Josh Yoskowitz

How to use GPT on the JLab iFarm

1 Intro

Creating many GPT simulations on a single desktop PC can be tedious and time consuming. The advantage of running GPT on the farm is that you can create many GPT simulations in parallel instead of in series, which can save countless hours (assuming you know what you are doing). The documentation below of basic instructions on how to run GPT simulations on the farm is the culmination of countless hours and countless Advil and countless questions asked to countless people and countless times I've screwed up and had to start all over. I assume that you are either a grad student or an employee looking to use the farm to speed up your GPT simulations. I also assume (and hope) that you have a decent amount of experience with GPT and some experience with Unix shells to understand the example I give below. I hope the below documentation proves useful to you and your research...at least it's much better than the online documentation.

2 Step 1: Are you sure you know what you're doing?

There are countless ways to run on the farm and not get what you want. There are even more ways to have some interesting syntax error that prevents GPT from even running in the first place. Thus, be sure that, at the very least, one of your GPT simulations runs to completion on one of the JLab machines (such as jlabl1), not just your desktop PC. Try running a simulation on jlabl1. If you get any errors at this point, it's better to deal with them here than on the farm. If it works smoothly, you're most likely good to go......we'll see.

Note that in order to run GPT on jlabl1, you'll need to know how to use GPT on a Unix machine (i.e. with a terminal interface). If you don't have access to a JLab Linux workstation (like an accelerator computer) and mainly use a Windows PC like I do, you can try logging onto jlabl1 using PuTTY (see https://cc.jlab.org/windows/remotedesktop You'll only need to do "Stage 1"). Once you've logged into login.jlab.org, you can type "ssh jlabl1" and enter your CUE password to log onto jlabl1. Be sure that *all* of your simulation files (i.e. your input file and fieldmap files) are in an easily accessible place on your J:\ drive on JLab Windows machines (or /u/home/<username>/ directory on Linux machines). You can then try running GPT on jlabl1 using your simulation files.

3 Step 2: Are you sure you're allowed?

Once you have a working GPT simulation, the first step to running on the farm is to gain access to the farm. If you don't already have access, you'll need to ask your supervisor about getting access to the farm. In order to use the farm, you need to be a part of one of the groups that is "allowed" to use the farm. I'm in the group called "inj_group", which is an allowed group for using the farm. Note that this has nothing to do with whether or not you are *allowed* to use the farm (Of course you are!), it's to make sure that the farm is only used by those who are authorized to use it (Of course you are, too!). To see what groups you are in, type "groups <your CUE username>" in any JLab terminal.

4 Step 3: Are you even ready?

Once you get access to the farm, you'll need to create an Auger command (batch) file. Auger is the software that manages batch files that are sent to the JLab farm computers to run your jobs. Basically what you'll do is to create a batch file where you'll provide info about your project and what kind of jobs you'd like to run on the farm. Auger will then take that info and send out your jobs to the farm computers (one job per computer). The jobs will be added to the queue with a certain priority (I wouldn't worry about this if you're just starting out. If I were you, I'd be ecstatic if your simulations successfully run on the farm period, regardless of when they actually run.) Once your simulations have finished (successful or not), Auger will then copy the resulting data file(s) back to your directory before deleting them on the farm computer(s).

There are two formats of Auger batch files that you can choose from: one of which you can use and one of which you're gonna use. I'll show how to create the latter below:

4.1 Creating an Auger XML batch file

The xml file consists of two main parts: one for global specifications and one for job specific specifications. To illustrate what each of these sections are and to understand the syntax, I'll show you an example xml batch file below:

```
<Request>
<Email email="yoskowij@jlab.org" request="true" job="true"/>
<Project name="inj_group"/>
<Track name="test"/>
<Name name="GPT_testrun"/>
<TimeLimit time="10"/>
<Memory space="600" unit="MB"/>

<Job>
        <Stdout dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/stdout/stdout_1.txt"/>
        <Stderr dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/stderr/stderr_1.txt"/>
        <Command>
                use gpt
                setenv OMP_WAIT_POLICY PASSIVE
                cp /u/group/inj_group/yoskowij/GPT_Farm_Files/result_100.gdf
                gpt -j 2 -o filename1.gdf T-Gun_Farm.in eN=10000 eKEmax=100
        </Command>
<Output src="filename1.gdf" dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/result_100.gdf"/>
</Job>
<Job>
        <Stdout dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/stdout/stdout_2.txt"/>
        <Stderr dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/stderr/stderr_2.txt"/>
        <Command>
                use gpt
                setenv OMP_WAIT_POLICY PASSIVE
                cp /u/group/inj_group/yoskowij/GPT_Farm_Files/result_200.gdf
                gpt -j 2 -o filename1.gdf T-Gun_Farm.in eN=10000 eKEmax=200
        </Command>
<Output src="filename1.gdf" dest="/u/group/inj_group/yoskowij/GPT_Farm_Files/result_200.gdf"/>
</Job>

</Request>
```

4.1.1 Global Specifications

Everything above <Command> are global specifications. They apply to *all* jobs sent to the farm. Some notes about each line:

- The <Email> command will tell Auger to email you when the job is completed and tell you whether or not it was successful. It emails the address specified in quotes.
- The <Project> command specifies what project your jobs are a part of, assuming it's an allowed project. You can input your project within the quotes. This is a required command.
- The <Track> command specifies the type of jobs you are running (see https://scicomp.jlab.org/docs/batch_job_tracks). If you are just starting out, I'd recommend using either the "test" or "debug" tracks. Once you know that your jobs will run smoothly on the farm, you can choose one of the other tracks. This is a required command.
- The <Name> command gives your jobs a name so that you can refer to these jobs after they are completed. You can name your jobs whatever you want, so long as you can refer to them easily. This is a required command.

- The <TimeLimit> command, as its name suggests, gives a time limit to your jobs. This is useful, as you don't want you jobs to hang on the farm and run forever. You can specify a value and its unit. If you don't specify a unit, the default is minutes. For example, to specify 30 minutes, you can use any of the following: <TimeLimit time="30"/>, <TimeLimit time="30" unit="minutes"/>, <TimeLimit time="0.5" unit="hours"/> .
- The <Memory>, as its name suggests, specifies the memory usage limit of your jobs. If you're just starting out, this should not be a problem, but it may become a concern for larger simulations. To see if you should be concerned about memory usage, try creating your GPT simulation on a Windows PC and use Task Manager to see how much memory GPT uses. From there, you can choose a reasonable memory limit for your simulations. You obviously don't want to use too much memory, as it would be unfair to everyone else using the farm.

For more info on the various specifications, refer to https://scicomp.jlab.org/docs/desc_xml_tags.

4.1.2 Global Commands vs Job Commands

Between <Command> and </Command> are the commands you give to a JLab farm computer, just as you would in Terminal. Which computers receive which commands depends on where the command is specified in the xml file. One option is to put the command specifications before the job specifications. In this case, the commands you specify are *global*, meaning that *all* jobs run these commands (i.e. each farm computer you use runs the same commands). Another option, as I have shown in the example, is to specify commands within the job specifications themselves. In this case, whatever commands you specify only run for a *specific* job (and no other job). Note that if you have both global and job-specific commands the job-specific commands *overwrite* ***all*** global commands. (This is why the command "use gpt" is written for each job instead of having it as a global command, as the job-specific commands would otherwise overwrite it).

The first line tells the farm computer you'd like to run GPT (specifically it adds the license number to your environment variables so that you have "permission" to run GPT). You are required to have at least one command in your xml file in order to run your job (obviously...).

The second line sets an irrelevant environment variable to passive. If you don't do this, you'll get some warning along the lines of "Warning: Environment variable OMP_WAIT_POLICY=PASSIVE not set," which is strange because the default value *is* passive! This environment variable luckily won't affect your simulations...I just want to reduce the chance that an error, no matter how trivial, screws up my simulations.

The third line copies your simulation files onto each farm computer that you're using (Remember that test run on jlabl1 from Step 1 above?) The directory specified in this command is where you have all of your GPT simulation files saved (input file, fieldmap files, etc.). The "*" means copy *all* simulation files in this folder and the "." after it means "copy it here", "here" meaning to the farm computer. If you do not do this, the farm computer will have no files to run your simulations on and you will receive a blank data file (This happened to me many times. Don't be like me!). Note that these files are deleted after your run, so you have to use this command every time you submit your job.

The fourth line is the familiar GPT command line used to run a GPT simulation. I recommend setting the number of cores each GPT simulation uses using the -j option, just for simulation stability reasons. In my case, I am using 2 cores. I specify a different output file name within each GPT command (filename1.gdf, filename2.gdf, etc.). You don't technically have to do this...I'm just doing this to be safe. After specifying the input file, each job has different values for two variables used in the input file. In this case, the number of particles is fixed at 10000 (it could be different, though) and each job has its own specification for the (maximum) electron energy. I'd recommend specifying variables that you'd like to vary between jobs within this GPT command line instead of creating different input files for each simulation. (This would obviously be much less tedious if GPT were allowed to talk to Auger, in which case one can use an MR file together with MPIRUN to scan through a set of variables and send out each job to farm computers using one command line instead of many job-specification lines. Of course, we can't ALL have nice things...)

4.1.3 Job Specifications

The next few lines specify each job that you'd like to run on the farm. In this example, I'm running a simulation of 10000 electrons at the T-Gun for two different electron energies (in this case, 100keV and 200keV). Each job is specified in the xml file between the two tags <Job> and </Job>. Any commands that you put in between these two tags gives specifications *for that job only.*

The first two lines in each job-specification are specifying where the standard output and standard error (stdout & stderr) texts are written. I highly recommend you include these for each job, as this is (to my knowledge), the only way to know for certain whether or not your GPT run has completed successfully. Any errors that would normally be given in the output window in GPTwin are written to stderr. Be sure to give different names for your stdout and stderr files for each job so that you can differentiate them.

The next few lines between <Command> and </Command> are the commands given for each job, which are described in the previous section.

Each job also has an <Output> command that allows you to name the resulting GDF data file and give it a destination (outside of the farm computer). Note that in the GPT command line, I've specified the output file as "filename1.gdf" or "filename2.gdf". This is really a dummy name that will be changed by the <Output> commands. Each output command takes the file name and puts it in the specified directory with the specified name. What this means is that even though the GPT command line tells each farm computer to call its subsequent GDF file the same name (in this case filename.gdf), these GDF datafiles get returned to me by Auger under the names that I specify in the <Output> command, which is how I can distinguish them. Thus, it doesn't really matter what you call the result file in the GPT command line.

Of course, you can run as many jobs as you'd like/need by including more jobs in the xml file. If the number of jobs is on the order of, say, more than 10, I'd recommend using ForEach, which will use a For loop to create many jobs (see https://scicomp.jlab.org/docs/xml_command_file for more info on this). Obviously I wouldn't start here, but once you get the hang of running simple jobs on the farm, you can try using ForEach to simplify your xml file instead of explicitly writing many job commands. It'll take some clever thinking and exact knowledge of what you're doing, but it's better and much less tedious to do this than to write hundreds of lines in your xml file. for hundreds of jobs.

One last note: make sure your job specifications and global specifications are within the <Request> and </Request> tags. This is important, as your xml file won't get sent to the farm without it.

5 Step 4: Well, we'll see if you're ready

Once you've created your xml file, put it in the same place as your simulation files. Now you are ready to test run your simulations on the farm. First, log onto any JLab computer (either using a terminal on a JLab Linux workstation or using PuTTY on Windows machines). If you use PuTTY, you'll start by logging on to login.jlab.org with your JLab CUE username and password. Once you are logged into a JLab computer, type

```
ssh ifarm.jlab.org
```

and enter your password (again!). This will log you into one of the two interactive nodes: either ifarm1401 or ifarm1402 (it doesn't matter which one you use). You'll have access to all network drives that you would normally have on a Windows machine, except that they'll have different names (see https://cc.jlab.org/desktopsupport for more info). If this is your first time running on the farm, you need to get a network certificate by typing in:

```
/site/bin/jcert -create
```

From here, if you haven't already done so, go to your folder on your project's directory here:

```
/u/group/<projectname>/<yourname>/
```

Create a folder for your farm simulation files and copy them there along with the xml file you've created. Once you've double checked your xml file for typos, you can submit your xml file to the farm by typing in

```
jsub -xml <filename>
```

Your xml file will then be parsed. If successful, it will return your job number and your batch will be submitted to the farm. If there are any syntax errors in your xml file, it will let you know then. What it will ***NOT*** do is tell you any errors or warnings that GPT gives (like it would in the output window of your GPT batch file when run on Windows or within the terminal environment on Linux machines or PuTTY). Thus, it is very important that you are absolutely sure that your GPT program will run correctly and without any non-trivial errors or warnings.

6 Step 5: Thought so...

If you've successfully created an xml file, sent multiple jobs to the farm and got back non-empty datafiles, then congrats! You've done it! You are ready to use the farm. From here, I'd recommend reading https://scicomp.jlab.org/docs/desc_xml_tags to help you customize your xml file. If you can submit a successful job to the farm, you should be able to figure out the rest to meet your needs. Good luck!