J. Chem. Theory Comput. 22 (5), 2282–2298 • DOI: 10.1021/acs.jctc.5c01980

2026 — **A Unified Formalism for Collinear and Non-Collinear Approaches in the Four-Component Dirac–Kohn–Sham Theory Based on G-Spinors**
Giulia Gamboni; Loriano Storchi; Paola Belanzoni; Leonardo Belpassi
arXiv:2606.31482 [physics.chem-ph] (preprint) • DOI: 10.48550/arXiv.2606.31482

Duplicate uploads consolidated: MPI = FullMPI; OPENMP = ReviewAll.

t's try it anyway...

$$\left(i\gamma^{\mu}\partial_{\mu}\right)\psi = 0$$

BACKUP

...

motion, want this, but $i\gamma^{\mu}$ not four

Introduction

Finally the matrix representation of the DKS operator in the G-spinor basis is (**Complex arithmetic**):

$$\begin{bmatrix} \boldsymbol{v}^{(\mathrm{LL})} + \boldsymbol{J}^{(\mathrm{LL})} + \boldsymbol{K}^{(\mathrm{LL})} + mc^2\boldsymbol{S}^{(\mathrm{LL})} & c\boldsymbol{\Pi}^{(\mathrm{LS})} \\ c\boldsymbol{\Pi}^{(\mathrm{SL})} & \boldsymbol{v}^{(\mathrm{SS})} + \boldsymbol{J}^{(\mathrm{SS})} + \boldsymbol{K}^{(\mathrm{SS})} - mc^2\boldsymbol{S}^{(\mathrm{SS})} \end{bmatrix}$$

Coulomb potential

Exchange-correlation potential

Nuclear potential

Kinetic energy operator matrix

Overlap matrix

ML & CHEMO: Transparent DFT energy corrections

Energy-decomposition descriptors + local linear models + a Random Forest selector in collaboration with G. Bistoni

1 / DESCRIPTORS

DFT energy decomposition

$aA + bB \rightarrow cC + dD$

ΔT_e ΔV_{NN} ΔV_{eN} ΔE_X

ΔE_C ΔE_J ΔE_{disp}

Physically meaningful inputs from the same low-cost DFT calculation

2 / LR LIBRARY

Interpretable energy corrections

$\Delta E_{LR} = w^T \Delta E + b$

FULL	SMALL
LARGE	BARRIER
INTER	INTRA

Global + specialized LR models trained on CCSD(T)/CBS data (i.e., the label)

3 / RF SELECTOR

Choose the best local correction

RF vote

vote > 70%?

specialized LR

fallback LR(FULL)

Training: GMTKN55 + LP14 | PBE/PBE0 | MINIX → def2-QZVP

JCTC 2025, 21, 10853-10862

ML & CHEMO: Accuracy and transferability

GENERAL BENCHMARK: GMTKN55 + LP14 | MAPE [%]

	DFT	LR(FULL)	RF/LR
PBE / MINIX	207.3%	144.2%	84.4%
PBE0 / def2-QZVP	32.2%	30.3%	25.3%

−122.9 percentage points

Improves even an already accurate DFT/basis-set combination

Largest gains occur for inexpensive calculations, where basis-set incompleteness is greatest.

OOD / WCCR10

70.7 → 52.7 → 47.6

PBE/MINIX; no transition metals in training

INTERPRETABLE

ΔE_X, ΔE_C, ΔT_e

most informative RF features

PRACTICAL

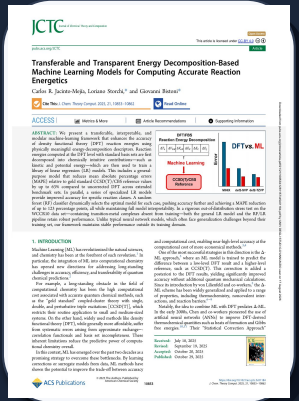
Low-cost DFT + lightweight ML

modular and expandable

Take-home: transparent local models + RF routing improve accuracy without another QM calculation.

Publication & Ongoing Development

PUBLISHED WORK


JCTC • 2025

Transferable and Transparent Energy-Decomposition-Based Machine Learning Models for Accurate Reaction Energetics

C. R. Jacinto-Mejía • L. Storchi • G. Bistoni

J. Chem. Theory Comput. 2025, 21, 10853–10862

DOI
10.1021/acs.jctc.5c01184

Transparent • Transferable • Modular

WORK IN PROGRESS

Extending the RF/LR framework

01 **NEW DATA**
Larger and more diverse reaction data

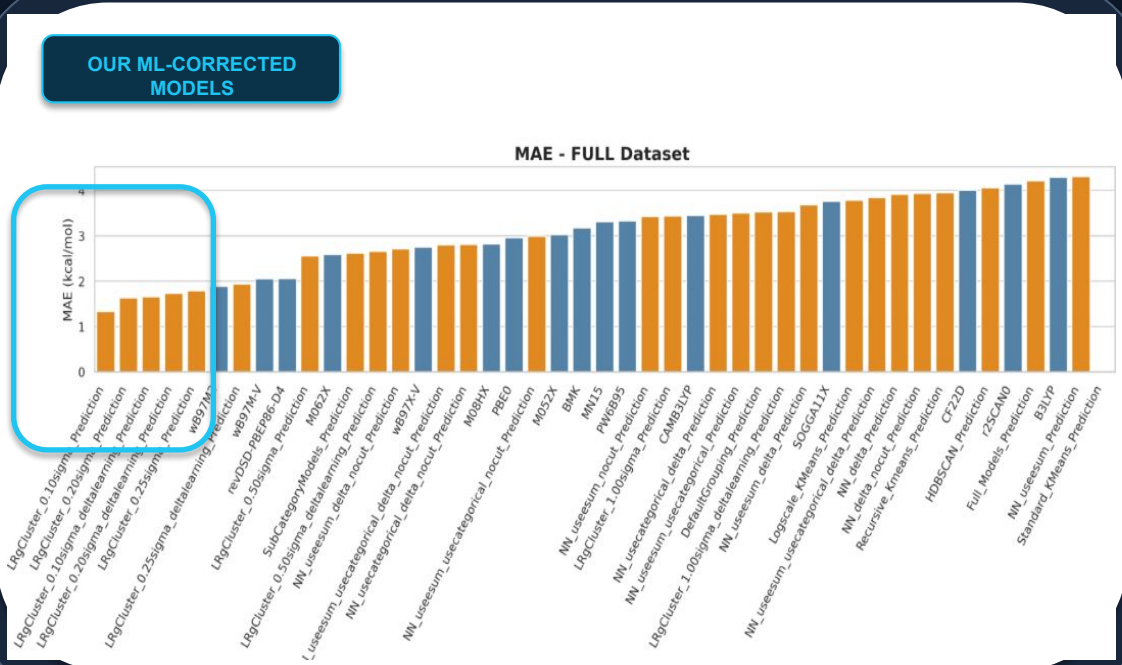
02 **METHOD**
Methodological refinements to the correction and model-selection workflow

03 **VALIDATION**
Broader transferability and robustness tests

We are currently updating the approach with new data and methodological improvements - stay tuned.

Low-cost DFT + ML outperforms many higher-cost methods

PBE/TZVP + the RF/LR correction framework • full-dataset MAE • lower is better



01 LOW-COST STARTING POINT

PBE / TZVP
The expensive part is still a standard DFT calculation.

02 ML CORRECTION

RF routing + local LR models
The same lightweight framework shown in the previous slides.

03 BENCHMARK RESULT

The best corrected predictions achieve lower MAE than many substantially more expensive methods.

Take-home: after training, the per-system cost remains essentially PBE/TZVP + lightweight ML, while the benchmark accuracy is competitive with — and often better than — much more expensive methods.