Submitting FLUKA simulations to LXBATCH

Tim Cooijmans
tim.cooijmans@cern.ch

version 2.x

1 Introduction

This file documents a collection of tools developed for the CMS BRIL Radiation Simulation group. The package consists of three scripts: `split.py`, `execute.py`, and `combine.py`. The typical use case is to split a big FLUKA simulation up into smaller jobs, submit them to LXBATCH, and finally combine the results.

2 Splitting the simulation into multiple jos

2.1 Syntax

```
$ /path/to/split.py SIMULATION NPRIMARIES NSPLITS
```

2.2 Arguments

- `SIMULATION` - the name of the main FLUKA input file. Inclusion of the `.inp` extension is optional.
- `NPRIMMRIES` - the number of primary particles to be simulated by each job.
- `NSPLITS` - the number of jobs to generate.

2.3 Description

The script `split.py` generates multiple statistically independent copies of the simulation `SIMULATION` for submission to LXBATCH.

The generated files are named starting with `SIMULATION_aaaa.inp` and counting up to `SIMULATION_zzzz.inp`.

The random seeds for the jobs are generated by counting up from the seed given in `SIMULATION`, or from 0 if no such seed was found. Note that the `RANDOMIZ` card must the present in the main input file and that it must be in fixed format.

As of version 2014-05-28, there is a more general mechanism for introducing differences between the generated input files. The main input file may contain lines of the form

```
*#lxbatch iterate /path/to/file
```

which will be replaced in consecutive generated inputs by consecutive lines from the specified file. In the above case, the line will be replaced by the first line of `/path/to/file` in the first generated input, by the second line in the second generated input, and so on. An example use case is given below.

If the current directory already contains files named like the files to be generated, the user is asked to choose whether to replace them or to generate the new files in addition to the old ones. The first option ("`replace`") moves the old files out of the way into a temporary directory, and then generates the new files as usual. The second option ("`union`") leaves the old files in place, and generates the new files with names and seeds counting up from the highest values already used for the old files.

Note: no attempt has been made to make the `iterate` feature work well in conjunction with the `union` feature, so always use replace or create the new files in a separate directory. Specifically, `iterate` always takes lines starting with the first line in the file, regardless of whether it has already been "used" in one of the existing files (which would be a complicated and fragile ordeal to detect).

2.4 Example

```
$ /path/to/split.py CMSpp 1000 10
```

The above splits the simulation `CMSpp.inp` into 10 jobs, each simulating 1000 primaries. Files `CMSpp_aaaa.inp` through `CMSpp_aaaj.inp` will be created, each with different random seed.

An example of where the iterate mechanism described above is useful is in passing varying arguments to custom user routines. For instance, the main input file could contain the following:

```
[...]
DEFAULTS
*#lxbatch iterate source_cards
BEAM           1.0                                     ISOTOPE
[...]
```

with the `source_card` containing:

```
SOURCE           0.0
SOURCE           1.0
SOURCE           2.0
SOURCE           3.0
SOURCE           4.0
[et cetera]
```

Now the generated input files would each have one of these `SOURCE` cards in place of the `*#lxbatch iterate source_cards` line. The `source_cards` file can be easily generated using a script.

3 Submitting the jobs for execution

3.1 Syntax

```
$ /path/to/execute.py  [-e EXECUTABLE] [-q JOBFLAVOUR] [-L]
                       [--unless-finished] SIMULATION [FILE1 FILE2 ...]
```

3.2 Arguments

- `-e EXECUTABLE` (optional) - the FLUKA executable to run. Passed to `rfluka` as-is.
- `-q JOBFLAVOUR` (optional) - run time for the job. The possible options are:

`espresso`	= 20 minutes
`microcentury`	= 1 hour
`longlunch`	= 2 hours
`workday`	= 8 hours
`tomorrow`	= 1 day
`testmatch`	= 3 days
`nextweek`	= 1 week

  If not specified, the default is `tomorrow`.
- `-L` (optional) - don't submit to LXBATCH; run the job locally instead. Useful for debugging.
- `--unless-finished` (optional) - don't submit jobs for which there is already a results file present.
- `SIMULATION` - the name of the main FLUKA input file. Inclusion of the `.inp` extension is optional.
- `FILE1, FILE2` (optional) - names of specific input files to submit: `.inp` extension should be included.

3.3 Description

The script `execute.py` submits FLUKA input files to LXBATCH. The output files generated by FLUKA for each job `JOB.inp` will be archived in a ZIP file named `results_JOB.zip`.

By default, the input files are found based on the naming scheme used by `split.py`, i.e., all files matching `SIMULATION_*.inp` are submitted. Alternatively, if any specific files `FILE1`, `FILE2`, ... are specified (with `.inp` extension), only these files will be submitted.

If the `--unless-finished` switch is given, files that correspond to already completed jobs will be filtered out before submission. An input file `JOB.inp` is considered to correspond to an already completed job if the corresponding result ZIP file is present. This feature is useful for resubmitting failed jobs, but at present it does nothing to avoid resubmitting jobs that are still running or pending.

The user can instruct `execute.py` to invoke additional shell commands as part of the job by placing before and after directives in the input file. For instance:

```
[...]
* copy a magnetic field map into the job's working directory
before invoking fluka
*#lxbatch before cp /path/to/some.map .
* copy the simulation results to eos
*#lxbatch after $LX_EOS cp results_$LX_INPUT_BASE.zip /eos/cms/store/somewhere/
GEOBEGIN
[...]
```

These directives can be located anywhere in the input files, and need not be the same for each input file. The following environment variables are made available for convenience:

- `LX_ORIGIN` - the path to the directory from which the jobs were submitted.
- `LX_INPUT_BASE` - the name of the main FLUKA input file, without `.inp` extension.
- `LX_INPUT` - the absolute path to the FLUKA input file, with `.inp` extension.
- `LX_EOS` - the path to the `eos` executable.

Note that in any case, the FLUKA installation located in `$FLUPRO` as well as any executable specified with the `-e` switch should have been compiled and linked on `lxplus`.

3.4 Example

```
$ /path/to/execute.py -e CMSpp CMSpp
```

The above submits files named `CMSpp_*.inp` to LXBATCH. `rfluka` will use `CMSpp` for the FLUKA executable. Multiple ZIP files `results_CMSpp_*.zip` holding the FLUKA output will be created.

4 Combining the results

4.1 Syntax

```
$ /path/to/combine.py SIMULATION SCORINGS
```

4.2 Arguments

- SIMULATION - the name of the main FLUKA input file. Inclusion of the .inp extension is optional.
- SCORINGS - the name of a file listing the FLUKA output units and their associated scorings.

4.3 Description

The script combine.py aggregates results from split FLUKA simulations. The results are expected to be found in ZIP files matching results SIMULATION_*.zip. The desired scorings are read from SCORINGS. Each nonempty line in this file specifies two values; the output unit (ranging from 21 to 99) and the corresponding scoring. For each output unit, the appropriate FLUKA program (e.g. usbsuwfor USRBIN) is called to combine the results across the jobs. The names of the output files are prefixed with SIMULATION, the scoring type and the output unit number.

4.4 Example

```
$ cat scorings
31 USRBIN
21 USRBDX
22 USRBDX
$ /path/to/combine.py CMSpp scorings
```

The above combines results from the ZIP files results CMSpp_*.zip. Output from unit 31 (respectively 21, 22) will be processed by usbsuw (usxsuw, usxsuw) and the combined results written to files matching CMSpp_usrbin_31* (CMSpp_usrbdx_21*, CMSpp_usrbdx_22*).