AiiDA Gollum Plugin Documentation

Release 0.11

V.M. Garcia-Suarez

Mar 30, 2022

CONTENTS

CHAPTER ONE

WELCOME TO THE AIIDA-GOLLUM DOCUMENTATION!

The aiida-gollum python package interfaces the GOLLUM transport code (http://www.physics.lancs.ac.uk/gollum) with the AiiDA framework (http://www.aiida.net). The package contains plugins for GOLLUM itself and basic workflows. It is distributed under the MIT license and available from (https://github.com/garsua/aiida-gollum/).

1.1 Acknowledgments:

The Gollum input plugin, parser, workflow and examples were developed by Víctor Manuel García-Suárez.

We acknowledge support from the Spanish MINECO (project FIS2015-63918-R), the Spanish Ministerio de Ciencia, Innovación y Universidades (project PGC2018-094783-B-I00) and the EU Centre of Excellence "MaX - Materials Design at the Exascale" (http://www.max-centre.eu). (Horizon 2020 EINFRA-5, Grant No. 676598). We also thank the AiiDA team for their help.

1.2 Contents:

1.2.1 Installation

Installation

It could be useful first to create and switch to a new python virtual environment before the installation to avoid conflicts with third-party packages.

Install the plugin by executing, from the top level of the plugin directory:

```
pip install -e .
```

As a pre-requisite, this will install an appropriate version of the aiida_core python framework.

Important: Next, do not forget to run the following command to make sure all other plugins are discovered and registered

```
reentry scan -r aiida
```

1.2.2 GOLLUM plugins

Standard Gollum plugin

Description

The AiiDA Gollum plugin is an AiiDA extension, written in Python, which allows to run in a user friendly way highly automated transport simulations with the Gollum code. It has two main parts, the input and the output, which facilitate the design, analysis and postprocessing of transport calculations within the AiiDA framework.

These docs are for version: *aiida-0.12.0–plugin-0.1.0* of the plugin.

Supported Gollum versions

At least 2.0.0 version of the code, which can be downloaded from the Golum webpage (http://www.physics.lancs.ac.uk/gollum/index.php/downloads).

Inputs

- **settings**, class `ParameterData`

The name of the localfolder that contains the `Extended_Molecule` and `Lead_*` input files (in case they are necessary) and the path of the (remote) directory where the Matlab MCR is located. For example:

```
emname = os.path.realpath(os.path.join(os.path.dirname(__file__),
    "../data"))+'/Extended_Molecule'
l1name = os.path.realpath(os.path.join(os.path.dirname(__file__),
    "../data"))+'/Lead_1'
l2name = os.path.realpath(os.path.join(os.path.dirname(__file__),
```

(continues on next page)

(continued from previous page)

```
    "../data"))+'/Lead_2'
settings_dict={'additional_local_copy_list': [emname, l1name, l2name],
               'cmdline': '/share/apps/MATLAB/MCR/MCR_R2017b/v92/'}
settings = ParameterData(dict=settings_dict)
```

- **parameters**, class `ParameterData`

A dictionary with scalar and string variables and blocks, which are the basic elements of the Gollum input file. The definition of each parameter is simpler than in the Gollum original input file (it is not necessary to specify the type of variable and the number of rows and columns):

```
'Mode': 1,
'Verbose': 0,
'HamiltonianProvider': 'tbm',
'Path_EM': './Extended_Molecule',
```

Complex data structures such as blocks with various rows and columns are defined by using an appropriate key and Python's multiline string constructor. It is necessary to distinguish between numerical blocks (`NBlock`):

```
'NBlock leadp': """
2 2 -1
2 2  1 """,
```

and string blocks (`SBlock`):

```
'SBlock Path_Leads': """
1 ./Lead_1
2 ./Lead_2""",
```

The only block that is defined differently is the `atom` block:

```
'atom': """
1 2  2
0 0 10
2 2  2 """
```

The first column of the `atom` block indicates the lead number or the extended molecule (0), the second column the number of principal layers in each lead (0 again for the extended molecule) and the third column the number of atoms in each lead. From these numbers the plugin constructs and writes in the input file the typical `atom` block.

Outputs

Different output nodes can be created by the plugin, according to the calculation details. All output nodes can be accessed with the `calculation.out` method.

The output parser gets information the Gollum output file (where the output of the run is redirected) and from data files generated by the run (transmission and open-channels files).

- **output_parameters** `ParameterData` (accessed by `calculation.res`)

A dictionary with metadata, scalar result values, errors and warnings lists, and time information:

```
{
   "gollum_version": "Version 2.0 GAMMA (Feb. 2018)",
```

(continues on next page)

(continued from previous page)

```
    "oc_ef": 3.0,
    "oc_M": 6.0,
    "oc_m": 0.0,
    "tt_ef": 2.999997,
    "tt_M": 5.999995,
    "tt_m": 0.0,
    "start_of_run": "05-Jun-2018 16:59:41",
    "end_of_run": "05-Jun-2018 16:59:50",
    "total_time": 8.375386,
    "warnings": [],
    "errors": []
}
```

The data include the number of open channels at the Fermi level (`oc_ef`), the maximum (`oc_M`) and minimum (`oc_m`) open channels, the transmission at the Fermi level (`tt_ef`) and the maximum (`tt_M`) and minimum (`tt_m`) transmission. All these values are converted to 'float'. The parser also distinguishes between spin-unpolarized and -polarized calculations. In the former case it gives the values `ou_ef`, `ou_M`, `ou_m`, `tu_ef`, `tu_M` and `tu_m`, for the up open channels and transmission and `od_ef`, `od_M`, `od_m`, `td_ef`, `td_M` and `td_m` for the down open channels and transmission, respectively.

The `warnings` list contains program warnings which do not stop the execution of Gollum. The `errors` list contains the last line of the output file when the execution stops for errors or for external reasons.

- **output_array** `ArrayData`

Contains the open channels and the transmission in an array form (the energy in the *x* axis and the open channels or transmission in the *y* axis).

Errors

Errors during the parsing stage are reported in the log of the calculation (accessible with the `verdi calculation logshow` command). They are also stored in the ParameterData under the key `warnings`, and are accessible with `Calculation.res.warnings`.

Restarts

A restarting capability is implemented following the basic idiom:

```
c = load_node(Failed_Calc_PK)
c2 = c.create_restart(force_restart=True)
c2.store_all()
c2.submit()
```

The `partial.mat` file is copied from the old calculation scratch folder to the new calculation's one.

This approach enables continuation of runs that may have failed due to lack of time or other problems.

Additional advanced features

Additional settings can be specified in the **settings** input, of type ParameterData, as explained before.

Some of the options that can be specified are summarized below. In each case, after having defined the content of `settings_dic`, it can be used as input of a calculation `calc` by doing:

```
calc.use_settings(ParameterData(dict=settings_dict))
```

The keys of the settings dictionary are internally converted to uppercase by the plugin.

Adding command-line options

In order to add command-line options to the executable (particularly relevant e.g. to tune the parallelization level), each option can be passed as a string in a list, as follows:

```
settings_dict = {
'cmdline': ['-option1', '-option2'],
}
```

Note that very few user-level command-line options (besides those already inserted by AiiDA for MPI operation) are currently implemented.

Retrieving more files

If there are additional files produced by the calculation that need to be retrieved (and preserved in the AiiDA repository), they can be added as a list as follows:

```
settings_dict = {
'additional_retrieve_list': ['aiida.EIG', 'aiida.ORB_INDX'],
}
```

These files are then copied from the remote folder to the local repository.

1.2.3 GOLLUM Workflows

Gollum Siesta workflow

Description

The **GollumSiestaWorkchain** workflow produces files with the transmission and the number of open channels from the electronic structure calculated with Siesta.

Important: In order for this workflow to work it is also necessary to install the aiida_siesta plugin (http://aiida-siesta-plugin.readthedocs.io/en/latest/)

The inputs to the Gollum workchain include the Siesta code, the Gollum code, the structures of the leads and the extended molecule, the protocol, the number of kpoints in the leads and the extended molecule and some parameters.

The Gollum package can calculate transport properties such as the transmission and the number of open channels either from tight-binding or ab-initio simulations. In the latter case, it can use the Siesta or QuantumEspresso-Wannier90 codes. This workflow presents an example of transport calculation with the Siesta code. First, it launches a Siesta calculation to simulate the leads, then another Siesta calculation for the extended molecule and finally a Gollum simulation to calculate the transport properties.

Supported Gollum versions

At least 2.0.0 version of the code, which can be downloaded from the Golum webpage (http://www.physics.lancs.ac.uk/gollum/index.php/downloads).

Inputs

- **siesta_code**

A code associated to the Siesta plugin

- **gollum_code**

A code associated to the Gollum plugin

- **structure_le**, class `StructureData`

A Siesta structure for the leads.

- **structure_em**, class `StructureData`

A Siesta structure for the extended molecule.

- **protocol**, class `Str`

Either "standard" or "fast". Each has its own set of associated parameters.

- standard:

```
{
   'dm_convergence_threshold': 1.0e-4,
   'min_meshcutoff': 150, # In Rydberg (!)
   'electronic_temperature': "25.0 meV",
   'pseudo_familyname': 'si_ldapsf',
   'atomic_heuristics': {
       'Au': { 'cutoff': 100 }
   },
   'basis': {
       'pao-energy-shift': '100 meV',
       'pao-basis-size': 'DZP'
   }
 }
```

- fast:

```
{
   'dm_convergence_threshold': 1.0e-3,
   'min_meshcutoff': 80, # In Rydberg (!)
   'electronic_temperature': "25.0 meV",
   'pseudo_familyname': 'si_ldapsf',
   'atomic_heuristics': {
       'Au': { 'cutoff': 50 }
   },
```

(continues on next page)

(continued from previous page)

```
    'basis': {
        'pao-energy-shift': '100 meV',
        'pao-basis-size': 'SZ'
    }
}
```

- **kpoints_le**, class `KpointsData`

An array with the number of k-points along each direction in the leads

- **kpoints_em**, class `KpointsData`

An array with the number of k-points along each direction in the extended molecule

- **parameters**, class `ParameterData`

Some parameters for the Gollum simulation (typically the *leadp* and *atom* blocks).

Outputs

- **open_channels** `ArrayData`

The number of open channels of the first electrode (we assume at the moment that both electrodes are equal). In case of a spin-polarized calculation the output distinguishes between spin-up and down channels.

- **transmission** `ArrayData`

The transmission between electrodes. In case of a spin-polarized calculation the output distinguishes between spin-up and down transmissions.

1.2.4 Indices and tables

- genindex
- modindex
- search