Exercises - Set 2

Advanced FLUKA usage: Parallelization and USER-routines in proton therapy context

Niels Bassler <bassler@phys.au.dk> for IFJ-PAN, Krakow; May 2014.

Notes to exercises:

- *General programming knowledge is required*, ideally FORTRAN 77 (but not necessarily)
- If you missed some of the tasks from last Exercise, feel free to investigate these. Get the basics right before you foray into more advanced use
- Still, exercises are meant as suggestions, you are welcome to deviate from these (at own risk, we might not be able to help you if you run into trouble).

1. The Need for Speed: Parallelization of your runs

1. *Simple parallel processing (local version)*
   If you have started your virtual machine with 6 CPUs, you can spawn / clone your jobs, to use more CPUs simultaneously.

   SPAWN: in flair the spawn field will generate as many more runs as you want using an incremental random seed. All files will be merged using the process button as usual.

   You can use the CLONE button the run menu, to clone your runs. Attach a unique random number to all of these, then submit them all to the queue and wait until they are done. If you intend to use many clones, take a look at the loop button. Here they will not be automatically merged, (presumably the clone function is intended if you want to vary some of the parameters).

2. *Using a job-submission system (torque) to submit to a cluster from FLAIR (still in serial):*
   In flair, go to the “flair” tab, select the “config” button. A window pops up, select the “Run” entry from the list. Right-click on the queue list, select “Add”, and add a new queue “myQueue” pointing to **`/usr/local/fluka/qfluka.sh`**
   You can try to submit the job to the cluster, using one of your previously calculated examples.

   It is here assumed that a working submission system is set up. This is however not really the case for your virtual machine, since I messed things up. If you want, you can repair it by following the fix-me user-guide guide I uploaded to the http://neptun.phys.au.dk/~bassler/TUTORIALS/MC

3. *Submitting to a cluster from FLAIR (Torque, parallel version):*
   Fully analog, you can use the SPAWN field, to spawn your runs on the cluster, submit them all to the queue and wait until they are done.

4. *An alternative for embarrassingly parallel processing (Ninja version):*
   Since I am not too happy with flair, I did a little script, which does all the parallelization for you. IMHO this is simpler and faster to use. *It will only work, if you have a job-system setup.*

   Download it

   ```
   wget http://neptun.phys.au.dk/~bassler/FLUKA/scripts/rtfluka.sh
   ```

   When you place the script in the same directory where your example is, you can submit it for parallel processing like this:

   ```
   qsub -V -t 0-9 -d . rtfluka.sh
   ```

   ...where the **`-t 0-9`** means we will run 10 instances. (20 instances are thus 0-19). This command should be equivalent to hitting the run button, but there is no GUI to tell you about the progress of the simulation. Instead, use the **`qstat`** command to see when they have completed.

   When all your runs have completed, you can switch back to the flair interface, and proceed with the data processing as usual.

a. You can use the command "**top**" from the command line to see your CPUs working. Toggle the CPU list by pressing "**1**". Press "**q**" to quit the program.

2. Define your own beam sources in FLUKA (calculate SOBPs)

a. Copy the default `/usr/local/fluka/usermvax/source.f` to your working directory, and modify the program, so three different energies are emitted, eg. 10, 20 and 50 MeV. Compile it using eg. ldpm3qmd and make sure you run it with the proper executable (you can select this in the run tab in flair.)
   Score the resulting depth-dose curve. This should give three different Bragg-peaks
b. Modify the same program to sample from a gaussian distribution of energies.
c. Use the supplied TRiP98 raster file (.rst), investigate the contents. Apply the script rst2sobp.py to convert it to "sobp.dat" file. Read the resulting sobp.dat file with the supplied custom made source.f file.

3. Solid state dosimetry and quenching effects: *the wonders of Alanine and the plight of TLDs*

a. Setup a PMMA phantom of sufficient size and shape for a circular r=5 cm 120 MeV beam.

b. Build a stack of 10 cylindrical alanine pellets. Pellets have a with diameter 0.45 cm, thickness 0.2 cm. Position them so they cover the bragg peak. (estimate again using http://dedx.au.dk or fancy android app)
c. Score the physical dose in each of the 10 pellets (Note: use region scoring!)
d. Using the supplied `comscw_alanine.f` script which the actual response equivalent photon dose (which is what e.g. NPL reports back when alanine pellets are used for read out)
e. Think about “dose-to-water” and “dose-to-alanine”.
    i. What is being scored by FLUKA?
    ii. What is the detector most likely calibrated in?
    iii. [***Expert question***] What would you have to do in order to translate from dose-to-water to dose-to-alanine, and how would you do it using FLUKA? *[**free beer** for anyone who comes up with a halfway decent solution, not involving the DOSE-H2O estimator]*
f. [***Super expert question***] Why can a similar function as `comscw_alanine.f` *not* be made for TLD-100/600/700 at clinical dose levels?

4. Ripple filters and range modulators in FLUKA

a. Make sure you understand what a ripple filter is.
b. Using a given sobp.dat and input file, do a spread out Bragg peak calculation with and without ripple filter. (Best effects are seen using a heavy ion beam, such as C-12.)
c. Modify the rifi user routine to simulate one of your own range modulator wheels, i.e. so you can generate an SOBP from a monoenergetic proton beam. Leszek will possibly provide a proper function.