Lightspeed: A domain-specific computational environment for electronic structure and quantum dynamics

Robert M. Parrish,[1] Xin Li,[1,2] Jason Ford,[1,2] Ruben Guerrero,[1,2] and Todd J. Martínez[1,2]

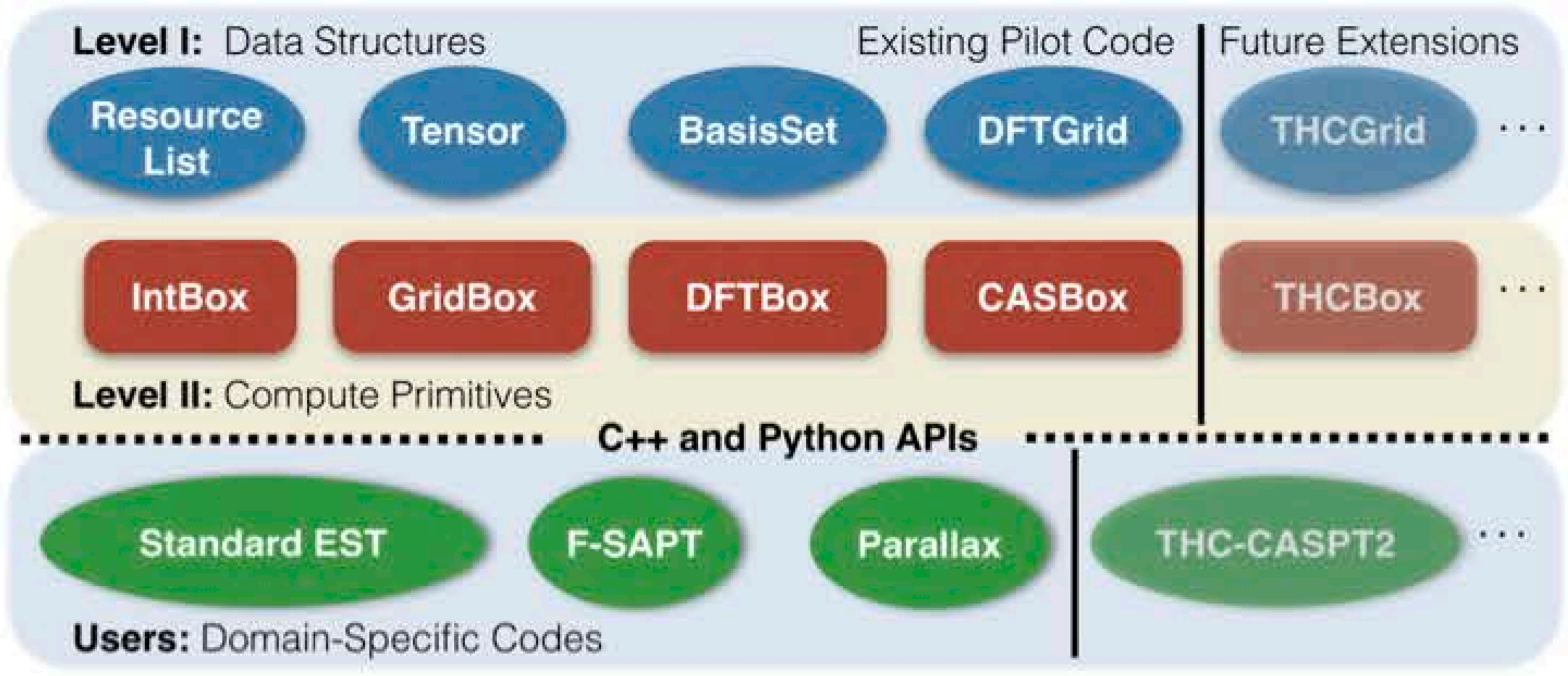
[1]SLAC National Accelerator Laboratory
[2]Department of Chemistry and the PULSE Institute
Stanford University

Overview

We discuss the architecture and developments in "Lightspeed," a C++/CUDA/Python library designed to enhance the ease of development of production-level electronic structure methods. Lightspeed does not do electronic structure theory –Lightspeed helps **you** do electronic structure theory. To that end, Lightspeed provides simple C++ and Python interfaces to such primitive operations as molecular integrals, Coulomb (*J*) and exchange (*K*) matrix builds, density functional theory (DFT) potentials, complete active space matrix-vector products, and gradients for all of the above. Additional utility libraries provide access to tensor operations, large-scale optimization algorithms (e.g., DIIS and Davidson methods), and several types of electronic structure analyses (e.g., orbital localization, grid properties, etc). These operations are implemented for CPU and/or GPU hardware, depending on the resources available at runtime. Moreover, Lightspeed enables rapid development of new electronic structure methods, as demonstrated herein with the "Parallax" fragment-based model, which is capable of single point and gradient evaluations for systems with >1 million atoms, and a new implementation of the "F-SAPT" intermolecular interaction analysis which can provide detailed analysis of full drug-protein interactions.

Lightspeed

High-Level Library Layout:



Code Snippet: GPU-Based *J* Builds:

```python
import lightspeed as ls # The Lightspeed module
resources = ls.ResourceList.build() # Use all available CPU/GPU resources
molecule = ls.Molecule.from_xyz_file('geom.xyz') # Read ./geom.xyz and build Molecule
basis = ls.BasisSet.from_gbs_file(molecule, 'cc-pvdz') # Construct cc-pVDZ basis
pairlist = ls.PairList.build_schwarz(basis, basis, 1.0E-14) # Construct PairList
S = ls.IntBox.compute_overlap(resources, pairlist) # Compute the overlap matrix as Tensor
J = ls.IntBox.compute_coulomb( # Compute the Coulomb matrix as Tensor
    resources,                 # The resources to use
    ls.Ewald.coulomb(),        # The standard Coulomb interaction operator
    pairlist,                  # The pairlist on the bra (12|
    pairlist,                  # The pairlist on the ket |34)
    S,                         # The input density matrix (S used for demo only)
    1.0E-6,                    # The double-precision cutoff
    1.0E-14)                   # The single-precision cutoff
print J
```

Code Snippet: Easy Cube Files:

```python
cube = ls.CubicProp(options={ # Build a CubicProp helper object
    'resources' : ref.resources,
    'molecule' : ref.molecule,
    'basis' : ref.basis,
    'pairlist' : ref.pairlist})
cube.save_density_cube( # Save a density cube corresponding to Dtot in rho.cube
    'rho.cube',
    Dtot)
cube.save_orbital_cubes( # Save 2 orbital cubes in psi0.cube and psi1.cube
    'psi',
    Cgap)
cube.save_esp_cube( # Save an ESP cube corresponding to Dtot in esp.cube
     'esp.cube',
     Dtot)
```

Code Snippet: IBO Localized Orbitals:

```python
ibo = ls.IBO(options={ # Build IAOs
    'resources' : ref.resources,
    'molecule' : ref.molecule,
    'basis' : ref.basis,
    'minbasis' : ref.minbasis,
    'Cref' : ref.tensors['Cocc']})
Focc = ls.Tensor.array(np.diag(ref.tensors['eps_occ'])) # The occ-occ Fock matrix
U, L, F = ibo.localize(ref.tensors['Cocc'],Focc) # Find localized occupied orbitals
Q = ibo.orbital_atomic_charges(L) # Local orbital atomic charges
```

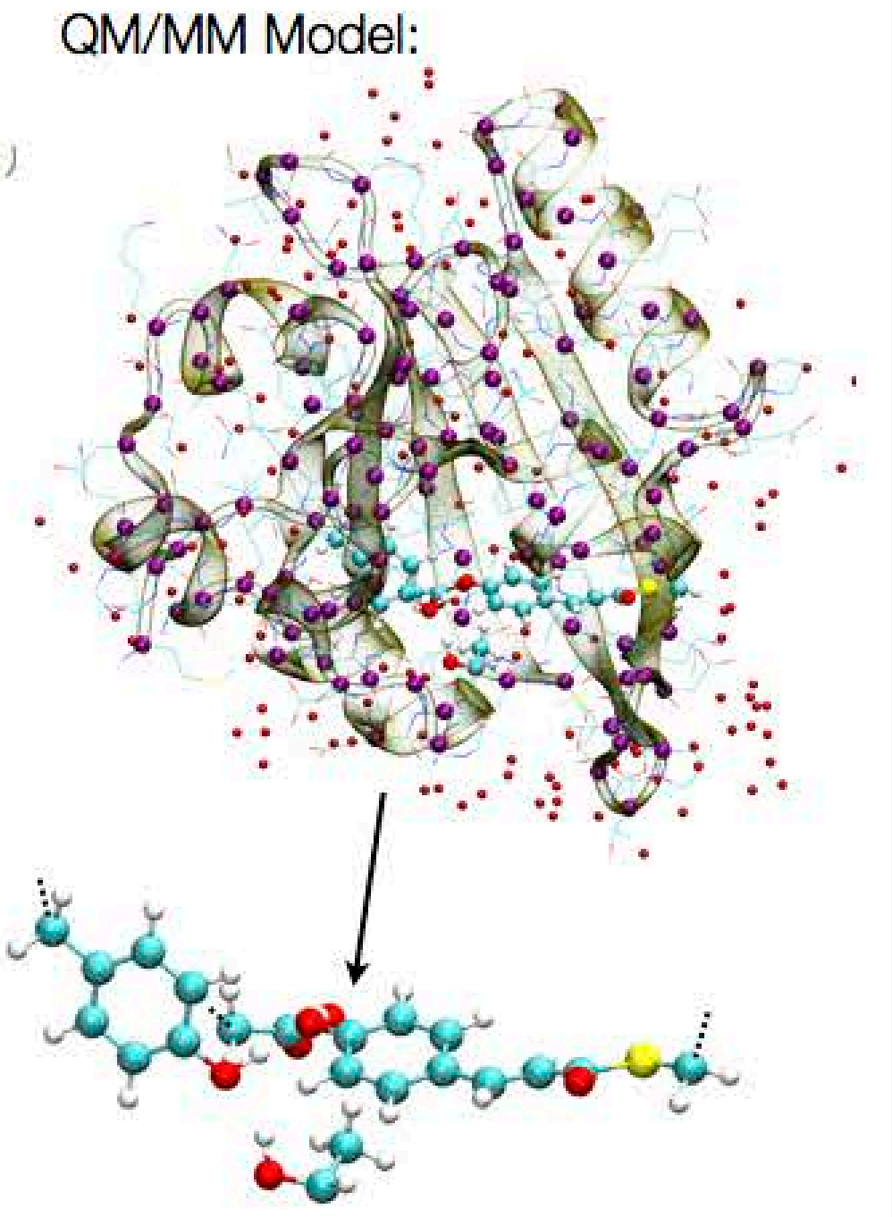
R.M. Parrish, X. Li, J.E. Ford, R.D. Guerrero, and T.J. Martínez, *in preparation*.

Integrative Example

FOMO-CASCI QM/MM Dynamics with Psidewinder:

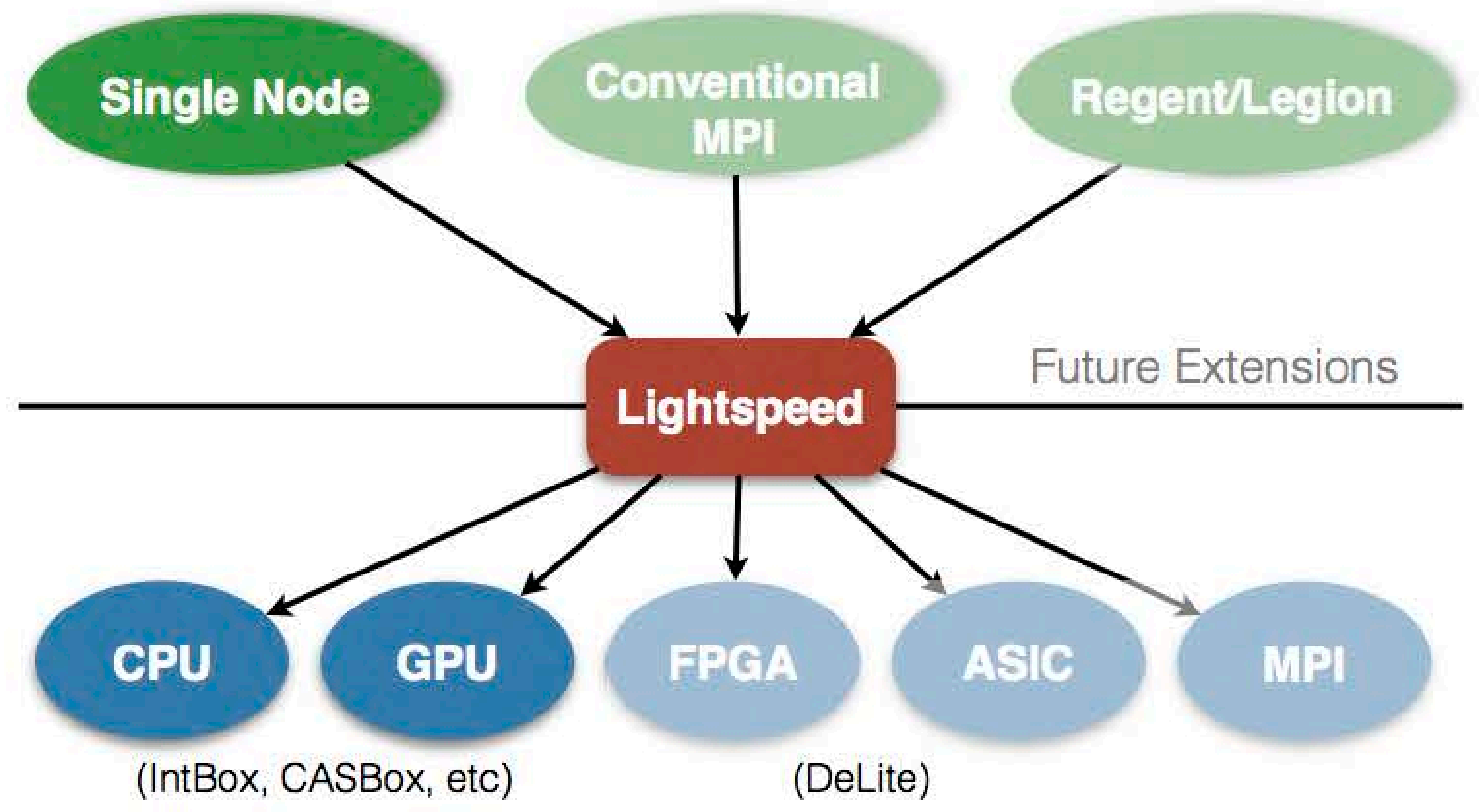
```python
import lightspeed as ls # The lightspeed module
import psiw # The "psidewinder" lightweight electronic structure module
import md # The lightweight adiabatic MD code
# CPU and/or GPU resources
resources = ls.ResourceList.build()
# OpenMM-based QM/MM (Mechanical + Coulomb embedding w/ link H atoms)
qmmm = psiw.QMMM.from_prmtop(
    prmtopfile='pyp.prmtop',
    inpcrdfile='pyp.rst',
    qmindsfile='pyp.qm',
    charge=-1.0,
    )
# Geometry manages all external environment considerations
geom = psiw.Geometry.build(
    resources=resources,
    qmmm=qmmm,
    basisname='6-31g',
    )
# FON-RHF (4 active electrons in 3 fractional orbitals)
ref = psiw.RHF.from_options(
    geometry=geom,
    g_convergence=1.0E-6,
    fomo=True,
    fomo_method='gaussian',
    fomo_temp=0.2,
    fomo_nocc=107,
    fomo_nact=3,
    )
ref.compute_energy()
# FOMO-CASCI (3 singlet states)
casci = psiw.CASCI.from_options(
    reference=ref,
    nocc=107,
    nact=3,
    nalpha=2,
    nbeta=2,
    S_inds=[0],
    S_nstates=[3],
    )
casci.compute_energy()
# Level-of-theory (LOT) manages guesses, MOM, etc
lot = md.CASCI_LOT.from_options(
    casci=casci,
    print_level=0,
    rhf_guess=True,
    rhf_mom=True,
    )
# Velocity Verlet integrator (other integrators have thermostats)
vv = md.VV.from_options(
    dt=20.0, # 20 au timestep
    )
# Get masses and inital momenta from table/rst file
masses = md.compute_masses(qmmm.molecule)
momenta = md.momenta_from_rst_file(
    filename='pyp.rst',
    masses=masses)
# Make an XYZ file entry for each MD frame (could also use DCD or other reporter)
xyz_reporter = md.XYZReporter.from_options(
    interval=1,
    filename='adiabatic.xyz')
# Make entries in a .npz file of scalar quantities.
npz_reporter = md.NPZReporter.from_options(
    interval=1,
    filename='adiabatic.npz',
    state_energies=True)
# Run 200 steps of adiabatic MD on S1
aimd = md.AIMD.from_options(
    lot=lot,
    integrator=vv,
    target_S=0,
    target_index=1,
    masses=masses,
    momenta=momenta,
    state_tracking='adiabatic',
    reporters=[xyz_reporter, npz_reporter])
aimd.initialize()
aimd.run(200)
aimd.finalize()
```

QM/MM Model:



Hiding Performant Code/Exposing Multilevel Parallelism

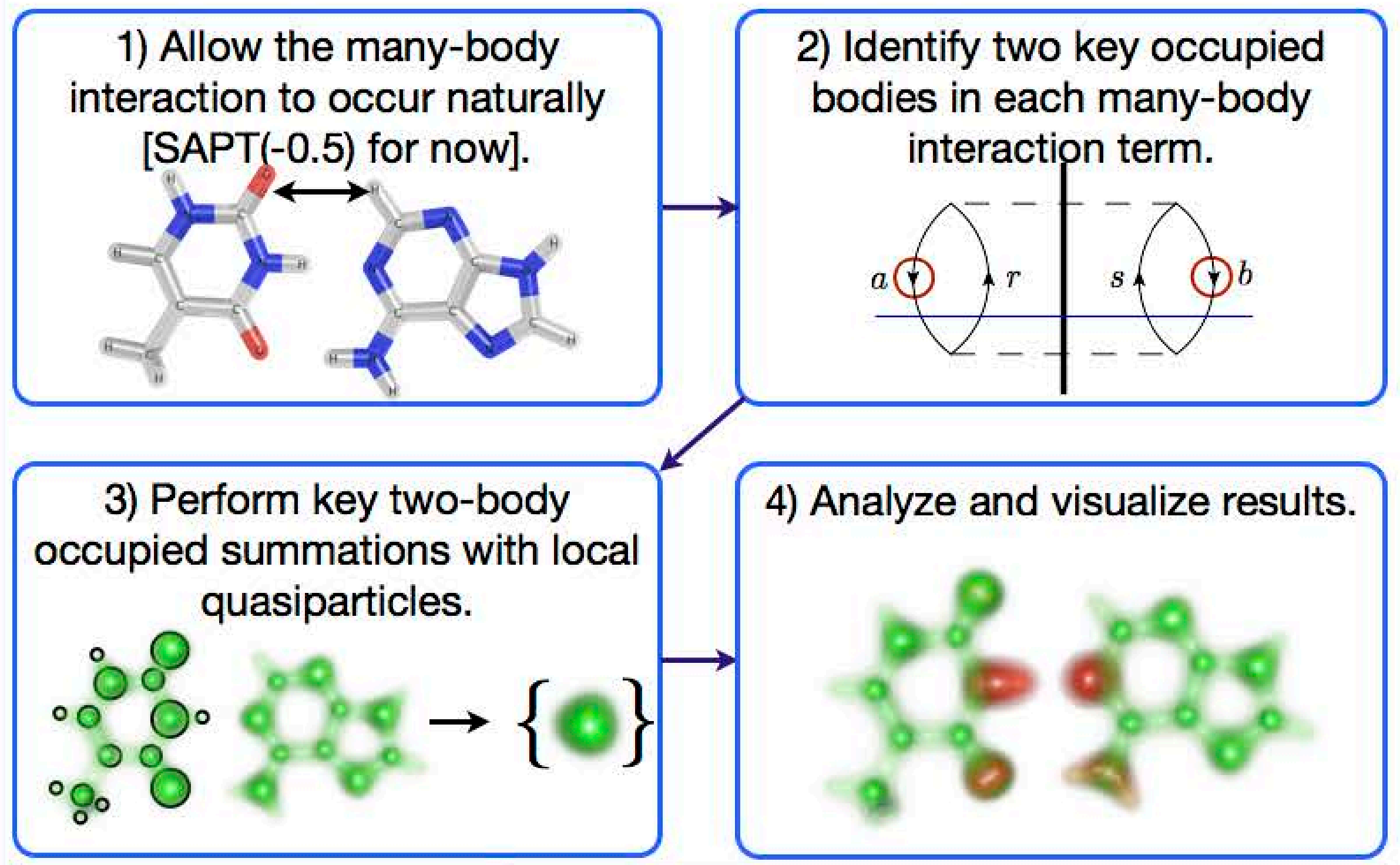
User Implementation Model:



Library Implementation Model:

Lightspeed Application: Large-Scale Functional-Group SAPT

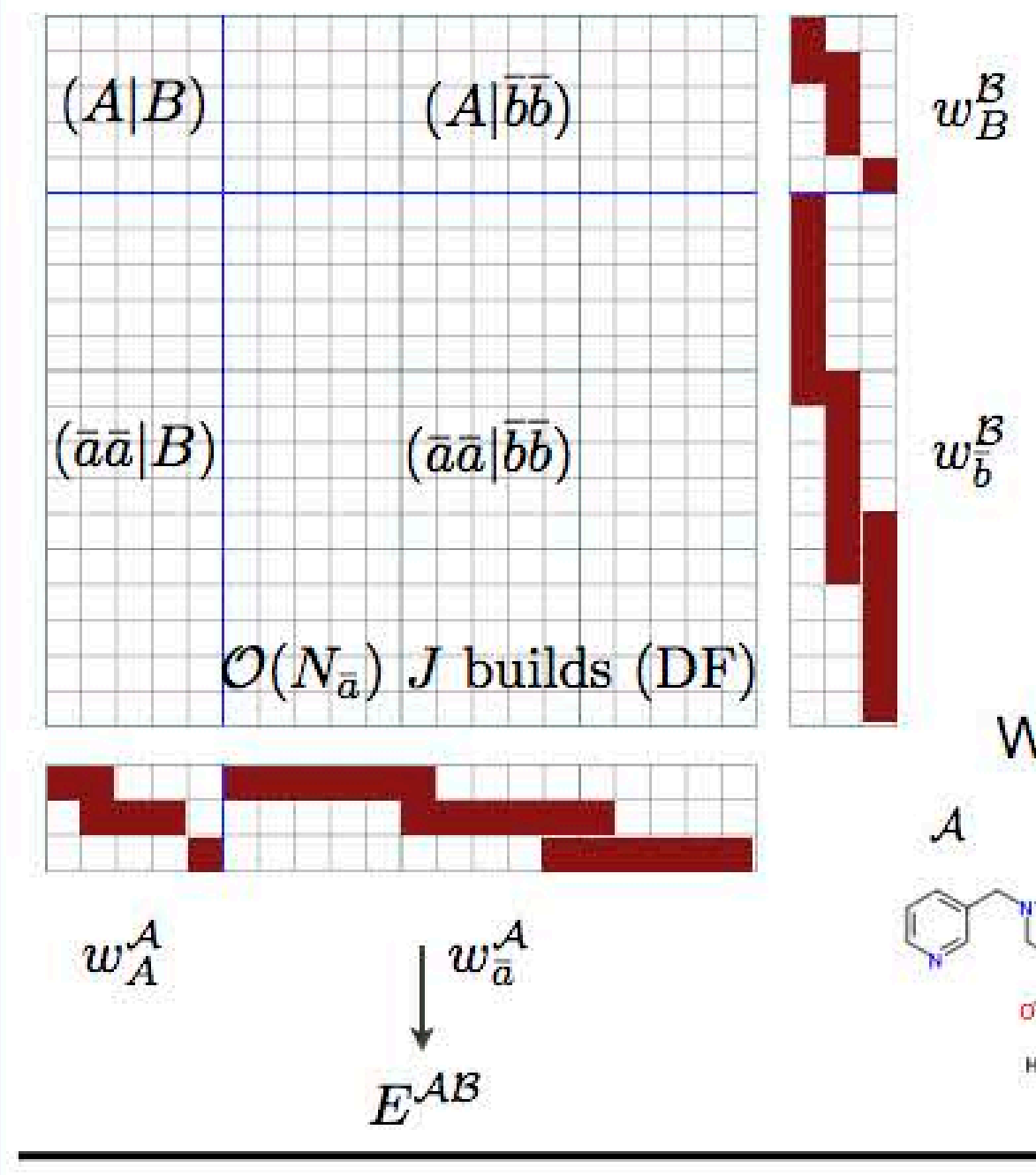
Basics of F-SAPT:


1) Allow the many-body interaction to occur naturally [SAPT(-0.5) for now].
2) Identify two key occupied bodies in each many-body interaction term.
a
r
s
b
3) Perform key two-body occupied summations with local quasiparticles.
4) Analyze and visualize results.

Computational Formulation:

~1300 lines of Lightspeed!

Old-Style F-SAPT (Electrostatics):


$(A|B)$
$(A|\bar{b}\bar{b})$
$(\bar{a}\bar{a}|B)$
$(\bar{a}\bar{a}|\bar{b}\bar{b})$
$\mathcal{O}(N_{\bar{a}})$ J builds (DF)
$w_B^{\mathcal{B}}$
$w_{\bar{b}}^{\mathcal{B}}$
$w_A^{\mathcal{A}}$
$w_{\bar{a}}^{\mathcal{A}}$
$E^{\mathcal{AB}}$

New-Style F-SAPT (Electrostatics):

$$Z_A^{\mathcal{A}} = w_A^{\mathcal{A}} Z_A, D_{pq}^{\mathcal{A}} = C_{p\bar{a}} w_{\bar{a}}^{\mathcal{A}} C_{q\bar{a}}$$

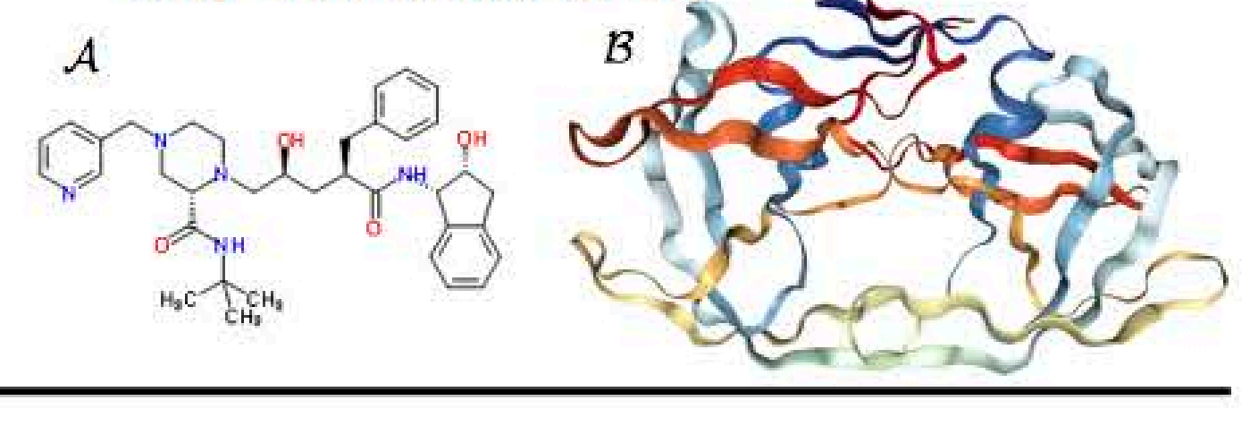
$$V_B^{\mathcal{A}} = Z_A^{\mathcal{A}}(A|B) + D_{pq}^{\mathcal{A}}(pq|B)$$

$$J_{rs}^{\mathcal{A}} = Z_A^{\mathcal{A}}(A|rs) + D_{pq}^{\mathcal{A}}(pq|rs)$$

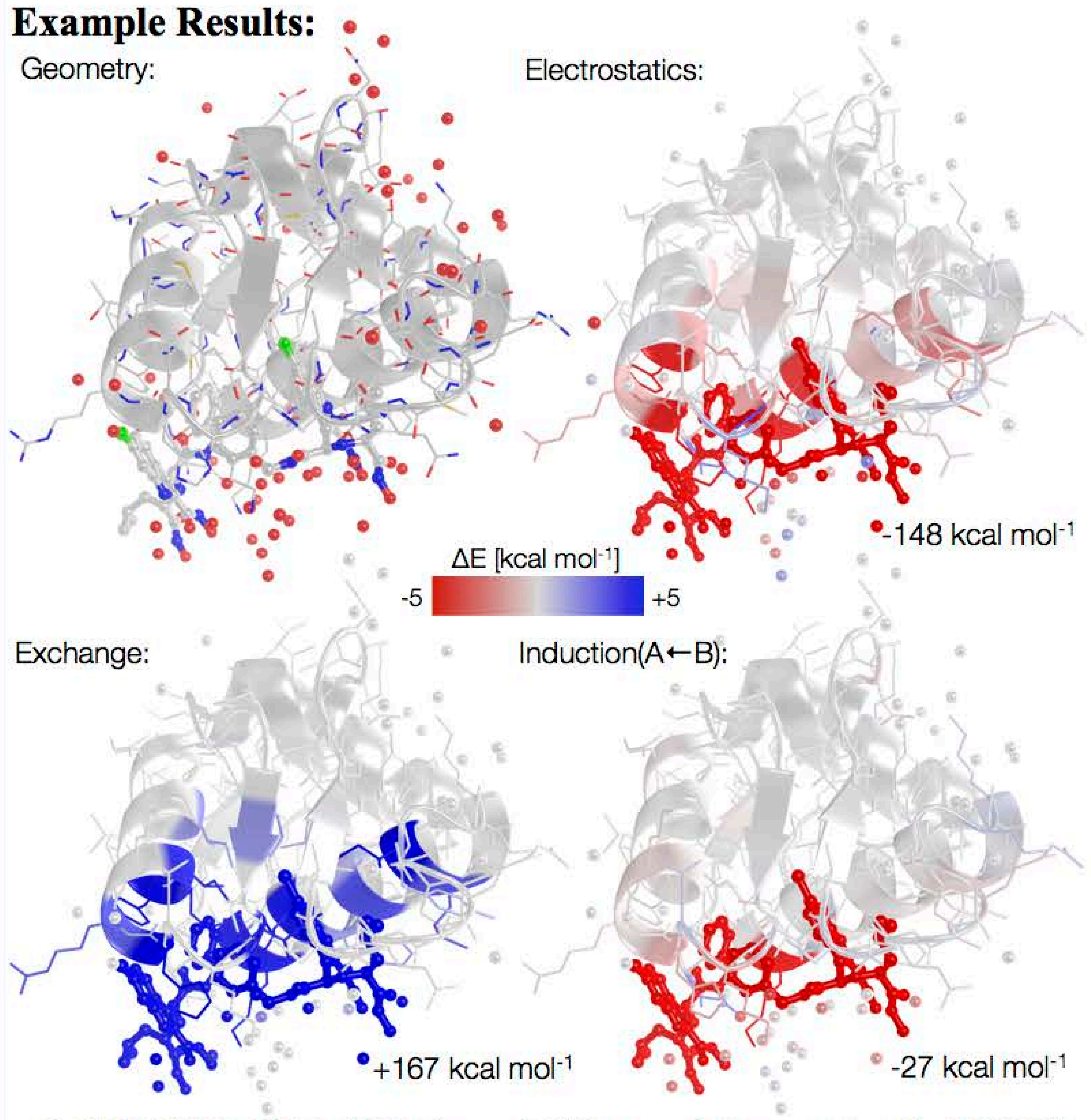
$$E^{\mathcal{AB}} = V_B^{\mathcal{A}} Z_B^{\mathcal{B}} + J_{rs}^{\mathcal{A}} D_{rs}^{\mathcal{B}}$$

$\mathcal{O}(N_{\mathcal{A}})$ J builds

Why drive from A to B?



Example Results:



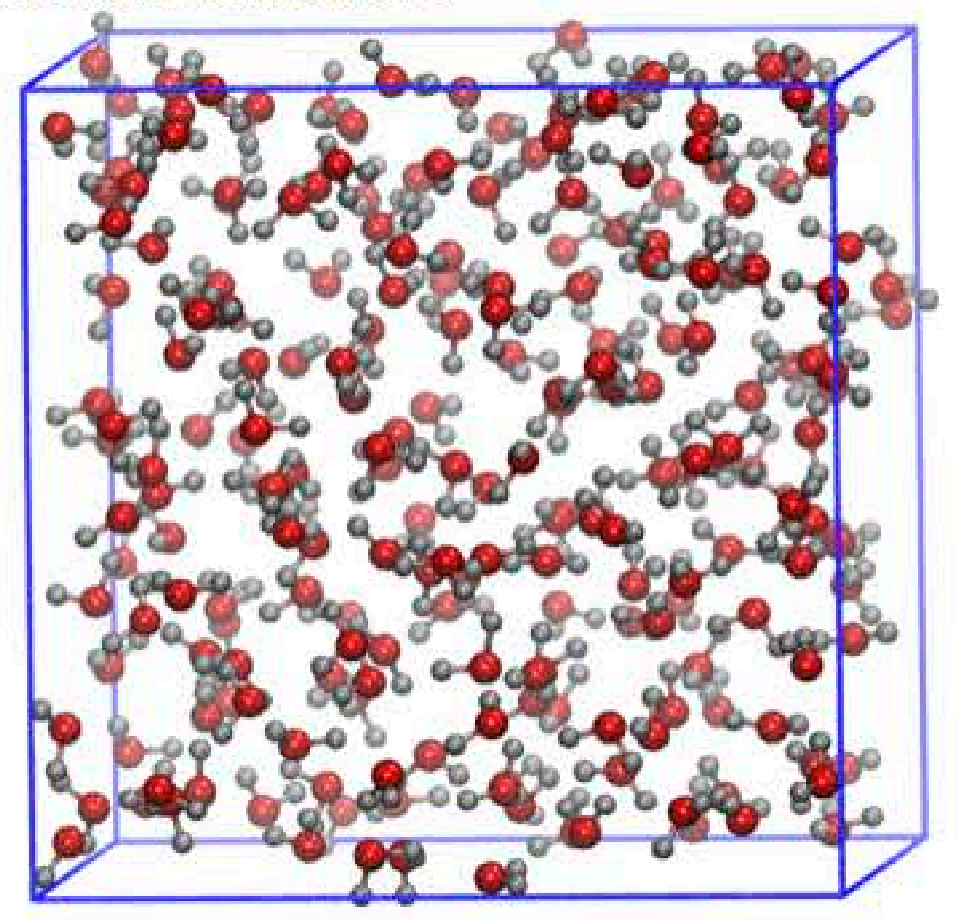
F-SAPT(-0.5)/6-31G* : 1831 atoms, 17809 nao, ~17 hours wall on 8x GTX 970.

B. Jeziorski, R. Moszyński, and K. Szalewicz, *Chem. Rev.* **94**, 1887 (1994).
R.M. Parrish and C.D. Sherrill, *J. Chem, Phys.* **141**, 044115 (2014).
R.M. Parrish, K.C. Thompson and T.J. Martínez, *J. Chem. Theory Comput.* **14**, 1737 (2018).

Primarily developed using pilot version of Lightspeed prior to SciDAC start.

Parallax: Large-Scale SCF-in-SCF

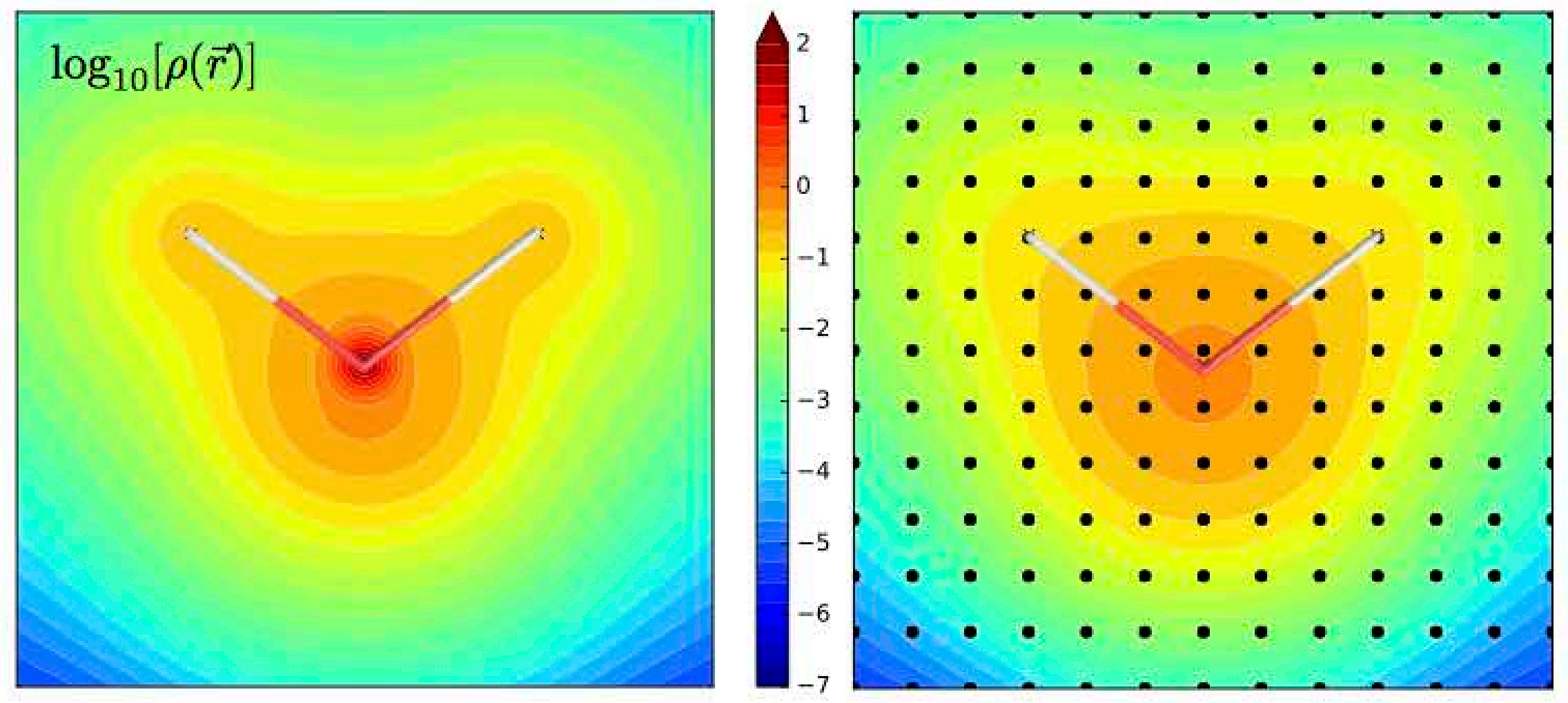
Parallax Model:



$$E \equiv \sum_A E_{\mathrm{A}}^{\mathrm{SCF}} + \frac{1}{2}\sum_A \sum_{B'} E_{\mathrm{AB'}}^{\mathrm{Coulomb}} + \frac{1}{2}\sum_A \sum_{B'} E_{AB'}^{6-12}$$

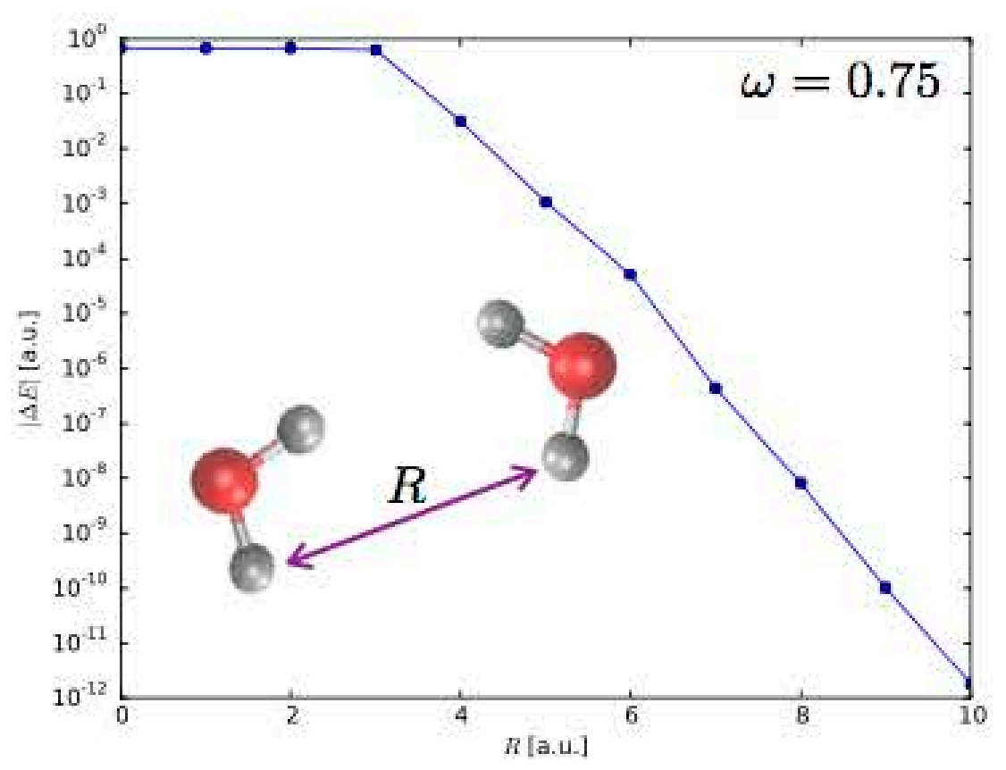
PME/FTC/QEM Blurring Approach:

Raw Density: Gaussian-Smoothed Density:

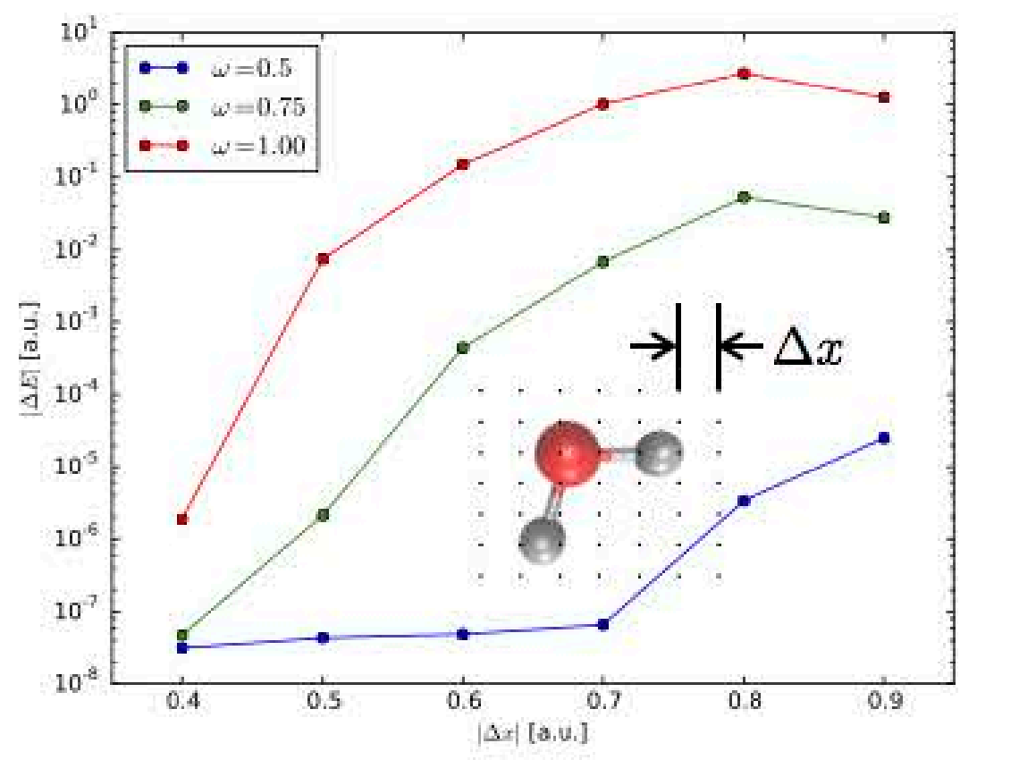


$$\tilde{\rho}(\vec{r}) \overset{\mathrm{FFT}}{\rightarrow} \hat{\rho}(\vec{k}) \overset{4\pi/k^2}{\rightarrow} \hat{Z}(\vec{k}) \overset{\mathrm{IFFT}}{\rightarrow} \tilde{Z}(\vec{r})$$

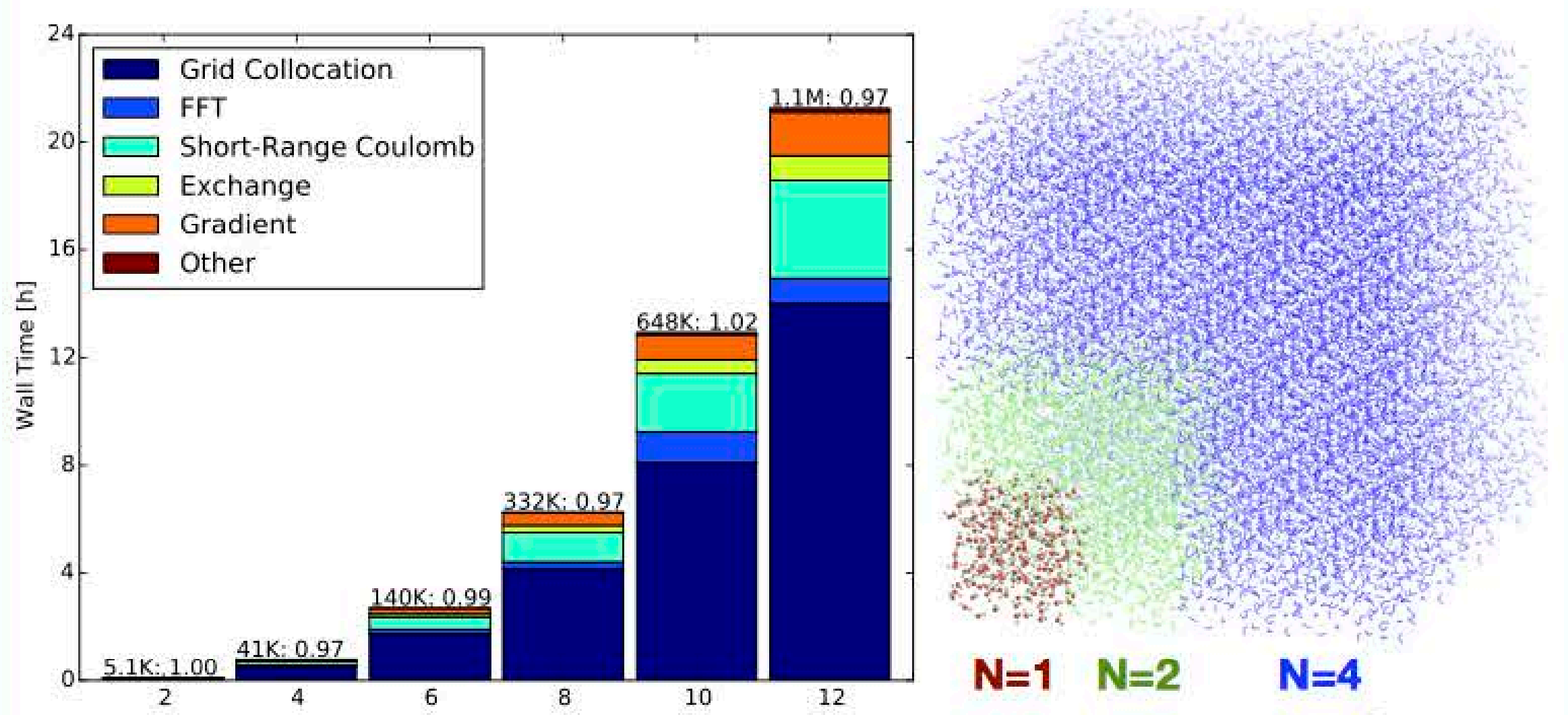
Short-Range Cutoff Distance:



Fourier Grid Spacing:



Timings: RHF/STO-3G Timings on Water Boxes (10 SCF Iterations + Gradient)



$$N_{\mathrm{atom}} = 648N^3$$

Acknowledgments

T. Darden, D. York, and L. Pedersen, *J. Chem. Phys.*, **98**, 10089 (1993).
L. Füsti-Molnár and Peter Pulay, *J. Chem. Phys.*, **117**, 7827 (2002).
C.-M. Chang, Y. Shao, and J. Kong, *J. Chem. Phys.*, **136**, 114112 (2012).