Monte Carlo Radiation Transfer (MCRaT) User's Guide

Tyler Parsotan

Contents

1 Introduction **2**

2 Getting Started **2**

2.1 Required Libraries . . . 2

2.2 Downloading and Compiling MCRaT . . . 3

2.3 The MCRaT Parameter File . . . 4

2.4 Understanding MCRaT's Parallelism . . . 5

2.5 Running MCRaT . . . 5

2.6 MCRaT Output . . . 6

2.7 Restarting MCRaT . . . 7

2.7.1 Using The mcrat_msp.sh Shell Script . . . 8

2.8 Using Hybrid-Parellelization . . . 8

2.9 Post-Processing MCRaT Data . . . 8

3 The MCRaT Code **9**

3.1 Assumptions about the RHD Simulation . . . 9

3.2 Algorithm . . . 9

3.3 Output . . . 12

3.4 Post-processing Algorithm . . . 14

4 Physics Included in MCRaT **15**

4.1 Injecting Photons . . . 15

4.2 Scattering Photons . . . 16

4.3 Polarization . . . 16

1 Introduction

The Monte Carlo Radiation Transfer (MCRaT; pronounced *Em-Cee-Rat*) code is a next generation radiation transfer code that can be used to analyze the radiation signature expected from astrophysical outflows. The code is written in C and uses the Message Passing Interface (MPI) library for inter-process communication, the Open Multi-Processor (OpenMP) library for intra-node communication, the GNU Scientific Library, and the HDF5 library for parallel I/O.

MCRaT injects photons in a FLASH simulation and individually propagates and compton scatters the photons through the fluid until the end of the simulation. This process of injection and propagating occurs for a user specified number of times until there are no more photons to be injected. Users can then construct light curves and spectra with the MCRaT calculated results. The FLASH simulation used in this version of MCRaT must be in 2D, however, the photon propagation and scattering is done in 3D by assuming cylindrical symmetry.

The code was initially written in python by Dr. Davide Lazzati as a proof of concept. The code was then translated into C by Tyler Parsotan and made to use the OpenMP, MPI, HDF5, and GNU Scientific libraries. MCRaT is highly parallelized and is easy to set up and use.

2 Getting Started

This section describes how to get the MCRaT code as well as the required libraries necessary to compile MCRaT. This section will also cover the necessary steps to run the compiled MCRaT code.

2.1 Required Libraries

MCRaT requires a number of libraries in order to compile properly. These are:

- MPI
- OpenMP
- GNU Scientific Library (GSL)
- HDF5

The MPI library can be an any implementation such as OpenMPI (open-mpi.org) or one that is specific to a given supercomputer. Following the steps to install the library should allow it to work properly for compiling MCRaT. The OpenMP library is automatically included with Intel and GNU compilers. The GNU Scientific Library (GSL) can be downloaded from gnu.org/software/gsl/. Following the included instructions should install GSL in the default directory for it to be found by the compiler. The HDF5 library can be found at support.hdfgroup.org/HDF5/ where version 1.10 or greater should be downloaded and installed. The parallel portion of HDF5 needs to also be installed which is included in the instructions for installing HDF5.

2.2 Downloading and Compiling MCRaT

Once the required libraries have been installed, the current MCRaT code can be downloaded from github.com/lazzati-astro/MCRaT/. There is a Makefile which makes compiling the code significantly easier. In the directory that all the MCRaT files were downloaded into, simply typing `make` will compile the MCRaT code into a binary file called `MCRAT`. Typing `make MERGE` will compile the included MCRaT post-processing code, which will be covered in later in this section, into a binary file called `MERGE`.

The Makefile has to be configured for a variety of specifics related to the user's system. The `CC` option in the makefile can be changed to the user's compiler of choice such as gcc or icc. The `HDF_INSTALL` line is for the directory that the HDF5 library is installed in. The `EXTLIB` line is for the directories of any external libraries that the user may have. For example, if the GSL library cannot be found by the compiler, the user can put the directory of the library in this line of the Makefile in order to properly compile MCRaT. The `CFLAGS` feeds any compiler options to the compiler such as optimizations, and the switch to compile MCRaT with OpenMP capabilities, which is necessary to exploiting the hybrid parallelization available in the code. If the user is using gcc, the `-fopenmp` does not need to be changed, however, if the user is using icc, `-fopenmp` needs to be changed to `-qopenmp`. The `INCLUDE` line can be changed to include any `include/` directories to help the compiler, and the same if for the `LIBSHDF` in relation to any shared libraries. The `LIB`, `DEPS`, and `OBJ` lines should not need to be changed. Once the lines in the Makefile appropriately reflect the users configurations, typing `make` and `make MERGE` should properly compile the MCRaT code as well as its post-processing code, `MERGE`.

Once compilation is successful, the user will need to decide where MCRaT's output will go. The user needs to make a directory within the directory that contains the FLASH simulation frames. This subdirectory is where the parameter file for MCRaT must be placed. This is the same directory in which MCRaT will do all of its work, including making its own directories and output files within the aforementioned directories. Once this directory is created, the user must modify the `mcrat.h` file in order fo the code to know where to find all the FLASH files and the MCRaT parameter file. The constant `FILEPATH` should be modified with the directory of the flash simulation files while the constant `FILEROOT` should contain the name of the FLASH simulation files without any frame numbers. The constant `MC_PATH` should be the name of the directory, located within the `FILEPATH` directory, that holds the MCRaT parameter file named `mc.par`. An example file structure, with the values for each variable in MCRaT, is shown below:

```
/dir/to/FLASH/simulation/ (FILEPATH)
├── dir_with_MCRaT_parameter_file/ (MC_PATH)
│   └── mc.par
└── example_FLASH_file_name_0001 (FILEROOT=example_FLASH_file_name_)
```

The other variable that can be changed in MCRaT is `THISRUN` which should be set as "Science", if using MCRaT for a production run, or "Spherical"/"Cylindrical" which is for testing MCRaT with these types of outflows (which will be covered in a later section). Once all the variables have been appropriately changed, by replacing the strings appropriately in the `mcrat.c` file, the code needs to be recompiled.

Additionally, there are a number of switches that the user can specify to the program to tell it whether they want to consider the effects of polarization, with the `STOKES_SWITCH`, and one that specifies whether the user wants MCRaT to save the comoving 4 momenta of each photon, with the `COMV_SWITCH`. Both of these switches can be set to 0, which means do not save the comoving 4 momenta or save and consider the effects of polarization, or they can be set to 1, which tells the program to consider polarization and save the stokes parameter associated with each photon or save each photon's 4 momentum.

The last switch, `DIM_SWITCH`, tells MCRaT the dimensionality of the hydro simulation that is used. This switch can be set to "2D" or "3D". The 3D feature is still in development so the switch should be set to "2D" for now.

2.3 The MCRaT Parameter File

MCRaT requires the user to provide information in the form of a parameter file named `mc.par` which should be placed in the appropriate directory, as is described in the previous section. An example `mc.par` file, named `sample_mc.par` is included in the MCRaT github repository. An example with a description of each line, is as follows:

1e12	The 1st line is the domain of the FLASH x axis
1e13	The 2nd line is the domain of the FLASH y axis
5	The 3rd line is the the FLASH simulation number of frames per second
200	The 4th line is the starting frame number for angles $\leq 2^\circ$
200	The 5th line is the starting frame number for angles $> 2^\circ$
3000	The 6th line is the last FLASH frame number
124	The 7th line is the number of frames in which photons are injected for angles $\leq 2^\circ$
124	The 8th line is the number of frames in which photons are injected for angles $> 2^\circ$
8e11	The 9th line is the radius at which the photons will be injected for angles $\leq 2^\circ$
9e11	The 10th line is the radius at which the photons will be injected for angles $> 2^\circ$
0	The 11th line is the minimum angle in degrees for which photons will be injected
20	The 12th line is the maximum angle in degrees for which photons will be injected
4	The 13th line is the change in angular resolution, this partly dictates how MCRaT will parallelize the simulation
5e50	The 14th line is the users initial guess of the weight of each injected photon for angles $\leq 2^\circ$
5e50	The 15th line is the users initial guess of the weight of each injected photon for angles $> 2^\circ$
1000	The 16th line is the minimum number of photons the user wants injected in each frame
3000	The 17th line is the maximum number of photons the user wants injected in each frame
b	The 18th line is the spectrum of the injected photons. "b"=Blackbody and "w"=Wein
r	The 19th line is to tell MCRaT to start a new simulation, "r", or continue an old one, "c"

There are values related to photons injected within 2° and outside of 2° because at this angle, the outflow may become significantly denser (as occurs in Gamma Ray Bursts) and the user may want to increase their radius of injection in order to speed up the computation. In order to match the time in which photon injection will occur at these larger angles means changing the parameters that play a role in the value of the simulation time when the photons begin being injected.

Lines 4 combined with line 9 dictate what time in the simulation you begin injecting photons. This is calculated as

$$t_{\text{start}} = \frac{\mathrm{f_{start}}}{\mathrm{fps}} - \frac{R_{\text{inj}}}{c}$$

where $\mathrm{f_{start}}$ is the value in line 4, fps is the frames per second which is the value in line 3, R_{inj} is the radius at which photons are injected, or the value in line 9, and c is the speed of light. Building on the above formula, the time in which photons stop being injected are calculated as

$$t_{\text{end}} = \frac{\mathrm{f_{start}} + \mathrm{df}}{\mathrm{fps}} - \frac{R_{\text{inj}}}{c}$$

where df is the number of frames in which photons are injected, which is line 7. These calculations are the same for photons injected at angles $> 2^\circ$, except you use the `mc.par` lines that are applicable to those photons in the calculations.

Line 19 allows the user to continue the MCRaT code from a previous point, if the value is set to "c", or it allows the user to restart (by erasing the full working directory of MCRaT) or start a simulation by setting it to "r". **This is an important parameter to check to prevent accidentally erasing work that has already been done.**

Line 13 should be a value that divides evenly into the angle range provided by lines 12 and 11. In the above example (20-0)/4=5. This is important for MCRaT's parallelism. Another number that MCRaT uses to break up the problem is lines 7 and 8. The way that MCRaT parallelizes the problem will be shown in the next section.

2.4 Understanding MCRaT's Parallelism

The first step that MCRaT takes in breaking up the problem is breaking it up into pieces based on the range of angles that photons will be injected into, which the user specifies in lines 11 - 13 in the `mc.par` file. As the example `mc.par` above shows, MCRaT will break up the problem into (20-0)/4=5 pieces initially. Then, for each angle range ($0^\circ-4^\circ$, $4^\circ-8^\circ$, etc.) the code will then break up the problem based upon how many times photons will be injected, which is 1 plus the values in lines 7 and 8 in the `mc.par` file; this is because the photons are injected in the frames $[\mathrm{f_{start}}, \mathrm{f_{start}}+1, ..., \mathrm{f_{start}}+\mathrm{df}-1, \mathrm{f_{start}}+\mathrm{df}]$, which gives us this n+1 behavior. These pieces of information are important for understanding how the processes are divided with MCRaT. An example will be given in the next section.

2.5 Running MCRaT

In this section we show how to run MCRaT and allocate the appropriate number of cores in order to optimize MCRaT's parallelism. We will use the example `mc.par` file in Section 2.3 for this section.

Running the MCRaT code is as simple as doing `mpiexec -np N ./MCRAT` where N is the total number of MPI processes that will be created. The OpenMP parallelism is controlled by setting the OpenMP environment variable `OMP_NUM_THREADS`. We recommended that `OMP_NUM_THREADS` be set to 1 for a newly started MCRaT simulation, which essentially turns off the OpenMP parallelism.

The total number of MPI processes, N, is initially divided by the set of angles that photons will be injected into. For the example `mc.par`, there are $(20 - 0)/4 = 5$ sets of angles that will be considered which leads to the N MPI process initially being divided by 5. The $N_\theta = N/5$ MPI processes will then be used to inject photons in the user specified frames in parallel. The example `mc.par` file shows that there will be 124+1 photon injections, as was described in the last section. Thus, the number of frames that each MPI process will inject photons into becomes $\mathrm{f_{inj}} = (124 + 1)/N_\theta$. If we set $N = 125$, $N_\theta = 25$ and the number of frames that each MPI process will inject photons into becomes 5. Alternatively, if we want each MPI process to only inject photons into one frame, $N_\theta = 125$ which means that $N = 625$. Calculating N by following these examples, shows how the problem is broken up between angles and photons injections. If the number of MPI processes available for the user to use does not divide well into the size of the problem, in terms of angle ranges and frames in which photons are injected, MCRaT will prioritize breaking up the problem evenly based on the set of angles. Thus, N should at least be evenly divisible by the number of angle ranges specified by the `mc.par`; Once N_θ is calculated, MCRaT will distribute the number of frames in which photons will be injected accordingly to calculate $\mathrm{f_{inj}}$.

Another option, when dealing with a limited number of available cores, is running MCRaT on a small portion of the problem. For example instead of doing 5 sets of angles at 4° intervals all at once, simply run 1 set of angles from $0° - 4°$, and when that simulation is complete run the second set from $4° - 8°$, etc. The increased parallelism in decreasing $\mathrm{f_{inj}}$ by using this method may speed up the simulation compared to running the whole problem at once. This is also advantageous due to the fact that as the MPI processes begin to complete their portion of the simulation, the user may restart MCRaT using only the appropriate number of MPI processes left that have calculations left to do. Thus, this feature lets a given MCRaT simulation (e.g. $0° - 4°$) use less MPI processes and resources over time, allowing another full simulation (e.g. $4° - 8°$) to start running using the rest of the available resources. We will go over this feature in a later section.

2.6 MCRaT Output

Once MCRaT is run, it will create subdirectories dedicated to the calculations and results of each angle range. Using the example `mc.par` file, the directory tree will now look like:

We have not included the subdirectories extending up to 20° for brevity, but those other 2 directories would also be created. Within each angle subdirectory, each MPI process, labeled process i, creates 3 files. The mc_output_i.log file prints exactly what the MCRaT code is doing for that given MPI process. This allows us to monitor the progress of the MCRaT simulation. This can be easily done by using Linux commands such as `less mc_output_`i`.log`. The mc_proc_i.h5 is a HDF5 file that contains all of the calculation results of the MCRaT MPI process. The mc_chkpt_i.dat file contains the necessary data for each MPI process to continue from the last point if MCRaT is interrupted for any reason. This is covered in the next section.

When all of the MPI processes in a given angle subdirectory have completed their calculations, they will produce HDF5 files with the results of the calculations for each file in the simulation starting with the frame specified in line 4 (or 5, depending on the angle range under consideration) in the `mc.par` file up to the frame specified in line 6 in the `mc.par` file. These files will be named `mcdata_`f`.h5` where f is the frame number of the hydrodynamic simulation.

```
/dir/to/FLASH/simulation/
├──example_FLASH_file_name_0001
└──dir_with_MCRaT_parameter_file/
   ├──mc.par
   ├──0.0-4.0/
   │  ├──mc_proc_i.h5
   │  ├──mc_chkpt_i.dat
   │  └──mc_output_i.log
   ├──4.0-8.0/
   │  ├──mc_proc_i.h5
   │  ├──mc_chkpt_i.dat
   │  └──mc_output_i.log
   └──8.0-12.0/
      ├──mc_proc_i.h5
      ├──mc_chkpt_i.dat
      └──mc_output_i.log
```

2.7 Restarting MCRaT

If the MCRaT code gets interrupted by running out of allocated time on a cluster or any other type of issue, the code is easily continued and it may also use less MPI processes when it is restart based on the number of MPI processes in the original simulation run that still have calculations to complete.

In order to ensure that the code knows to continue a MCRaT simulation, the 19th line in the `mc.par` file must be set to “c”. **This is important to check since not changing this parameter means that all of the simulation progress will be erased and a completely new simulation will be started, thus nullifying the attempt to save time and resources by continuing a MCRaT simulation.**

Once this parameter has been changed, the user needs to identify how many total MPI processes have completed their calculations. This is done by running `tail -n 1 dir_with_MCRaT_parameter_file/angle_dir/mc_output_* |grep "completed" |wc -l` in each angle range directory created by MCRaT. Add all the printed values and subtract from the original number of MPI processes, N, that the MCRaT simulation was started with. This is the number that has to replace N in the new run of MCRaT. From here, MCRaT automatically distributes the given number of processes among the MPI processes in each angle subdirectory that still need to run calculations.

This feature of only continuing the MCRaT simulation with the necessary number of MPI processes means that as the simulation progresses, fewer resources will be used which is invaluable for running jobs in a pipeline type of fashion and for conserving resources such as SBU allocations.

If a certain process is also taking a long time to conduct its simulation it is possible to modify the mc.par file to specify that the simulation should inject fewer photons, to speed up the calculation and force the process's simulation portion to restart. To force the restart delete that processes mc_chkpt file and it's mc_proc file. This is exactly what the `mcrat_msp.sh` shell script does.

2.7.1 Using The mcrat_msp.sh Shell Script

The mcrat_msp.sh shell script determines which MCRaT processes have a very large number of scatterings and allows the user to delete those process' mc_chkpt and mc_proc files, if the process' photons have undergone a number of scatterings greater than some user specified number of scatterings, which would then force a new restart for those processes with less injected photons. It also allows the user to refine the mc.par file to set a new number of injected photons for these same processes that will be restarted the next time the user starts mcrat with the 'c' flag to continue a simulation. **This is important to change once MCRaT has been restarted and the photons injected. If it isn't other processes, which don't have so many scatterings, will inject photons (if applicable, which it should be on a relatively small cluster where the user can't assign one photon injection frame to each process) within the new range specified by the mc.par file. To do this, once the simulation restarts and is running smoothly, the user has to end it, modify the minimum and maximum range of injected photons to the original numbers, and restart the simulation once again. (We are currently working on a better way of implementing this feature.) The user must remember to reset the restart flag in the mc.par file to 'c' otherwise all the simulation data files will be erased and all progress lost.** Opening the mcrat_msp.sh file in a text editor will bring up different flags for the shell script and give examples of how to properly use the function.

2.8 Using Hybrid-Parellelization

As was mentioned earlier, it is not advised to enable hybrid parallelization at the start of a new MCRaT simulation, especially if limited resource availability is an issue. This is due to the fact that the OpenMP threads may interfere with one another and neighboring MPI processes and their OpenMP threads, thus slowing the overall code. To prevent this, set `OMP_NUM_THREADS` to 1.

If MCRaT is being used to continue a simulation, there is a good chance that some of the MPI processes have finished and the remaining MPI processes are running slower than the rest due to a variety of factors, which will be covered later. Thsi is where hybrid parallelization helps to ensure that MCRaT completes its calculations is a reasonable amount of time. These MPI processes can be sped up by using the OpenMP threads. The number of threads per MPI process can be set with the `OMP_NUM_THREADS` parameter. Even in this case, it is important to make sure that threads are not interfering with one another and undermining any potential performance gain.

2.9 Post-Processing MCRaT Data

If the user is simply running running a simulation with one angle range under consideration, there is no post-pocessing to be done. The MCRaT produced `mcdata_`f`.h5` files contain all the information for the photons (which will be covered in the next section).

If not we need to be able to merge all the simulation results for photons injected in all angle ranges; following the example `mc.par` file, we need to merge the data in angles $0° - 4°$, $4° - 8°$, etc. This is done is with the MERGE code that is included with MCRaT. To run merge, the user simply types `mpiexec -np N ./MERGE`. In this case N needs to be a multiple of the number of subdirectories the MCRaT code has created and it has to be greater than or equal to the number of subdirectories; of cource the larger the number of MPI processes used, the less work each process has to do, making the post-processing code faster. In the example `mc.par` file there are 5 directories created, so N has to be a multiple of 5 in this case. The minimum value of N can be 5; however, since the MERGE code is also parallelized with OpenMPI, the larger the number of processes used, the less time it takes to merge all of the MCRaT data. If $N = 10$, 5 processes will be allocated to merge the data from the first half of the frames from 200 to 3000, as specified in the `mc.par` file, and the other 5 will merge the data from the last half of the range of frames.

The merged data is placed into another folder that the code creates named `ALL_DATA/`.

The standard work flow for MCRaT is as follows:

1. Compile the MCRaT and MERGE codes
2. Write the `mc.par` file in the correct directory
3. Run MCRaT
4. use MERGE to post-process the MCRaT results

This work flow can be practiced using a spherical or cylindrical outflow problem that is included in MCRaT. These simulations can be run using any FLASH simulation because MCRaT reads in the data from the FLASH simulation and over writes it (in memory, not in the FLASH file) with the correct values assuming a spherical or cylindrical outflow.

In order to run a spherical outflow simulation simply change the `THISRUN` parameter in `mcrat.c` file to `"Spherical"`. Then recompile the code and run it as we outlined previously. In order to run a cylindrical outflow simulation, change the `THISRUN` parameter to `"Cylindrical"`. These test simulations allow the user to become acquainted with MCRaT and test that MCRaT produces the expected results for these type of outflows.

3 The MCRaT Code

This section describes the algorithm and output of MCRaT. It will cover the steps that MCRaT takes to inject and scatter photons.

3.1 Assumptions about the RHD Simulation

MCRaT has to make some assumptions about the output of the RHD simulation. These assumptions are:

1. The velocity in the RHD simulation is normalized by the speed of light, c
2. The pressure is normalized by c^2

3.2 Algorithm

MCRaT first reads the data from the `mc.par` file and then appropriately divides the MPI processes based on the number of angle ranges, if starting a new simulation, or, for continuing a simulation, reads the mc_chkpt file in each directory to determine which MPI processes still have work to complete and then distributes the MPI processes.

Each MPI process is assigned a range of frames in which the process injects photons into and then propagates through all the FLASH simulation frames until the last FLASH simulation frame, which is specified in the `mc.par` file. The photons have a `photon` data structure that fully describes each photon. This includes the photon's lab frame 4-momenta, the x, y, and z coordinates of the photon, the Stokes parameters of the photon, the weight of the photon, the number of times the photon has scattered, and the index of the FLASH fluid element that the photon is located within.

In order to inject the photons, MCRaT reads in the first FLASH frame in which the photons will be injected in and chooses FLASH elements that have radii, r, such that $R_{\text{inj}} - c/(2 * \text{fps}) < r < R_{\text{inj}} + c/(2 * \text{fps})$, where c is the speed of light, R_{inj} is the injection radius and fps is the FLASH simulation frames per second, both specified in `mc.par`. These FLASH elements also have to be within the angle range that the MPI process is injecting photons within. Thus, we choose a slab of FLASH elements to inject photons into. The number of photons that would be expected from the i^{th} FLASH element, n_i, is calculated using the energy density of the element and by assuming a weight for each MCRaT photon which is taken from the `mc.par` file. This is further explained in Section 4.1. Once the expected number of photons is calculated, the actual number is acquired by assuming that the photon injection is random, which means that we acquire the actual number of photons injected into the i^{th} FLASH element, N_i by randomly drawing from a Poisson distribution with a mean of n_i. The total number of photons that would be injected across all the chosen FLASH elements becomes $N = \sum N_i$. If N is within the minimum and maximum number of photons that the user wants, which is specified in the `mc.par` file, the calculation for the number of photons is complete. If not, then the code continues to recalculates n_i using a new weight for each photon, then recalculates N_i and N, and after rechecks so see if N is within the user specified number of photons again. This loop is exited when N falls within the user specified number of desired photons. If the code needs to inject more photons to meet this condition, it will adjust the weights to be smaller and vice versa.

Once N and N_i have been determined, the N_i photons are placed at the center of the i^{th} FLASH element, with a random azimuthal angle uniformly distributed between 0 and 2π (since we assume cylindrical symmetry for the 2D FLASH simulation), and they are given an energy based on the comoving temperature of the FLASH element, thus the photons are injected in the comoving frame fo the outflow. If the user specified that the photons should be injected with a Wien spectrum then the frequencies, ν, are assigned to the N_i photons by sampling from a Wien spectrum defined by the i^{th} fluid element's comoving temperature. The same thing is done with a Blackbody spectrum if the user specified that the photons should be injected as a Blackbody spectrum. This sampling is done by using the acceptance-rejection Monte Carlo sampling of the spectra.

For their 4-momenta, the photons are assigned a random azimuthal angle, ϕ, from a uniform distribution between 0 and 2π, as well as a polar angle, which is calculated as $\theta = \arccos(2\xi - 1)$, where ξ is a random number drawn from a uniform distribution. The i^{th} injected photon 4-momentum becomes:

$$p_i^{\mu} = \frac{h\nu_i}{c} \begin{pmatrix} 1 \\ \sin(\theta_i)\cos(\phi_i) \\ \sin(\theta_i)\sin(\phi_i) \\ \cos(\theta_i) \end{pmatrix}$$

After the 4-momenta are set, the photons are boosted into the lab (observer) frame using the velocity of the fluid element that the photons are located within. The lab frame 4-momenta are saves to each photon's `photon` data structure, as well as the photon's position and weight. The number if scatterings is initialized to 0 and the stokes parameters are initialized to 0 except for s0 which is set to 1.

Once the photons have been injected into the simulation and their 4-momenta have been determined, the scattering process through the hydrodynamic (HD) simulation occurs. MCRaT reads in a HD simulation frame, starting from the frame in which the photons were initially injected into up until the last HD simulation frame available, and determines which HD fluid elements each photon is in and then calculates the mean free path of each photon. In order to reduce memory and optimize the code, MCRaT determines the minimum and maximum radii of the photons, $R_{\rm min}$ and $R_{\rm max}$ respectively, and selects a slab of FLASH fluid elements with radii, r_i such that

$$R_{\rm min} - \alpha \frac{c}{\rm fps} < r_i < R_{\rm max} + \alpha \frac{c}{\rm fps}$$

where α is a multiplicative factor that can be increased to ensure that the number of chosen FLASH elements is greater than 0. The code finds which fluid element each photon is located within, and saves the index to the `photon` data structure. This data field only gets updated if the photon propagates outside of the fluid element. If this happens, then the code finds and saves the index of the new fluid element that the photon is located within. Sometimes, the code is not able to find a fluid block that a given photon is located within. This may happen due to a photon propagating outside of the range of radii of the chosen slab of FLASH element, and is very rare. To deal with this, we set the mean free path, l, to an arbitrary large value to ensure that the photon doesn't scatter, which would inaccurately change its energy. If the fluid element is correctly identified, the mean free path is calculated as

$$l = -\frac{m_p}{\rho \sigma_{\rm T}(1-\beta\cos\theta_{fp})} \ln \xi$$

where we have inverted the exponential probability distribution for a photon scattering in a medium. Here, ξ is a random number uniformly distributed between 0 and 1, m_p is the proton mass, $\sigma_{\rm T}$ is the Thompson cross section, ρ is the lab frame density of the FLASH fluid element that the photon is located within, β is the velocity of the fluid element normalized by the speed of light (this is already done in the FLASH code), and θ_{fp} is the angle between the fluid element's and the photon's velocity vectors. From l, the time for each photon to scatter is calculates as $t_{\rm s} = l/c$, where c is the speed of light. The photon with the smallest t_s is assumed to scatter first, however that is not necessarily true since we use the full Klein-Nishina cross section in the electron rest frame to actually determine if the scattering occurs. The positions of all the photons are advanced by the time for the photon of interest to scatter, t_s.

In determining whether a photon will scatter or not, we first boost the chosen photon with the smallest t_s to the fluid rest frame and produce an electron by sampling the Maxwell-Boltzmann or Maxwell-Juttner distributions for non-relativistic and relativistic gasses. The comoving temperature, T'_i, of the fluid element is used to distinguish between which distribution is used. If $T'_i \geq 1 \times 10^7$ K the Maxwell-Juttner distribution is used, otherwise the Maxwell-Boltzmann distribution is used. When the Maxwell-Boltzmann distribution is used we choose 3 random x, y and z velocities from gaussian distributions with a standard deviation of $k_B T'_i/m_e$, where k_B is the Boltzmann constant and m_e is the mass of the electron. The velocities are normalized by the speed of light and used to calculate the Lorentz factor of the electron in the fluid frame. To get the full 4-mometum of the electron, the azimuthal angle of the electron's direction fo travel is drawn from a uniform distribution between 0 and 2π and the polar angle is acquired by acceptance-rejection sampling of the polar angle distribution $(1 - \beta \cos(\theta)) \sin(\theta)$. We then perform rotations based on the velocity vector of the photon that will scatter to ensure that the electron velocity is oriented relative to the photon's velocity.

Once the electron has been initialized, the photon and the electron are boosted into the electron rest frame. Then the photon velocity vector is rotated such that it is directly along the x-axis with the stationary electron at the origin of the coordinate system. We then use the Klein Nishina (KN) cross section including the effects of polarization, σ_{KN}, to determine if the photon will actually scatter. The ratio $\sigma_{\mathrm{KN}}/\sigma_T$ is calculated and a random number, ξ, from a uniform distribution between 0 and 1 is drawn. If $\xi > \sigma_{\mathrm{KN}}/\sigma_T$ the photon does not scatter, otherwise it does scatter with the electron.

If the photons scatters with the electron, a random θ and ϕ are drawn from their respective distributions that are derived from the KN cross section including polarization. The photon's new 4-momentum and stokes parameters are calculated and then the scattered photon is rotated to its original orientation with respect to the previously stationary electron. The photon is then Lorentz boosted twice, back to the lab frame. The process of keeping track of the Stokes parameters and calculating them will be covered in subsection 4.3.

If the photon does not scatter, then the next photon with the second shortest time to scatter is chosen and the process to create an electron and determine if the photon will scatter repeats. This loop continues until a photon scatters or all of the photons have been determined to not scatter.

While this loop is occurring, the photons in the simulation are being propagated incrementally by the change in scattering time for each photon until the time for the next FLASH simulation frame is reached. The same thing occurs when a photon does scatter, the photons get propagated through space using the scattering time and the process of determining each photons' time to scatter, etc. repeats until the time corresponding to the next FLASH frame is reached.

Once the time limit for the next FLASH fluid frame is reached, MCRaT saves all the photons' information in the `photon` data structure into the HDF5 file under a group with the same FLASH fluid frame number that the photons were being scattered in. Once the data is written, a checkpoint file is saved for MCRaT to be able to continue the simulation if it were interrupted. Then, the next simulation frame is loaded and the process of determining photons' time to scatter, actual probability of scattering and resulting scattered 4-momenta until the next HD simulation frame time is reached. This repeats until the last HD simulation frame is reached.

If the MPI process has more photons to inject, then the process of injecting the photons into the next frame and propagating those photons through the FLASH simulation repeats. If all of the MPI processes have finished injecting and propagating photons for the FLASH simulation frames assigned to them, the code combines all of the photons' data from each MPI process into `mcdata_`i`.h5` files where i is the FLASH simulation frame number. The output is covered in the next section.

3.3 Output

Each MCRaT MPI process outputs 3 files. These files are named `mc_proc_`i`.h5`, `mc_chkpt_`i`.dat`, and `mc_output_`i`.log` where i is the number for the MPI process producing those files. As the end of the simulation, MCRaT produces the actual data files named `mcdata_`f`.h5` where f is the FLASH simulation file frame.

The `mc_chkpt_`i`.dat` file is a binary file that contains the data from the MPI process that allows it to continue the simulation if MCRaT gets interrupted for any reason. This information is: the MPI process' range of FLASH frames to inject photons into, the frame that the current set of photons were injected in, the frame in which the photons were recently scattered in, the current time of the MCRaT simulation, the number of photons that the code is keeping track of, and the `photon` data structure for each photon in the simulation.

The `mc_proc_`i`.h5` file is a HDF5 file that contains all the data for the photons that the i^{th} MPI process injects and propagates through the FLASH simulation. The data is organized by the FLASH simulation frame which constitutes a HDF5 group. Within the group, there are the following datasets, depending on if `STOKES_SWITCH` or `COMV_SWITCH` id set to 1 or 0.

- Weight - The photon weight for any photons that were injected into the frame. If there were no photons injected, then this data set does not exist.
- P0 - the energy of the photons
- P1 - the magnitude of the photons' 4-momentum in the x direction
- P2 - the magnitude of the photons' 4-momentum in the y direction
- P3 - the magnitude of the photons' 4-momentum in the z direction
- COMV_P0 - the comoving energy of the photons
- COMV_P1 - the magnitude of the photons' comoving 4-momentum in the x direction
- COMV_P2 - the magnitude of the photons' comoving 4-momentum in the y direction
- COMV_P3 - the magnitude of the photons' comoving 4-momentum in the z direction
- R0 - the x position of the photons
- R1 - the y position of the photons
- R2 - the z position of the photons
- S0 - the I Stokes parameter for the photons
- S1 - the Q Stokes parameter for the photons
- S2 - the U Stokes parameter for the photons
- S3 - the V Stokes parameter for the photons

- NS - the cumulative number of scattering each photon has undergone

These data sets include information for all of the photons after they were scattered through the given FLASH simulation frame. The order of the data is based on the chronological order in which each set of photons were injected. For the example `mc.par` file in subsection 2.3, if the user uses N MPI processes such that each process injects photons into 2 frames, the `mc_proc_0.h5` file would look like:

```
mc_proc_0.h5
 ├── 200
 │    ├── Weights
 │    ├── P0
 │    ├── :
 │    └── NS
 ├── 201
 │    ├── Weights
 │    ├── P0
 │    ├── :
 │    └── NS
 └── 202
      ├── P0
      ├── :
      └── NS
```

where all the datasets in group 200 would have only n_0 elements, where n_0 is the number of photons first injected into FLASH frame 200. If we let n_1 be the number of photons injected in frame 201, the dataset Weight in group 201 will have n_1 elements while the rest of the datasets will have n_0 elements, corresponding to the information for the photons first injected in frame 200 that are propagated through frame 201, *followed by* n_1 elements for the newly injected photons in frame 201.

The last file produced by the MCRaT MPI processes are named `mc_output_`i`.log`. These are files that output the progress of the given process, allowing the user to keep track of how far along MCRaT is in the radiation transfer simulation and how much more work the code has left to complete. This file contains the number of scatterings that the code completes in increments of 1000 scatterings as well as the average photon energy in units of ergs, the comoving temperature, in Kelvin, of the FLASH fluid element that was used in the most recent scattering, and the most recent time step for a photon to scatter and the current time in the simulation. When all the scatterings are completed for a given frame, the code prints out the average cumulative number of scatterings that each photon has undergone, as well as the the maximum and minimum number of scatterings for the photons.The code also prints out the average radius of the MCRaT photons, the simulation time and the frames left to inject photons into and the last FLASH simulation frame.

When the MCRaT MPI process has completed the Monte Carlo scattering portion of the code it will print: “Process i has completed the MC calculation.” where i is the ID of the MPI process. When all of the MPI processes are complete with the Monte Carlo scattering they will collectively produce the `mcdata_`f`.h5` files which contain the merged data from each MPI process within a given angle range.

These `mcdata_`f`.h5` files are formatted in the same way that the HDF5 groups are formatted in the `mc_proc_`i`.h5` files. The only exception is that there is no `Weight` dataset. All of the photon weights are saved to a file named `mcdata_PW.h5` which has the weight do each of the photons in their chronological order of injection.

3.4 Post-processing Algorithm

When the user has completed running MCRaT for each angle range that they are interested in, the data for every simulated photon gets merged into a new directory called `ALL_DATA`. the MERGE code that is included with the download of MCRaT uses parallel HDF5 I/O to speed up this post-processing.

The code takes the MPI processes and distributes them evenly among the MCRaT produced angle subdirectories. If there are n_θ angle subdirectories, the MPI processes are grouped into sets of n_θ, with each MPI process operating withing a given subdirectory. Then, the code distributes the FLASH simulation frames that need to be created among the groups of n_θ MPI processes. It also counts how many processes were originally used by MCRaT in each directory so if MCRaT is independently run 3 different times for 3 different ranges of angles, but with each of the 3 runs using a different number of MPI processes, the post-processing code will still work.

These processes open the `mc_proc_`i`.h5` files and tally how many photons will be merged to produce the final `mcdata_`f`.h5` file in the `ALL_DATA` directory. Using this information, the code constantly ensures that any files created by MERGE files that were created, before the code was terminated for any reason, are not corrupted; thus the MERGE code can be stopped and started again without worrying about corrupting data. The code iterates over the photons injected from first to last in the MCRaT simulation, by reading the data from the `mc_proc_`i`.h5` files. These photon data are then collectively appended to the final `mcdata_`f`.h5` file. The same process is used to produce the final merged `mcdata_PW.h5` file.

4 Physics Included in MCRaT

This section describes the physics of the MCRaT code injecting photons and scattering them through the FLASH simulation.

4.1 Injecting Photons

the code selects RHD fluid elements at a given radius and calculates the expected number of photons in each element as

$$n_i = \frac{\xi T_i^{'3} \Gamma_i}{w} dV_i \tag{1}$$

where ξ, the number density coefficient which is used to calculate the number density of photons as $n_\gamma = \xi T^{'3}$, is 20.29 for a Blackbody spectrum or 8.44 for a Wien spectrum, T_i' is the comoving temperature of the fluid element, Γ_i is the Lorentz factor of the element, w is the weighting factor of each injected photon, and dV_i is the volume of the fluid element.

The comoving temperature is calculated as

$$T_i' = \left(\frac{3p_i}{a}\right)^{\frac{1}{4}} \tag{2}$$

where p_i is the pressure of the fluid element, and a is the radiation density constant.

If the RHD simulations used are 2D axis-symmetric, we assume cylindrical symmetry and calculate $dV_i = 2\pi x_i dA_i$ where x_i is the distance of the fluid element from the y-axis of the simulation and dA_i is the area of the element.

Once the number of photons is calculated, we draw a random number from a Poisson distribution using n_i as the average value in order to get the actual number of photons that will be injected into the i^{th} RHD element, N_i. The weight, w, for a given set of injected photons can be adjusted accordingly in order to inject more or less photons into the simulation; this allows us to conserve energy and increase photon statistics for time resolved spectra. This weight is essentially how many actual LGRB photons each MCRaT photon represents.

The total number of photons that would be injected across all the chosen FLASH elements becomes $N = \sum N_i$. If N is within the minimum and maximum number of photons that the user wants, which is specified in the `mc.par` file, the calculation for the number of photons is complete. If not, then the code continues to recalculates n_i using a new weight for each photon, then recalculates N_i and N, and after rechecks so see if N is within the user specified number of photons again. This loop is exited when N falls within the user specified number of desired photons. If the code needs to inject more photons to meet this condition, it will adjust the weights to be smaller and vice versa.

Once N and N_i have been determined, the N_i photons are placed at the center of the i^{th} FLASH element, with a random azimuthal angle uniformly distributed between 0 and 2π (since we assume cylindrical symmetry for the 2D FLASH simulation), and they are given an energy based on the comoving temperature of the FLASH element, thus the photons are injected in the comoving frame of the outflow. If the user specified that the photons should be injected with a Wien spectrum then the frequencies, ν, are assigned to the N_i photons by sampling from a Wien spectrum defined by the i^{th} fluid element's comoving temperature. The same thing is done with a Blackbody spectrum if the user specified that the photons should be injected as a Blackbody spectrum. This sampling is done by using the acceptance-rejection Monte Carlo sampling of the spectra.

For their 4-momenta, the photons are assigned a random azimuthal angle, ϕ, from a uniform distribution between 0 and 2π, as well as a polar angle, which is calculated as $\theta = \arccos(2\xi - 1)$, where ξ is a random number drawn from a uniform distribution. The i^{th} injected photon 4-momentum becomes:

$$p_i^\mu = \frac{h\nu_i}{c}\begin{pmatrix} 1 \\ \sin(\theta_i)\cos(\phi_i) \\ \sin(\theta_i)\sin(\phi_i) \\ \cos(\theta_i) \end{pmatrix}$$

After the 4-momenta are set, the photons are boosted into the lab frame using the velocity of the fluid element that the photons are located within. The lab frame 4-momenta are saves to each photon's `photon` data structure, as well as the photon's position and weight. The number if scatterings is initialized to 0 and the stokes parameters are initialized to 0 except for s0 which is set to 1.

4.2 Scattering Photons

4.3 Polarization