Blast Workshop

Data management and reproducible simulation runs

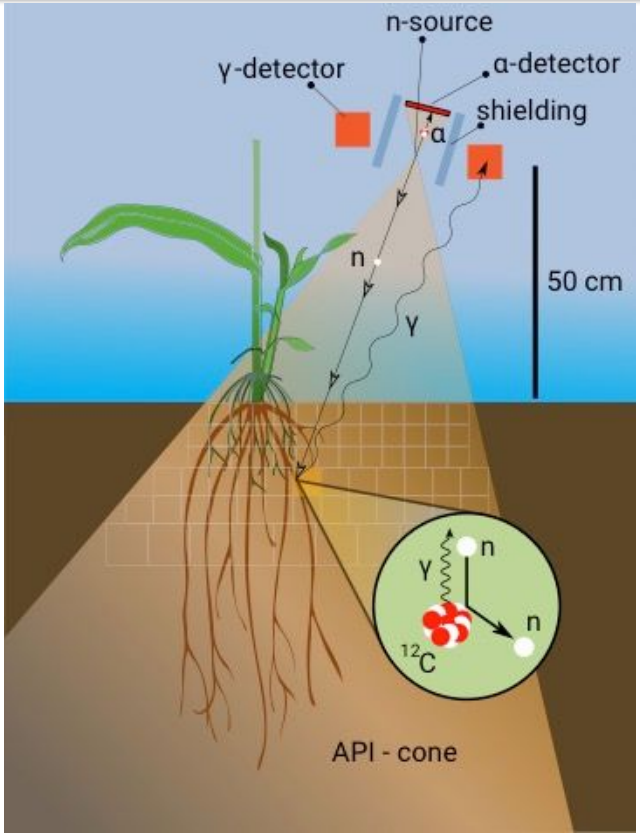
Arun Persaud
Lawrence Berkeley National Laboratory

2018-05-08

Data management and reproducible simulation runs

- Physicist, not a computer scientist
- Don’t really know what I’m doing ;)
- No Warp or other BLAST codes involved, can be applied to Warp input files

Utilize isotope-specific response to neutron flux to measure carbon distribution in soil



Neutrons excite isotopes which emit characteristic gamma rays of isotope-specific energies

Associated Particle Imaging combined with time-of-flight analysis enables correlation of measured gamma ray with nucleus location in the soil

Measured gamma rates reflect carbon concentration

We need to run many simulations

- Geant4
- MCNP
- Python scripts

We run these locally and on a cluster, producing (for us) large output files. We often run the simulations again with slight modifications, e.g. more particles, different number of cores, etc.

Problem: how to create reproducible runs

Possible solution:

- One directory for each run, keep all inputs, perhaps even the outputs

Issue:

- hard to keep track
- hard to compare
- space requirements

We already keep all our input files in git, so instead of saving all inputs, just save the git hash we are using and make sure that the file we are running is actually in git.

Started working on a script to run tasks

- Check if input file is in git otherwise don't run (currently only works for main input file)
- Create a single line in a logfile
  - hash for the run
  - name of input file
  - git commit hash of the input file
  - save each event as a JSON string

```
commits = subprocess.run(['git', 'rev-list', '--all'],
                         cwd=dir,
                         stdout=subprocess.PIPE,
                         encoding='ascii').stdout[:-1].split('\n')
# go through all commits newest will be first
for c in commits:
    blobs = subprocess.run(['git', 'ls-tree', '-r', c],
                           cwd=dir,
                           stdout=subprocess.PIPE,
                           encoding='ascii').stdout.split('\n')
    for b in blobs:
        if not b:
            continue
        h = b.split()
        if not h[1].startswith('blob'):
            continue
        if h[2] == hash:
            return c
```

Logfile entry:


```
{"hash":"72fbce0c7d2d779c7f40c9273cbbe8b446ce1ce8",
 "file":"resistive-network.py",
 "file_commit":"94f2ef35b281a7bd5c0204ac93eb75eeadbe685d",
 "command_line":"python3 resistive-network.py"}
```

Since we now have a script that runs our programs...

- Make the input files jinja[1] templates
  - make number of cores/particles or other parameters accessible
    important for our monte carlo simulations:
    random number seed
  - replace before we run the script
  - save parameters as dictionary in logfile
- Make the command line also a jinja template
- Save some other metadata
  - runtime
  - start time
  - version of the script that runs the software

[1]http://jinja.pocoo.org/

Also make running on different platforms easier

- Automatically figure out what platform we are running on and execute the script accordingly
  e.g. on the cluster create a SLURM sbatch file (from a jinja template) and submit job automatically
- Use config file to define repositories and regexp for platforms, pre/post scripts to run on different platforms
- If running for example dask, create scheduler on cluster

This makes reproducing results easier

- Ability to rerun a certain script with the same input parameters
- Rerun and overwrite some parameters for parameter scans

Command line options

```
Usage:
  reproduce run [options] [--] <command>
  reproduce dryrun [options] [--] <command>
  reproduce addrepo [options] <alias> <path>
  reproduce listrepos
  reproduce addplatform [options] <regex> <type>
  reproduce listplatforms
  reproduce list [<howmany>]
  reproduce show <hash>
  reproduce monitor <hash>

Options:
  -p <key:value>              for several parameters use k1:v1,k2:v2 syntax
  --list-parameters           list all parameters that need to be set
  --show                      shows the input file (same as 'show', but can be used for run/dryrun)
  --template <template>       the name of the file that should be treated as a template inside the
                              command
```

Config file

```
[roots]
path = /home/arun/projects/roots/simulations/atap-roots-simulations/
log = simulations.log

[reproduce]
path = /home/arun/projects/roots/simulations/reproducible/
log = simulations.log

[platforms]
^n0[0-9]+\.[a-z]+[0-9]+$ = SLURM
# default is desktop
```

Issues

- Still reproducible issues: which version of python, which compiler for geant4, which modules loaded on cluster, …
- Better backend than just a text file?
  - python tinydb, but hard to merge DB between different computers
  - git itself as a backend?

Next steps

- Planning on making this open source
- Automatically add hash to output, e.g. monkey patch matplotlib's plot function to include the hash. This makes it easy to rerun results if you just have the plot

Curious what other people are using.
Happy to talk more about this.

Thank you for your attention!