What are the *httk* and the *omdb*? And, what can they do for YOU?

Rickard Armiento

Linköping University, Sweden

May 27, 2015

Linköping Linnaeus Initiative for Novel Functional Materials, LiLi-NFM

LiLi-NFM

Linköping University

Overview

The High-Throughput Toolkit (*httk*)

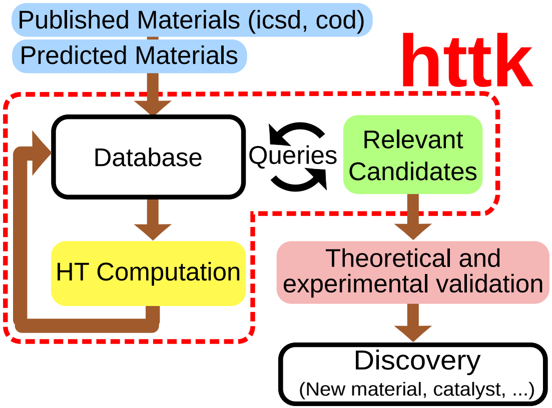
- A toolkit for preparing and running calculations, analyzing the result, store them in a global and/or in a personalized database.
- The primary focus is automatization: run with as little human intervention as possible.
- *Crucial* for large datasets; *convenient* for smaller projects!
- Intended to expand beyond atomistic calculations, but those are our primary focus for now.

The Open Materials Database (omdb)

- A central collection of computational data where we store our results.
- You can, if you want, submit results there as well.
- Easily interacts with *httk*; built using *httk*.

Database-centric High-Throughput

The *httk* is an independent implementation of the Database-centric high-throughput methodology pioneered by G. Ceder, and others.


Published Materials (icsd, cod)
Predicted Materials
httk
Database
Queries
Relevant Candidates
HT Computation
Theoretical and experimental validation
Discovery
(New material, catalyst, ...)

See: A. Jain, G. Hautier, C. J. Moore, S. P. Ong, C. C. Fischer, T. Mueller, K. A. Persson, G. Ceder, Comp. Mat. Sci. **50**, 2295 (2011).

Overview

Components:

- The *httk* python library:
    - Handling crystal structures.
    - Prepare calculations to be run.
    - Storage, retrieval, search and analysis of data in database.
- The *httk* scripts:
    - Handling large sets of computer runs.
    - Scripting that allow advanced multi-stage runs to be run on clusters with limited walltime.
    - Managing ongoing runs across many supercomputers.
    - Easy submission of results to *omdb*.

Overview

Why not extend existing libraries (ASE, pymatgen, etc.) instead?

- Different core design choices
  - Database interaction as easy as possible; python objects can be stored, searched, retrieved; mixing different databases.
  - Preserves numbers exactly (fractions instead of floating point), helps a lot with crystal geometry and database interaction.
- Different attitude to dependencies
  - No libraries outside standard python needed to get *httk* up and running. Not even numpy or scipy.
  - Other libraries can be/are called when needed.
  - *httk* goes out of its way to help you load the library you want from e.g. odd locations (helpful to, e.g., avoid old system-wide version)
- Instead, *httk* is compatible / interacts with those libraries; i.e., you can translate between ASE, pymatgen, etc., and use their features interchangeably.

Examples

A few programming examples for atomistic computation will follow.

(This is a more technical part of this presentation)

Examples: Structures

Very easy to load a cif file, or poscar, etc.:

```
import httk

struct = httk.load("example.cif")

print("Formula:", struct.formula)
print("Volume", float(struct.uc_volume))
print("Assignments", struct.uc_formula_symbols)
print("Counts:", struct.uc_counts)
print("Coords", struct.uc_reduced_coords)
```

Output:

```
('Formula:', 'BO2Tl')
('Volume', 509.24213999999984)
('Assignments', ['B', 'O', 'Tl'])
('Counts:', [8, 16, 8])
('Coords', FracVector(((1350, 4550, 4250), ..., ,10000)))
```

Examples: Structure

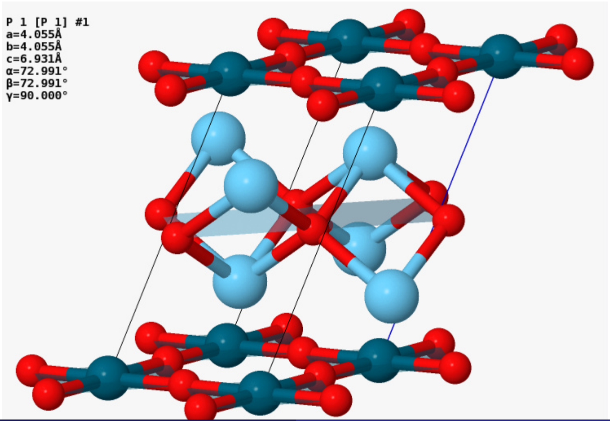
Of course one can also create and modify structures directly in code:

```
from httk.atomistic import Structure

cell = [[1.0, 0.0, 0.0],
        [0.0, 1.0, 0.0],
        [0.0, 0.0, 1.0]]

coordgroups = [[
                  [0.5, 0.5, 0.5]
               ],[
                  [0.0, 0.0, 0.0]
               ],[
                  [0.5, 0.0, 0.0],[0.0, 0.5, 0.0],[0.0, 0.0, 0.5]
              ]]

assignments = ['Pb','Ti','O']
volume=62.79
struct = Structure.create(uc_cell=cell,
            uc_reduced_coordgroups=coordgroups,
            assignments=assignments,
            uc_volume=volume)
```

Examples: Visualization

Easy visualization using, e.g., jmol

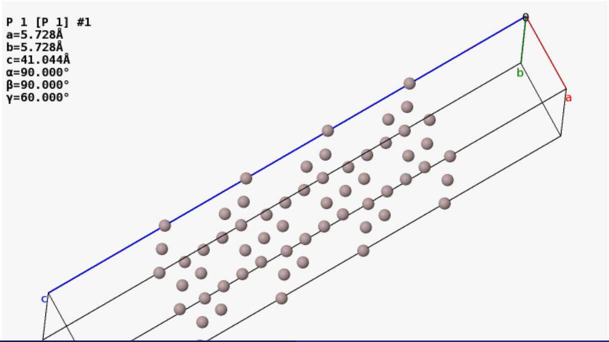
```
import httk
import httk.atomistic.vis

struct = httk.load("POSCAR")
struct.vis.show()
```



Examples: External libraries

Easy interaction with other useful libraries and software, e.g., ASE:
(Present bindings: ASE, aflow, cif2cell, isotropy, jmol, pymatgen, platon, gulp)

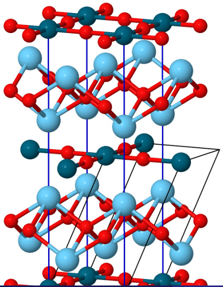
```
from httk.atomistic import Structure
import httk.atomistic.vis
import httk.external.ase_glue
import ase.lattice.surface

slab = ase.lattice.surface.fcc111('Al', size=(2,2,10), vacuum=10.0)
struct = Structure.ase.from_Atoms(slab)
struct.vis.show()
```

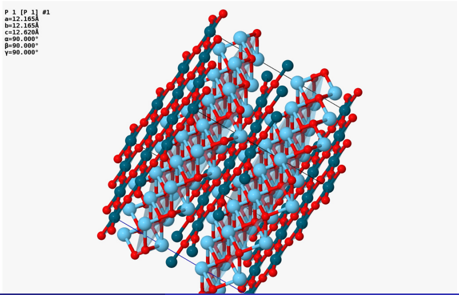


Examples: Supercells

Build supercells, find orthogonal and cubic ones:

```
import httk
import httk.atomistic.vis

struct = httk.load("POSCAR")
supercell1 = struct.build_supercell([[2,0,0],[0,2,0],[0,0,1]])
supercell1.vis.show()
supercell2 = struct.build_orthogonal_supercell(tolerance=20)
supercell2.vis.show()
supercell3 = struct.build_cubic_supercell(tolerance=20)
supercell3.vis.show()
```



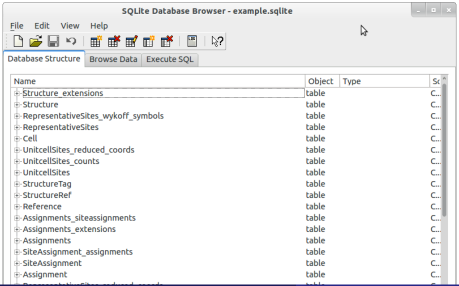


Examples: Database

Store data in a local relational database (sqlite):

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)

struct = httk.load('example.cif')
store.save(struct)
```

```
user@computer:> sqlitebrowser example.sqlite
```



Name	Object	Type	Sc
Structure_extensions	table		C...
Structure	table		C...
RepresentativeSites_wykoff_symbols	table		C...
RepresentativeSites	table		C...
Cell	table		C...
UnitcellSites_reduced_coords	table		C...
UnitcellSites_counts	table		C...
UnitcellSites	table		C...
StructureTag	table		C...
StructureRef	table		C...
Reference	table		C...
Assignments_siteassignments	table		C...
Assignments_extensions	table		C...
Assignments	table		C...
SiteAssignment_assignments	table		C...
SiteAssignment	table		C...
Assignment	table		C...

Examples: Database

Search in your local database:

```
import httk
from httk.atomistic import Structure
import httk.db

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output(search_struct,'structure')

for match, header in search:
    struct = match[0]
    print("Found:",struct.formula)
```

Output:

```
('Found:','ZnO2')
```

Examples: Computations

Setup a simple VASP calculation to run manually:

```
import httk
import httk.iface.vasp_if

poscarspath="/path/to/your/poscars/POT_GGA_PAW_PBE/"

struct = httk.load("example.cif")

httk.iface.vasp_if.prepare_single_run("Run", struct,
     template='t:vasp/single/static', poscarspath=poscarspath)
```

```
user@computer:> cd Run
user@computer:> vasp
```

Output:

```
 running on    1 nodes
 distr:  one band on    1 nodes,    1 groups
 vasp.5.2.12 11Nov11 complex

 POSCAR found type information on POSCAR  Ti N
 POSCAR found :  2 types and       2 ions


 ----------------------------------------------------------------------
```

Examples: Computations

Generate a big batch of computations:

```
import httk, httk.task, httk.db
from httk.atomistic import Structure

backend = httk.db.backend.Sqlite('tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add_all(search_struct.formula_symbols.is_in('O','Ca','Ti'))
search.output(search_struct,'structure')

for match, header in search:
  struct = match[0]
  httk.task.create_batch_task('Runs','/vasp/batch/relax',
    {"structure":struct})
```

Examples: Computations

Run the batch of computations on a supercomputer ('kappa'):

```
user@computer:> httk-project-setup example_project
user@computer:> httk-computer-setup ssh-slurm kappa
user@computer:> httk-computer-install kappa
```

```
user@computer:> httk-tasks-send-to-computer kappa Runs/
user@computer:> httk-tasks-start-taskmanager kappa
```

```
user@computer:> httk-tasks-status kappa
```

```
user@computer:> httk-tasks-receive-from-computer kappa Runs/
```

Examples: Computations

Read results back into database

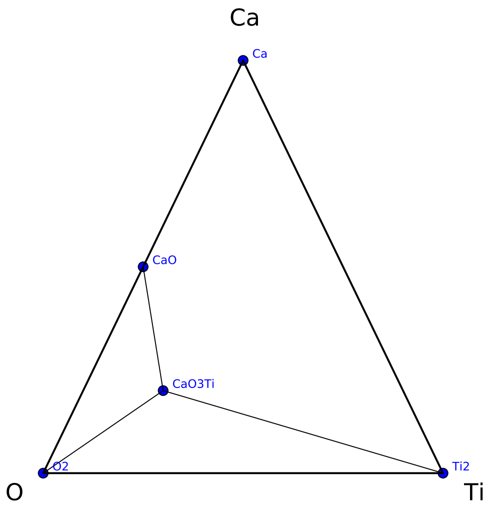
```
import httk, httk.db, httk.task, os
from httk.atomistic.results import Result_TotalEnergyResult

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)

reader = httk.task.reader('./','Runs/')

for rundir, computation in reader:
    struct = httk.load(os.path.join(rundir,"CONTCAR"))
    outcar = httk.iface.vasp_if.read_outcar(os.path.join(rundir,"
        OUTCAR.cleaned.relax2"))
    total_energy_result = Result_TotalEnergyResult(computation,
        struct, float(outcar.final_energy))
    store.save(total_energy_result)
```

Examples: Computations

Draw a phase diagram from your stored batch runs

```python
import httk, httk.db, httk.task, httk.atomistic.vis
from httk.atomistic import Structure, StructurePhaseDiagram
from httk.atomistic.results import Result_TotalEnergyResult

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)
search = store.searcher()
search_total_energy = search.variable(Result_TotalEnergyResult)
search_struct = search.variable(Structure)
search.add(search_total_energy.structure == search_struct)
search.add_all(search_struct.formula_symbols.is_in('O','Ca','Ti'))
search.output(search_total_energy,'total_energy_result')

structures, energies = [],[]
for match, header in search:
    total_energy_result = match[0]
    structures += [total_energy_result.structure]
    energies += [total_energy_result.total_energy]

pd = StructurePhaseDiagram.create(structures,energies)
pd.vis.show(debug=True)
```

Examples: Computations



Note: phase-diagram support in httk does not yet draw *all* phase lines.

Examples: Computations

It is easy to put your own data in the database

```
import httk, httk.db
from httk.atomistic import Structure

class StructureIsEdible(httk.HttkObject):
    @httk.httk_typed_init({'structure':Structure,
                           'is_edible':bool})
    def __init__(self, structure, is_edible):
        self.structure = structure
        self.is_edible = is_edible

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)

tablesalt = httk.load('NaCl.cif')
arsenic = httk.load('As.cif')

edible = StructureIsEdible(tablesalt,True)
store.save(edible)
edible = StructureIsEdible(arsenic,False)
store.save(edible)
```

Submit results to the global database

Submit to central database

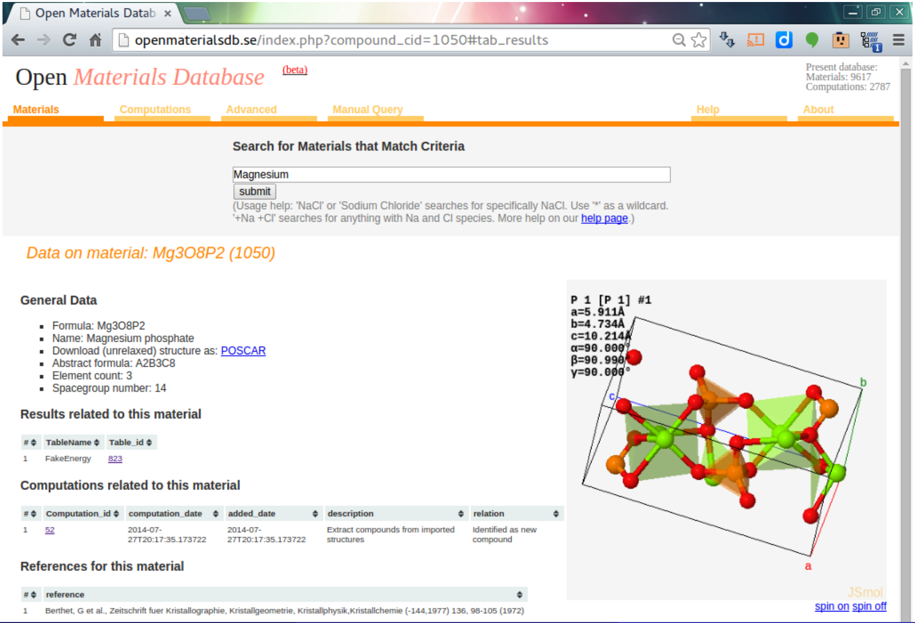
```
user@computer:> httk-project-submit
```

Note:

- Will verify very carefully that you actually mean to make your data publicly available on the web via *omdb*.
- Your files are signed by a private key; you can always be identified as the 'owner' of these files.
- You can change/add reference information after submission by editing ht.project/references and running httk-project-submit-update-references
- You can withdraw your data at a later point with httk-project-submit-widthdraw

Note: when you run `httk-project-setup` a directory `ht.project` is created to identify *this* project. You can copy the project directory everywhere you have files relating to this project. You can then run `httk-project-submit` in each such directory, and the files are aggregated on our servers.

Open Materials Database



Open Materials Datab ×

openmaterialsdb.se/index.php?compound_cid=1050#tab_results

Open *Materials Database* (beta)

Present database:
Materials: 9617
Computations: 2787

Search for Materials that Match Criteria

Magnesium

submit

(Usage help: 'NaCl' or 'Sodium Chloride' searches for specifically NaCl. Use '*' as a wildcard. '+Na +Cl' searches for anything with Na and Cl species. More help on our help page.)

Data on material: Mg3O8P2 (1050)

General Data

- Formula: Mg3O8P2
- Name: Magnesium phosphate
- Download (unrelaxed) structure as: POSCAR
- Abstract formula: A2B3C8
- Element count: 3
- Spacegroup number: 14

Results related to this material

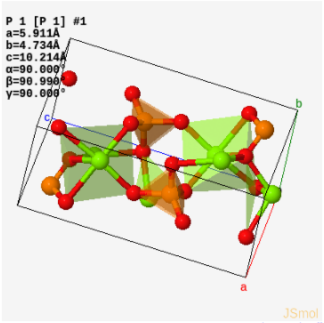
#	TableName	Table_id
1	FakeEnergy	823

Computations related to this material

#	Computation_id	computation_date	added_date	description	relation
1	52	2014-07-27T20:17:35.173722	2014-07-27T20:17:35.173722	Extract compounds from imported structures	Identified as new compound

References for this material

#	reference
1	Berthet, G et al., Zeitschrift fuer Kristallographie, Kristallgeometrie, Kristallphysik,Kristallchemie (-144,1977) 136, 98-105 (1972)



spin on spin off

Operate on Data in Open Materials Database

Operate directly on data present in the open materials database:
(Not in present version; will be in next.)

```
import httk, httk.db
from httk.atomistic import Structure

store = httk.db.open_materials_database_store

search = store.searcher()
search_struct = search.variable(Structure)
search.add_all(search_struct.formula_symbols.is_in('O','Ca','Ti'))
search.output(search_struct,'structure')

for match, header in search:
  struct = match[0]
  httk.task.create_batch_task('Runs','template',
    {"structure":struct})
```

Installation

Easy to install

Linux / Unix / Mac OS X / Cygwin:

```
user@computer:> mkdir ~/bin/python
user@computer:> cd ~/bin/python
user@computer:> curl -O http://httk.openmaterialsdb.se/downloads/
    httk-latest.tgz
user@computer:> tar -zxvf httk-latest.tgz
user@computer:> ls
  httk-1.0.0  httk-latest.tgz
user@computer:> ln -f -s httk-1.0.0 httk-latest
user@computer:> source ~/bin/python/httk-latest/setup.shell
```

Put the last statement in your .bashrc / .cshrc to always set the paths up correctly.

Concluding remarks

- The High-Throughput Toolkit (*httk*) is a framework for easy automatization of computational projects. It helps with setup, execution, storage and search.
- A framework like this is *crucial* for projects that work with large datasets, but also *convenient* for smaller projects.
- For our own calculations, we store them at `openmaterialsdb.se`; you can also submit your results there if you want (with your papers cited and linked).

Funding:

- The Swedish Research Council (VR) Grant No. 621-2011-4249.
- The Linnaeus Environment at Linköping on Nanoscale Functional Materials (LiLi-NFM) funded by The Swedish Research Council.