1. Installation

2. Lua in Siesta
   - MPI and Lua
   - What can you do?
   - Tutorial

3. sisl
   - Tutorial

Installation

- Install sisl, `conda install sisl` or `pip install sisl`
- Install flos, `github.com/siesta-project/flos`
  Download, and extract, then set:

  ```
  export LUA_PATH="<path>/flos/?.lua;<path>/flos/?/init.lua;$LUA_PATH"
  ```

Lua in Siesta

- New-comers (devs) to SIESTA have a *very* large barrier on entry to the code

Lua in Siesta

- New-comers (devs) to SIESTA have a *very* large barrier on entry to the code
- Implementations of different molecular dynamics routines are *very* time consuming given the importance of prior knowledge to SIESTA

Lua in Siesta

- New-comers (devs) to SIESTA have a *very* large barrier on entry to the code
- Implementations of different molecular dynamics routines are *very* time consuming given the importance of prior knowledge to SIESTA
- Developers spend much time on 1) compiling, 2) debugging code in fortran, 3) running examples

Lua in Siesta

- New-comers (devs) to SIESTA have a *very* large barrier on entry to the code
- Implementations of different molecular dynamics routines are *very* time consuming given the importance of prior knowledge to SIESTA
- Developers spend much time on 1) compiling, 2) debugging code in fortran, 3) running examples

What if we could make everything easier by allowing a higher level language?

Lua in Siesta

- New-comers (devs) to SIESTA have a *very* large barrier on entry to the code
- Implementations of different molecular dynamics routines are *very* time consuming given the importance of prior knowledge to SIESTA
- Developers spend much time on 1) compiling, 2) debugging code in fortran, 3) running examples

What if we could make everything easier by allowing a higher level language?

Lua

A high-level language built for small memory footprint, easy-to-learn and high flexibility.
Downside is that the core-language is very limited *on purpose*!

How it works

- An interface between SIESTA and Lua is enabled through: flook
- Enables exchange of data between the scripting language and the SIESTA core.
- Change forces → custom constraints
- Change positions → custom MD
- Change energies → custom energy-corrections
- Change *insert-your-variable(s)* → custom ???

ANYTHING MAY BE CHANGED!

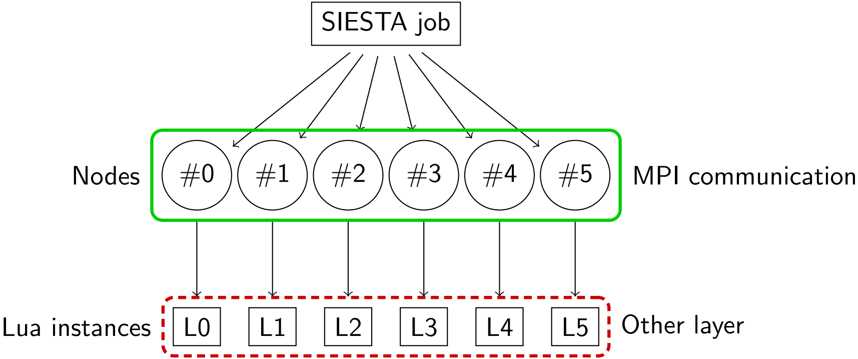
How it works

- An interface between SIESTA and Lua is enabled through: flook
- Enables exchange of data between the scripting language and the SIESTA core.
- Change forces → custom constraints
- Change positions → custom MD
- Change energies → custom energy-corrections
- Change *insert-your-variable(s)* → custom ???

ANYTHING MAY BE CHANGED!

- There *are* limitations with respect to MPI and currently available variables, however any new variable requires 1 line of fortran code.

MPI vs. Lua


SIESTA job
Nodes
#0
#1
#2
#3
#4
#5
MPI communication
Lua instances
L0
L1
L2
L3
L4
L5
Other layer

What is possible now?

flos

Lua library to perform these optimizations:

- Conjugate-gradient geometry optimization
- FIRE geometry optimization
- L-BFGS geometry optimization (extremely efficient)
- Force-constants
- NEB calculator (same as in VASP texas group)
- Mesh-cutoff convergence in *one* run

`Dependency_Tests/lua_h2o`

Tutorial

Transferability

The Lua scripts in these tutorials can directly be transferred to other systems.

1. Convergence of the mesh-cutoff
2. Geometry optimization:
   - Default siesta CG method
   - Lua CG method
   - Lua L-BFGS method
   - *mixing* of Lua CG/L-BFGS methods for faster optimization

sisl

What is it?

sisl is a powerful tool to create, post-process, and do tight-binding explorations in pure Python.
It interfaces with Siesta in many ways and can easily be used to create input tight-binding matrices for NEGF calculations (TBtrans).
Additionally, it can read and understand Siesta matrices, and thus do band-structure calculations, PDOS and much more. . .

sisl can

- interface seamlessly with ASE, pymatgen etc.
- read Hamiltonians, density matrices and can do analysis on these
- understand all spin-configurations (up to Nambu-spin, see Zeila's talk later)
- allow custom scripting in Python to create custom workflows
- interface to other DFT codes (tries to be code agnostic)
- do advanced visualization of geometries, density of states, eigenstates etc. (work by Pol Febrer)

Tutorials