Users' Guide
`Milagro` Version 5,
An Implicit Monte Carlo Code for Thermal Radiative Transfer

Thomas M. Evans[a], Gabriel Rockefeller[b], Allan Wollaber[b], Jeffery Densmore[c], Michael Buksas, Kelly Thompson[b], Todd Urbatsch[d], Seth R. Johnson[e], Scott Mosher[a], Ryan McClarren[f], Timothy Kelley[g], Paul Henning[h], Mathew Cleveland[i], Aimee Hungerford[j]

Author address:

[a]Oak Ridge National Laboratory, Oak Ridge, TN 37831

[b]Computational Physics and Methods Group (CCS-2), MS D413, PO Box 1663, Los Alamos, NM 87545
E-mail address: jayenne@lanl.gov

[c]Bettis Atomic Power Laboratory, West Mifflin, PA 15122

[d]XTD–5, MS T086, Los Alamos National Laboratory, Los Alamos, NM 87545

[e]University of Michigan, Ann Arbor, MI 48109

[f]Texas A & M University, College Station, TX 77840

[g]CCS–7, MS D413, Los Alamos National Laboratory, Los Alamos, NM 87545

[h]XCP–1, MS T086, Los Alamos National Laboratory, Los Alamos, NM 87545

[j]Lawrence Livermore National Laboratory, Livermore, CA 94550

[j]XTD–6, MS T085, Los Alamos National Laboratory, Los Alamos, NM 87545

LANL Report Designation: LA-UR pending.

ABSTRACT. This users' guide describes how to configure, make, and run `Milagro`, Version 5. `Milagro`-5 is an object-oriented, C++ code that performs radiative transfer using Fleck and Cummings' Implicit Monte Carlo (IMC) method. `Milagro`, a part of the `Jayenne` program, is a stand-alone driver code used as a methods research vehicle and to verify its underlying classes. These underlying classes are used to construct IMC packages for external customers. `Milagro`–5 represents a design overhaul that allows better parallelism and extensibility.

Contents

CHAPTER 1

Overview

1.1. Introduction

Milagro is a parallel, object-oriented, C++ code that performs radiative transfer simulations using Fleck and Cummings' Implicit Monte Carlo (IMC) method [1]. Milagro is templated on mesh type, where curvilinear geometries are modeled with representative truncated 3-D Cartesian geometries with reflecting boundaries. Mesh types include XYZ, RZ (RZWedge AMR Mesh), R (Sphyramid Mesh), and AMR XYZ. Also, a tetrahedral mesh type has been developed and verified but remains to be implemented into Milagro.

Milagro's IMC classes run on two parallel topologies, one where the mesh is fully replicated on each processor, and one where the mesh is decomposed among the processors.

In order to facilitate code reuse, we store our reusable classes in a transport library called Draco [2]. Sitting above Draco is the ClubIMC (component library used by IMC) library, which contains the mc component directory for general use for Monte Carlo calculations and the imc component directory for general use with Implicit Monte Carlo (IMC). Draco also contains general services in the ds++ component directory, visualization classes in the viz component directory, and communication classes in the c4 component directory. For Monte Carlo applications in particular, Draco's rng component directory contains C++ wrappers for the vendor-supplied random number generators, SPRNG [3] and Random123 [4].

Milagro's purposes are to verify its underlying classes, which are to be used in assembling external IMC packages, and to provide a testbed for methods research. Milagro and its underlying classes have been well verified from the lowest line-of-code level, through each component, to the highest compiled-code level. Extensive verification is made possible by Milagro's levelized design.

1.2. History

We began working on the Jayenne project [5] in October 1997. The Jayenne project consisted of three phases. The first phase was a simple neutronics code, **mctest**, which was our first experiment with C++ classes. The next phase of the project was **imctest**, which was a simplified IMC code that gave us some experience with C++ . As we gained experience with **imctest** and the C++ language, **imctest** evolved into Jayenne's final phase, Milagro.

Milagro was first officially released on June 4, 1999 [6]. For its first release, Milagro had a significant regression test suite and had run the following verification test problems: a Marshak wave with an isotropic incoming intensity [7, 8], a Marshak wave with a delta function source, the Su/Olson non-equilibrium transport benchmarks (no scattering and 50% scattering) [9], and an Olson variant of the Marshak wave [10].

Milagro-1_1_0 was released on October 26, 1999, with the added capability of user-defined surface source cells [11]. Milagro-1_2_0 was released on November 12, 1999, with a corrected material energy volume source [12].

Packages built from Milagro's underlying classes were also produced as Milagro was being developed. The Milstone [13,14] package was developed to provide an IMC capability on orthogonal, Cartesian meshes. Milstone results were first published in October, 1998 [15].

Milagro-2_0_0 was released on December 14, 2000 and contained the improved parallel design. Milagro-2_1_0 was released on July 31, 2001 and it contained the RZ-Wedge functionality, new mesh and particle packing capabilities, and a wrap-around at one billion random numbers to accommodate SPRNG's inability to provide more than size_of(int) generators. Milagro-2_2_0 was released on December 19, 2001, and it contained the AMR RZ-Wedge capability, particle tracking with implicit capture down to a weight cutoff and analog capture thereafter, improved graphics via Draco's viz package, and the incorporation of the Common Data Interface (CDI) for accessing opacity and equation-of-state data.

Milagro-3_0_0 was released on May 10, 2002. It introduced multigroup frequency capabilities. Version 3_1_0 was released January 8, 2003, with Doxygen autodoc support. Milagro-3_2_0 was released April 1, 2003. It improved Compaq support and upgraded the autodoc system. The 3_3_0 release on July 3, 2003, introduced a new Dracobuild system and random walk capabilities for gray XYZ problems.

Milagro-4_0_0 was released August 3, 2004. This release introduced ClubIMC to Milagro; introduced R (AKA Sphyramid) geometry; added tally spheres to RZ geometries; and extended Random Walk to R and RZ geometries, and to multigroup problems. Version 4_1_0 was released on January 27, 2005, added tally spheres to R geometries and introduced Camel Norton code refactor to Milagro. Milagro-4_2_0, released July 8, 2005, implemented Compton scattering, an improved get-draco script, improved source-tilt sampling, changes to census particle sampling, and performance improvements. On November 2, 2005, Milagro-4_3_0 was released with an implicit Compton scattering option, further improvements to the get_draco script, and changes in the topology_builder.

Version 5_0_0 is scheduled for release in September 2010. Between the year 2005 and 2010, the team focused more on the Wedgehog package and less on Milagro. The Milagro regression tests were kept up and included new capabilities, but there were no code releases in this time frame. This was also the time frame where another version of Milagro was developed for the IBM Cell chip in the Roadrunner supercomputer mainly by Tim Kelley and Paul Henning. Some of the capabilities added during that time are as follows:

- Opacity Distribution Functions (Seth Johnson, Memo CCS-2:08-52, Mike Buksas)
- Higher-Order Integration for the Fleck Factor (Ryan McClarren)
- Rage-type meshes in XYZ (Mike Buksas)
- Discrete Diffusion Monte Carlo for gray, R and RZ (Jeff Densmore and Kelley Thompson)
- advection, particle, and random number generator refactor (Paul Henning)
- source cutoff functionality (Scott Mosher)
- RZ line segment tallies (Mike Buksas)
- XYZ finite, rectangular, axis-aligned tally surfaces (Gabe Rockefeller)
- Frequency-dependent surface sources (Todd Urbatsch)
- improved algorithm for calculating source numbers (Todd Urbatsch, Scott Mosher, Memo CCS-2:07-59)

Although we've listed the responsible parties, the entire team was engaged in all the activities that go along with producing, maintaining, and supporting code.

CHAPTER 2

Users' Guide

Milagro is mainly used as a verification tool for the classes making up IMC packages delivered to external customers. It is also useful as a research testbed for computational radiative transfer methods, algorithms, and computer science issues. For all of these reasons, we describe how to build and run Milagro and the input it requires. For access to executables, source code (if appropriate), and required libraries, contact one of the Milagro code developers listed in Section 2.8 of this chapter.

2.1. Building Milagro

The Milagro code base emphasizes a modular design to maximize code reuse and minimize code complexity in any single package. To meet this objective, the Milagro solver links against package libraries found in the ClubIMC and Draco software suites. Milagro also makes use of third party vendor libraries for parallel communication, the generation of random numbers and some generic mathematical algorithms (like singular value decomposition).

Because of the the modular design, building Milagro requires access to Milagro, Draco, and ClubIMC source code as well as access to an implementation of MPI (library and header files) and the GNU Scientific Library (GSL) [16]. The Milagro, ClubIMC and Draco source code can be checked out from CCS-2's SVN repository using the commands:

```
% svn co svn+ssh://ccscs9.lanl.gov/ccs/codes/radtran/svn/jayenne/trunk jayenne
% svn co svn+ssh://ccscs9.lanl.gov/ccs/codes/radtran/svn/draco/trunk draco
```

These commands will place the Draco and Jayenne source code in directories that are at the same level. The Jayenne directory contains the ClubIMC and Milagro source code in parallel directories. Milagro requires all of the libraries provided by ClubIMC, but only a small portion of those provided by Draco. However, it is expedient to generate all of the Draco and ClubIMC components even though some will not be used by Milagro. This guidance differs from previous versions of Milagro where only the required portions of Draco and ClubIMC were checked out and built.

The preferred compilation model uses separate source, build and target directories. We have just finished discussion the source directories that contain source code checked out from SVN and the CMakeFiles.txt build system files. New directories will be created for the build system and object files and the final set of libraries, headers and executable software will be installed to the target directory. For example, to make a parallel executable, one might call the build directory "parallel" and make two build subdirectories, "jayenne" and "draco," one for each software suite:

```
% mkdir parallel
% cd parallel
% mkdir draco jayenne
```

Now Draco and Jayenne need to be configured and compiled. All software suites use CMake [17] for generating Makefile-based projects. To generate the Makefile-based project for Draco, change the working directory to the the Draco build directory, and generate the build system with the following command:

```
% cd draco
% cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=${target-dir}/parallel \
   ${draco-src-dir}
```

where the CMAKE_INSTALL_PREFIX specifies the install (or target) location and draco-src-dir would be the path to the Draco sources (i.e., /users/jdoe/working/draco). The CMAKE_BUILD_TYPE is set to RELEASE to build an optimized version of Milagro. Use the value DEBUG to generate a binary that saves debug symbols in the binary. The configuration step makes many assumptions for you including the choice to build a MPI (Message Passing Interface) aware executable if the MPI libraries are found on the local system and it chooses the Design-by-Contract level automatically based on the value of CMAKE_BUILD_TYPE. The default values can be altered by the developer at configure time by providing additional arguments to the cmake command. The Draco Build System manual [18] should be reviewed for detailed descriptions of configuration options.

Jayenne codes use two external libraries: MPI [19] and GSL [16]. These libraries must be found by the build system. If these libraries are not in customary locations, you may need to provide additional information to CMake to help if find the libraries. The Draco Build System manual should be referenced for this type of advanced configuration.

Once configured, Draco can be compiled:

```
% make -j N install
```

Where N is the parallel level desired for the build. Now, repeat the procedure for Jayenne. Go up one directory and back down to the Jayenne build directory

```
% cd ../jayenne
% cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=${target-dir}/parallel \
    ${clubimc-src-dir}
% make -j N install
```

The component and regression tests in a directory may be run using the command 'ctest -j N' from the build directory. For example:

```
% cd .../parallel/milagro/src/milagro_xyz/test
% ctest -j16
```

After compilation, the milagro executable is stored in the build tree at .../parallel/milagro/src/milagro/bin.

2.2. Command Line Input

The Milagro executable is called milagro. Usage is as follows:

```
milagro --input filename --verbose --core --restart restart_file --version
```

For parallel runs using MPI with N processors, the Milagro command line would begin with "mpirun -np N."

Standard usage is to supply the input file if it is a new run or the restart input file if it is a restart run. The "-verbose" option is for debugging; if supplied, detailed particle events and

data are printed to standard out. The presence of the "`-core`" argument specifies that, instead of catching an assertion, `Milagro` dumps a core file when an assertion is fired.

If "`-version`" is specified on the command line it takes precedence, and `Milagro` reports its version and the version of the packages it depends upon. The "`-version`" argument may appear alone.

2.3. Input File

`Milagro` reads its input from a file. Mesh information for an orthogonal, structured, nonuniform Cartesian or RZ mesh may be supplied directly in this input file or from a separate mesh file, the name of which is specified in the input file. Both the input and mesh parsers look for known keywords and subsequent input data in the form "keyword: {*free form input data*}". If a keyword is not present, its associated data will be defaulted, if applicable. If a keyword is present, it must contain data. The keywords are sectioned into blocks, which are delineated by respective end-block statements.

2.3.1. Comment Lines. Line comments may be inserted by placing any of the characters `#`, `!`, `%`, `C`, or `c` at the beginning of the line, followed by a space and then the comment text.

2.3.2. Unit System. `Milagro` uses the units shown in Table 2.1.

TABLE 2.1. Unit system employed by Milagro.

Quantity	Unit	Comment
Length	cm	100 cm = 1 m
Mass	g	1000 g = 1 kg
Energy	jerk	10^9 J = 1 jk
Time	shake	10^9 sh = 1 sec
Temperature	keV	1 K = 8.621738×10⁻8 keV
Opacity (σ)	cm^2/g	
Heat Capacity (C_{ve})	$jk/keV/cm^3$	

2.3.3. Mesh File. The mesh specification file, if it exists separately from the input file, must contain the following blocks:

- The title block of the mesh file must specify the coordinate system.
- An abbreviated source block must contain mesh-specific, surface-source position information for "num_ss", "sur_source", and, optionally, "num_defined_surcells" and "defined_surcells".

Also, the "mesh_type" entry in the title-block is optional, but must be provided if the mesh description is for an AMR mesh. The format for the mesh specification file should have the following form:

```
# mesh specification file

coord: ["xy", "xyz", "rz", or "r" (all lower or all upper case)]
[mesh_type: amr]
[frequency_model: gray|mg|odf ]
end-title
··· initialization block ···
end-init
··· mesh block ···
end-mesh
··· abbreviated source block: surface source position information ···
end-source
```

2.3.4. Mesh Defined in Input File. If the input file does not contain explicit mesh specifications, it must point to a mesh file that does contain the mesh specifications. In this case, the input file has the following block layout with the following particular title block:

```
c
c title block
c

title: mytitle [default "Milagro"]
mesh_file: mymeshfile
[frequency_model: gray|mg|odf ]
end-title

c
c material block
c

··· material block ···
end-mat

c
c source block
c

··· source block ···
end-source
```

If the input file explicitly contains the mesh specification, it has the following block layout with the following particular title block:

```
c
c title block
c

title: mytitle [default "Milagro"]
coord: ["xy", "xyz", or "rz" (all lower or upper case)]
[mesh_type: amr]
[frequency_model: gray|mg|odf ]
end-title

c
c initial zoning info block
c

··· initialization block ···
end-init

c
c mesh info block
c

··· mesh block ···
end-mesh

c
c material block
c

··· material block ···
end-mat

c
c source block
c

··· source block ···
end-source
```

2.3.5. Initialization Block. The initialization block contains coarse–grained mesh information for an orthogonal, structured, possibly nonuniform mesh. The mesh is first specified at a coarse level. Relevant physics parameters are also defined on the coarse mesh. Each coarse portion of the mesh is called a zone. Zones are numbered beginning with "1" at the lowest (x,y,z) zone and increase with the x-dimension "spinning" fastest.

For RZ meshes, the y-related variables are not input and the x-related input quantities are referred to with "r" instead.

For graphics dumps, regions may also be optionally set. They are set by stating the number of regions, the number of coarse zones per region, and the zones per region. Implicit in describing the

number of zones per region is the increasing region number, which begins with "1". Specifying the actual zones per region requires specifying, for each region, the keyword "regions:", followed by the region number, followed by the zones in the region.

Consider a 2×2×2 block of coarse cells that is radiatively reflecting on its low sides and vacuum on the high sides. Suppose we are interested in dividing the graphics output into two regions, one for the lower half in the z-direction and one for the upper half. The initialization block looks as follows:

```
num_xcoarse: 2
num_ycoarse: 2
num_zcoarse: 2
lox_bnd: reflect                          ← must be either "reflect" or "vacuum"
hix_bnd: reflect
loy_bnd: reflect
hiy_bnd: reflect
loz_bnd: vacuum
hiz_bnd: vacuum
num_regions: 2                            ← for graphing purposes
num_zones_per_region: 4 4          ← for graphics purposes, num_regions entries
regions: 1    1 2 3 4                     ← for graphics purposes
regions: 2    5 6 7 8                     ← for graphics purposes
end-init
```

Note that the spacing and carriage returns in the "regions:" specification are not required, but they add clarity.

2.3.6. Mesh Block. `Milagro` supports two fundamental mesh types. *Tradiational Meshes* are single level and with uniform cell size and spacing. *AMR Meshes* are an unstructured mesh format that allows a single level of refinement between adjacent cells.

Traditional Meshes

The mesh block specifies the mesh. For each dimension, the user specifies the locations of the coarse zoning, the number of fine mesh divisions per coarse zone division, and, optionally, the ratio of the fine mesh sizes. In a given dimension, ratio zoning means that the next fine mesh division in the positive direction has a size that is "ratio" times as large. A ratio of unity, which is the default, implies uniform fine meshing within a coarse zone. A ratio greater (less) than one implies increasing (decreasing) sizes in the positive direction. If any ratio is specified for any given dimension, the ratios for all the coarse zone divisions in that dimension must be specified. An example mesh block follows:

```
wedge_angle_degrees: 5.0              ← required only for RZ geometries
xcoarse: 0.0 1.0 2.0                  ← requires num_xcoarse+1 entries
num_xfine: 1 2                          ← requires num_xcoarse entries
ycoarse: -1.0 0.0 1.0                 ← requires num_ycoarse+1 entries
num_yfine: 1 1                          ← requires num_ycoarse entries
zcoarse: 0.0 0.1 0.2                  ← requires num_zcoarse+1 entries
num_zfine: 10 1                         ← requires num_zcoarse entries
zfine_ratio: 1.5 1.0          ← requires num_zcoarse entries, if any specified
end-mesh
```

Note that, in this example, the ratio zoning defaults to uniform zoning within each coarse zone in the x- and y-dimensions.

Mesh Block for AMR XYZ meshes

The "AMR XYZ" versions of `Milagro` (`milagro_amr_xyz` and `milagro_amr_xyz_mg`) use a mesh designed for compatability with the AMR style mesh used in hydrodynamics codes. This mesh requires that all cells have a unit aspect ratio, all cell refinement is done by bisection, and that neighboring cells differ by at most a single level of refinement. These requirements mean we use different keywords in the mesh description.

We still retain the concept of coarse and fine meshes, but use them differently in the AMR XYZ mesh. The coarse mesh is defined with the keywords: `num_Xcoarse` where X=`x,y,x`, `coarse_cell_size`, which specifies all three (equal) dimensions of the coarse cells, and `lowX_val`, (again, X=`x,y,z`) which specifies the low-coordinates in the three dimensions.

We restrict the mesh to a single level of refinement within each coarse cell. (This is considerably less general than the internal mesh type is capable of, but is a reasonably flexible mesh which is easy to describe in a `Milagro` input deck.) The refinement level of a coarse cell is given with the `refined` keyword:

`refined:` *{cell number} {refinement level}*

It is not necessary to specify the refinement level of *all* of the coarse level zones required to satisfy the requirement that neighboring cells differ by at most one level. In the process of building the mesh, `Milagro` will automatically elevate the refinement level of all coarse cells to the minimum value that meets this requirement. For example if a coarse zone is given refinement level 3 (meaning that it is bisected three times in each dimension, for a total of $8^3 = 512$ cells), its neighboring zones in all three directions are assigned refinement level 2, and its neighbors' neighbors' are assigned level 1. The level of any of these zones, including ones that are specified explicitly, can be raised if necessary to meet the 2:1 requirement of any other zones. For example, see figure 2.1. In the two dimensions that we can see, the coarse mesh has been refined to 3×3. The center coarse zone was given a refinement level of 2, causing its neighbors to be refined to level 1.

Table 2.2 shows a complete AMR XYZ mesh description. It results in a 3D checkerboard of zones, where the alternating zones have refinement levels 0 or 1. The result is pictured in Figure 2.2.

2.3.7. Material Block. The material block is where the user defines the material properties and associates a material with each coarse–mesh zone. Currently, `Milagro` only reads in user–specified opacities and specific heats. The user must always specify in the material block the number of zones in the problem so that material arrays may be properly sized. The number of zones is required because the mesh may be specified in a separate mesh-file. The "zonemap" specifies the material in zones 1 through num_zones. The material IDs begin with unity. For each material, the user must give a description keyword and specify the density [g/cm^3], the initial temperature [keV], the absorption/emission opacity index, the isotropic scattering opacity index, and the equation of state index.

The user may specify a number of different opacities and opacity models. The numbers thereof are dictated by the keywords "num_opacities" and "num_opacity_models", respectively. For each opacity model, the user enters the keyword "opacity_model:" followed by the index (starting at unity) of the model and then the model type and coefficients. Likewise, opacities are entered with the "opacity:" keyword, followed by the index of the opacity and the opacity data: either the name of a file of opacity data or the keyword "model" followed by the type and the index of the desired

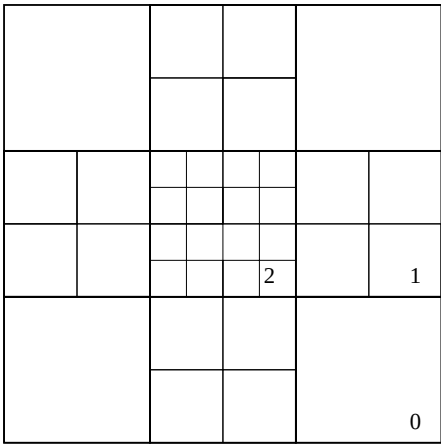

FIGURE 2.1. Example of neighboring zone refinement. Center zone was refined to level 2.

```
num_xcoarse:  2
num_ycoarse:  2
num_zcoarse:  2
end-init

lowx_val:  0
lowy_val:  0
lowz_val:  0

refined:  2 1
refined:  3 1
refined:  5 1
refined:  8 1
```

TABLE 2.2. Example input deck for an AMR XYZ mesh

opacity model. The opacity model can be set to "constant" or "polynomial". The "constant" model sets the opacity to the fixed value provided in the model description (see example below where opacity model 2 is fixed so that $\sigma = 0.0$). The polynomial model is specified by providing the four constants for the following expression:

$$\sigma(T, \rho) = (a + b \cdot T^c \cdot \nu^e) \cdot \rho^d,$$

where a is a constant with units $cm^2/g \cdot (cm^3/g)^d$, b is the temperature multiplier with units $keV^{-c} \cdot Hz^{-e} \cdot cm^2/g \cdot (cm^3/g)^d$, c is the temperature power, d is the density power and e is the frequency power.

Each problem may use multiple equations of state, given by the keyword "num_eos:". For `Milagro` the EOS model is used to spacify the heat capacity model. Each model is specified by using the "eos:" keyword followed by the index of the eos (again, indexed from unity) and the eos

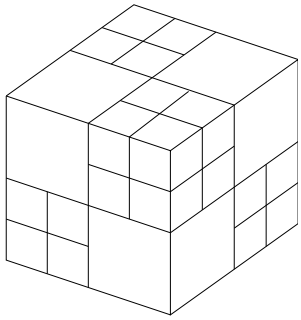

FIGURE 2.2. Example AMR AYZ mesh. See Table 2.2 for input.

model specification. The EOS model may be "analytic" or "tabular". For analytic models, the input keyword "analytic" must be followed by the model type ("constant" or "polynomial"). To provide tabular data, the user may input a filename for a two-dimensional array of heat capacity values.

The material block varies somewhat depending on whether the problem is gray, multigroup, or ODF. In gray problems, the scattering model must be "isotropic_coherent", while in multigroup and ODF problems "explicit_compton" is also valid. Gray and multigroup problems may include a "hybrid_diffusion" specification, which is an invalid input for ODF. Multigroup and ODF problems must specify the number of groups; in multigroup, the user must specify the group boundaries; and in ODF problems, the user gives the number of bands (which, input once, is the same for each group).

The material block also contains the Fleck implicitness factor, α, where $0 \leq \alpha \leq 1$. For $\alpha = 0$, the Fleck and Cummings IMC reverts to time-explicit radiation/material coupling. For $\alpha = 1$, the Fleck and Cummings IMC method attains its maximum degree of time-implicitness. Additionally, the user can designate the keyword "implicitness: Adaptive". This allows the code to choose a zone- and time-step-dependent $\alpha(\Delta_t)$ such that $0.5 \leq \alpha(\Delta_t) \leq 1.0$, where $\alpha \approx 0.5$ for optically short time steps and $\alpha \approx 1$ for optically long time steps.

The first example material block has one material, with a density of 3.0 g/cm^3, an initial temperature of 1.0e-6 keV, a polynomial absorption opacity of 100.0/T^3 cm^2/g, a scattering opacity of zero, and a constant eos of 0.1 Jerks/g/keV.

```
num_zones: 1                                    ← always required
zonemap: 1
num_materials: 1
num_opacity_models: 2
num_opacities: 2
num_eos: 1
opacity_model: 1 polynomial 0.0 100.0 -3.0 0.0
opacity_model: 2 constant 0.0
opacity: 1 model absorption 1
opacity: 2 model scattering 2
scattering_model: isotropic_coherent
eos: 1 analytic constant 0.1
mat: 1 marshak_mat 3.0 1.0e-6 1 2 1
                                                ← material id, material descriptor, density (g/cm^3), initial temperature (keV), absorption opacity id, scattering opacity id, and eos id
implicitness: 1.0                               0 is fully explicit in time; 1 is fully implicit in time.
end-mat
```

The following example is an ODF multigroup problem. There is one material with a density of 1.0 g/cm^3 and an initial temperature of 0.4 keV. It reads its absorption opacity from an external file, odfmg32.ipcress, and has a scattering opacity of zero and a constant eos of 0.1 Jerks/g/keV.

```
num_zones: 1
zonemap: 1
num_materials: 1
num_opacity_models: 1
num_opacities: 2
num_eos: 1
num_groups: 32
num_bands: 8
opacity_model: 1 constant 0.0
opacity: 1 odfmg32.ipcress absorption rosseland 19000
opacity: 2 model scattering 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
eos: 1 analytic constant 1.0
mat: 1 mat_1 1.0 0.4 1 2 1
implicitness: 1
end-mat
```

Special Material Setup for Random Walk Problems

If Random Walk is enabled, the user may need to assign Rosseland opacities to regions where Random Walk is active. This can be done by using the keyword `hybrid_diffusion` from within the material block. This command takes two integer arguments. The first is the material index and the second is the opacity model index. For example:

```
...
opacity: 3 model rosseland absorption 1      ← create a special Rosseland opacity model to
                                               be used by RW regions.
...
c RW requires a Rosseland opacity model
c for random_walk: material id, opacity id
hybrid_diffusion: 1 3                        ← assigns opacity 3 to material 1 for RW.
end-mat
```

2.3.8. Source Block. The source block contains physical source input, run time parameters, and edit specifications.

Available source options are an external material volume source, an external radiation source, and a black body surface source. The external material and radiation sources are each entered by coarse-mesh zone in units of jerk/cm^3/shake. The user must specify the temperature, in keV, of the black body surface source. Due to `Milagro`'s limited input capability, its external sources are limited. The material volume source is required to be constant in time, and the radiation source is constant from time zero to a user-input stop-time. These limitations are of little concern because `Milagro`'s underlying classes do not suffer these limitations, and `Milagro`'s main mission is as a verification and research testbed. Defaults for all external sources are zero.

The external material volume source in each zone is specified after the keyword "vol_source:". The radiation source in each zone is specified after the keyword "rad_source:" and the stop-time, in shakes, is entered after the keyword "rad_s_tend:". If a value is entered for any zone, entries must be made for all zones.

The number of surface sources in the problem is specified with the keyword "num_ss:". Any number of surface sources may be entered, but the ultimate limitation is that any given cell in the problem may have a surface source on no more than one of its faces. The temperature, in keV, of each surface source is specified with the keyword "sur_temp:". Each surface source must have the same angular distribution, either "cosine" or "normal", specified with the keyword "ss_dist:". The location of each surface source is specified in one entry with the "sur_source:" keyword, which may take any of the following values, "lox", "hix", "loy", "hiy", "loz", or "hiz". In RZ geometry, "hir", "loz", and "hiz" are acceptable; "lor" is not. With no further specification, the surface source will be applied to the entire requested face of the problem. `Milagro` will check that the requested surface source is on an edge of the system with a vacuum boundary, unless the keyword "ss_descriptor:" is set to something other than its default of "standard". Allowable for testing purposes, but not recommended for physics reasons, is "ss_descriptor: allow_refl_bc" which allows a surface source to exist on a specularly reflecting boundary. If a nonstandard ss_descriptor is specified for a zone, then the ss_descriptor must be specified for all preceding zones regardless of whether they are standard or not. For multi-group only, the user can specify a frequency-dependent source (which is in its own input block) and refer particular surface sources to those spectra with the keyword, "ss_spectrum," which, if =0 defaults to the original Planckian distribution. Otherwise, the user-defined spectra indices are 1-based.

The user may make additional specifications to refine a surface source to an area less than an entire system boundary [11]. In this case, the user must define the individual cells where a surface source is applied. First the user specifies how many user-defined cells are in each surface source with the keyword "num_defined_surcells:". Default for each surface source is zero; any leading zeros must be input. For example, if there are three surface sources and the user only wants to specify the cells for the second surface source, the user would write "num_defined_surcells: 0 16"

or "num_defined_surcells: 0 16 0". Each surface source with user-defined cells is specified with its own instance of the keyword "defined_surcells:" followed by the surface source number (whose numbering begins with 1) and then the list of globally indexed cells.

The initial radiation temperature, in keV, is also specified for each zone in the source block after the keyword "rad_temp". The default initial radiation temperature is zero.

The maximum integration order is set in the source block. By setting "integration_order:" the user can modify how the Fleck factor is defined so as to avoid artificial overheating in the solution. The default value for "integration_order:" is 1. When the value (an integer) is set higher than 1, the Fleck factor is allowed to decrease (making the solution more implicit) in regions where overheating could occur.

The user may specify a temperature below which particles will not be sampled with the keyword "cutoff_temp:". Setting a source cutoff temperature allows the user to only sample the energetic particles in the source region, excluding large areas of cold cells away from the radiation source.

By default, Milagro does not use hybrid diffusion, but the user can activate it using the "hybrid_diffusion:" keyword. It can take one of two valid integer inputs: 1 represents random walk, allowing Milagro to approximate many scattering events with one condensed history random walk; and 2 represents DDMC, which calculates transport to a zone boundary in thick diffusive regions using diffusion theory.

Also, by default `Milagro` does not tally Eddington tensors, but these can be turned on by setting the "eddington_tallies:" keyword to any case variation of "On", "True", or "1" (setting it to "Off", "False", or "0" is also permissible). Eddington tensor data is edited along with the usual "edit cell" output.

The run time parameters are specified in the source block. They include the number of particles per cycle (nominal "npnom:", maximum "npmax:", and rate of change "dnpdt:" in particles per shake); the number of cells per processor, "capacity:"; the size, in particles, of the buffer for communication and census dumping, "buffer_size:"; and the random number seed, a positive integer, "seed:".

The simulation time is controlled through the maximum number of cycles to run, "max_cycle:", and the maximum time to run to, "max_time". One of these parameters must be specified in the input file. If only one of these parameters is specified, the other is effectively ignored (by setting it to the maximum possible value). If both "max_cycle" and "max_time" are specified, then the first condition met will stop the simulation.

The time step is determined through the input parameters "timestep" and "dt_ramp_cycle". The "timestep" parameter (in shakes) is the constant time step the code will take, after ramping the time step over "dt_ramp_cycle" cycles. The first time step is size "timestep / dt_ramp_cycle", and then the time step is ramped linearly until cycle "dt_ramp_cycle", after which the time step is held constant at "timestep". If not specified, "dt_ramp_cycle" is 1, so that "timestep" is the time step value for all cycles. If "max_time" is specified, then the final time step might be adjusted so that the final time lands on "max_time".

Edit parameters control the amount and frequency of various types of output. Output is printed to standard out every "print_frequency:" cycle. The number of cells that are printed out defaults to the total number of cells in the problem, but it can be limited by the "num_edit_cells:" keyword and corresponding list of (globally numbered) edit cells after the keyword "edit_cells:". Restart dumps are made every "restart_frequency:" cycle. Graphics dumps are made every "graphics_dump:" shakes.

Let us consider a problem out to 0.1 shakes, where we steadily increase the number of particles. We will apply a surface source to the low and high z-faces of the cold material. We will also apply a small (constant) material volume source and a radiation source of duration 0.01 shakes. The initial radiation temperature matches the initial material temperatures. We will ignore cells below 0.005 keV and allow random walk hybrid diffusion. An example source block follows.

```
timestep: 0.001       ← in shakes, constant
dt_ramp_cycle: 10     ← ramp time step to final value over this many cycles
max_time: 0.1         ← final time, in shakes
npnom: 100            ← nominal number of particles per cycle (or time step)
npmax: 1000           ← maximum number of particles per cycle (or time step)
dnpdt: 900            ← rate of change of particle in particles/shake
rad_temp: 0.2 0.2 0.3 0.3 0.2 0.2 0.3 0.3      ← initial radiation temperature (keV) for each zone
vol_source: 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01      ← material volume source in jerks/sh/cm^3
rad_source: 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1      ← radiation source, in jerks/sh/cm^3
rad_s_tend: 0.01      ← duration of radiation source from time zero, in shakes
cutoff_temp: 0.01     ← minimum sampling temperature (keV)
num_ss: 2             ← number of surface sources
sur_source: loz hiz   ← position of the num_ss surface sources
sur_temp: 0.4 0.4     ← temperature, in keV, of the num_ss blackbody surface sources
ss_dist: cosine       ← angular distribution of all surface sources.
ss_spectrum: 0 1      ← MULTIGROUP ONLY: index into user-defined spectra; 0=default Planckian.
hybrid_diffusion: 1   ← enables random walk hybrid diffusion
num_defined_surcells: 0 6      ← six user-defined cells for the 2nd surface source.
defined_surcells: 2    61 62 63 64 65 66      ← six user-defined cells for the 2nd surface source.
integration_order: 4  ← Allow adaptive definition of the Fleck factor, up to order 4
num_ss: 2             ← number of surface sources
print_frequency: 1    ← print every cycle
num_edit_cells: 12    ← print out only 12 cells
edit_cells: 1 2 3 4 5 6 61 62 63 64 65 66      ← print out only these cells
restart_frequency: 50 ← make restart dumps after every 50 cycles
graphics_dump: 0.1    ← make graphics dumps after every 0.1 shakes
buffer_size: 100      ← buffer size, in particles
seed: 12345           ← random number generator seed
end-source
```

2.3.9. User-Defined, Frequency-Dependent Spectra. The user can define frequency-dependent spectra in the freq-dep-source block. The surface source (currently, only the surface source) can utilize these spectra via the "ss_spectrum" keyword in the source block. This block can appear anywhere after the material block, where the number of frequency groups is defined. This example block shows two user-defined spectra, the first one with all the source in the last group, and the second one with all the source in the first group. The elements of the spectra must be non-negative. The strength of each surface source is still defined by the Planckian-equivalent temperature.

```
num_defined_spectra: 2                          ← number of user-defined spectra
defined_spectra: 1    0.0 0.0 1.0   ← 1-based index; num_group non-negative entries
defined_spectra: 2    1.0 0.0 0.0   ← 1-based index; num_group non-negative entries
end-freq-dep-source
```

2.3.10. Surface Tally Block.

The Capability

The `Milagro` surface tally capability allows the user to define a set of surfaces through which the radiation intensity may be estimated. Currently, three types of surfaces are supported: Spheres (and half-spheres) in R and RZ geometries, line segments in RZ, and axis-aligned rectangles in XYZ. Two kinds of tallies are supported. The "point" tally records all radiation crossing the surface according to the angle it makes with an absolute direction: $\hat{z}$ in RZ and $\hat{r}$ in R. This tally is useful for estimating the radiation intensity at distant points (e.g. the "far-field" solution) and is normally used with spherical surfaces. The "normal" tally records the direction of radiation according to the angle it makes with the normal of the surface. This tally is normally used with the segment surfaces in RZ. Tallies are available in both gray and multigroup frequency treatments. Multigroup surface tallies use the same group structure as the opacity data for frequency bins.

For both types of tallies, the angular information is recorded into discrete bins. The user defines these bins by specifying the cosines (μ) of their boundaries, including the boundary values of -1 and 1. The angular mid-point where $\mu = 0$ must be one of the angular bin boundaries as well. A single angular mesh is used for all of the angular tallies.

The tally surfaces must lie entirely with the domain of the problem, as described by the mesh, and obey whatever geometric constraints are implied by the symmetries of the geometry. E.g. spheres in RZ must lie on the $\hat{z}$ axis and spheres in R must be centered at the origin. We also allow a sphere in RZ to be centered at either end-point of the z-axis, forming a half sphere.

Figure 2.3 shows a spherical surface in an RZ geometry. A single bin of the angular mesh is also illustrated, oriented according to the point tally method.

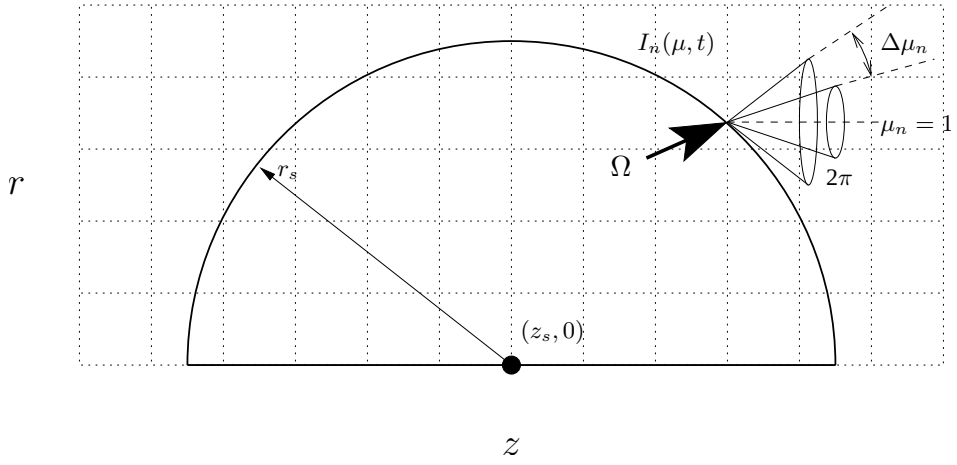

FIGURE 2.3. A spherical tally surface on an RZWedge mesh.

Input Specification

A description of the Surface Tally block of the input deck can be found in Figure 2.3.

`num_surfaces:` $S > 0$
`surface:` *`surface_kind tally_kind {geometry data}`* ← Repeat for S surfaces.
`num_bins:` $B \geq 2$
`bin_cosines:` `-1 {`$M-1$ `values, including 0} 1`
`end-surfaces`

TABLE 2.3. Form of Surface Tally Input Block

Notice that -1 and 1 must be included in the list of angular bin cosines and that 0 must be among the remaining $M-1$ values. Allowable values for *surface_kind* and *tally_kind* and their meanings are given in Table 2.4.

Surface Type	
0	Sphere
1	Segment
2	Oriented Segment
3	Axis-Aligned Rectangle

Tally Type	
0	Absolute direction (point tally)
1	Relative to normal

TABLE 2.4. Meanings of *surface_type* and *tally_type* parameters.

Surfaces are assigned numbers according to their order of appearance in the input deck, starting with Surface 1. The geometry data required for each surface description consists of one, two, four, or six values as given in Table 2.5.

	Geometry		
Surface Type	R	RZ	XYZ
Sphere	r_s	z_s, r_s (radius)	–
Segment	–	z_1, r_1, z_2, r_2	–
Rectangle	–	–	x_1, y_1, z_1, x_2, y_2, z_2

TABLE 2.5. Contents of *`{geometry_data}`* from Table 2.3

For RZ line-segment tallies, the outward direction is defined as left-to-right from the perspective of someone standing at point one (z_1, r_1) and looking toward point two (z_2, r_2). Notice that the ordering of the coordinate system is significant in this definition. See Figure 2.4 for further clarification. In this diagram we have illustrated one of the angular bins corresponding to an outward direction. Furthermore, this angular bin is measured from the normal vector of the segment, indicating a normal tally operation.

The angular mesh is conveyed to `Milagro` by the cosines of the angles of the edges of the angular bins. The number of edges is given in parameter `N_cosine_edges` is therefore one greater than the number of bins. The cosine values themselves are passed in the parameter `cosine_edges`, which is a double-precision array. The values should be in increasing order, beginning with -1, including 0 as a value, and finishing with 1. Because zero must be one of the angular bin boundaries, the angular mesh must contain at least two bins, and `N_cosine_edges` must be at least three.

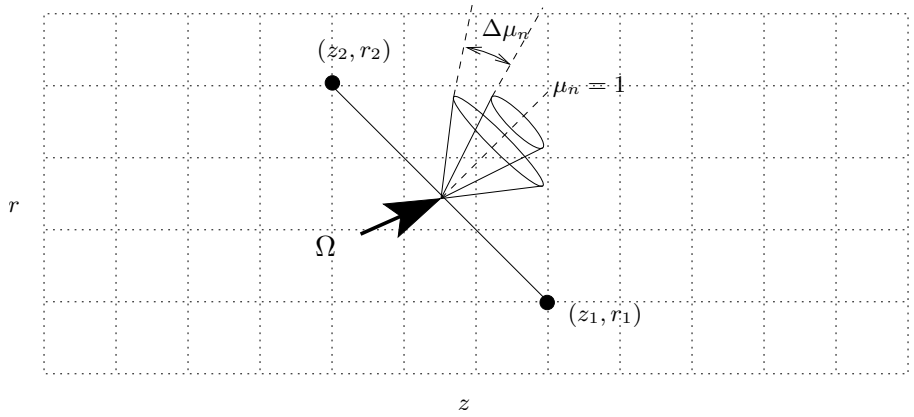

FIGURE 2.4. A segment tally surface on an RZWedge mesh.

Output

In gray problems, the tally sphere response is given by

$$\begin{aligned} R_n &= \frac{S_n}{2\pi\Delta\mu_n A\Delta t} \\ &= \frac{\text{jks}}{\text{str}\cdot\text{cm}^2\cdot\text{sh}}, \end{aligned} \tag{1}$$

where S_n is the tallied energy crossing the surface, in or out, in each cosine bin during the time step, A is the area of the tally surface subtended by the mesh, Δt is the time-step size, n is the cosine-bin index, and $\Delta\mu_n$ is the size of the cosine bin (see Fig. 2.3). For multigroup problems, the response is

$$\begin{aligned} R_{n,g} &= \frac{S_{n,g}}{2\pi\Delta\mu_n A\Delta t\Delta\nu_g} \\ &= \frac{\text{jks}}{\text{str}\cdot\text{cm}^2\cdot\text{sh}\cdot\text{keV}}, \end{aligned} \tag{2}$$

where $S_{n,g}$ is the tallied energy crossing the surface, in or out, in each cosine-frequency bin during the time step, g is the frequency-group index, and $\Delta\nu_g$ is the group width. We note that each angular bin (or angle-frequency bin in multigroup problems) yields two responses: one for particles leaving the sphere and one for particles entering the sphere.

Usually, the return quantities from `Milagro` are given per volume to remove any imprint of the RZWedge approximation [20]. However, in this case, the surface tally intensities are independent of RZWedge volume and angle (except for inaccuracies due to large wedge angles) and do not require further normalization.

Milagro records the surface tally information in files with names of the form:

{problem_name}`_surface_tallies.`*{cycle number}*

The format of the output is as follows:

```
Number of angular bins:  B
Cosines of bin edges:  -1.00000000 {B-1 values} 1.00000000
Number of surfaces:  S
Tallies for group:  g                    ← Block repeats for G groups.
  Tallies for surface:  s                    ← Repeats S times inside each group block
    Inward energies:  B real values
    Inward counts:    B int values
    Outward energies: B real values
    Outward counts:   B int values
```

FIGURE 2.5. Output format for surface tally files

2.4. Restart Input File

`Milagro` may be restarted from its restart dumps. Restarting `Milagro` is much like normally running `Milagro` except that a restart file must be constructed. In lieu of the input file, the restart file is specified on the command line with a "-r".

The required entries in a restart file are "mesh_file:", "title:", "restart_cycle:" and "frequency_model:". Again, the title and file names must be less than 20 characters long. The edit-cell inputs, "num_edit_cells:" and "edit_cells:", are not saved in the restart dumps. Therefore, during a restart, the user must enter the edit cell information in the restart file in order to maintain or change the edit cell information.

Given the example input blocks above, we could restart the calculation after the 50th cycle with the following input:

```
mesh_file: myinputfile          ← since we specified the mesh in the input file
title: mytitle
restart_cycle: 50
frequency_model: [gray|mg|odf]
num_edit_cells: 12              ← required to maintain the same edit cells
edit_cells: 1 2 3 4 5 6 61 62 63 64 65 66   ← required to maintain the same edit cells
end-restart
```

This restart would exactly replicate the original calculation from cycle 51 to 100.

The restart capability may also be used to modify some parameters of the calculation. Edit and restart frequencies may be modified. The number of particles may be modified. The parallel topology may also be modified on a restart. The list of keywords that may be modified follows:

- npnom: Number of particles.
- npmax: Maximum number of particles.
- capacity: Number of cells per processor.
- dnpdt: Differential number of particles per time step.
- max_cycle: Maximum problem cycle.
- max_time: Maximum problem time.
- num_edit_cells: Number of edit cells (not saved in restart).
- edit_cells: Cells to print out during problem edits (not saved in restart).
- print_frequency: Cycle frequency to print out edits.
- restart_frequency: Cycle frequency to print dump restarts.

- timestep: Time step.
- dt_ramp_cycle: Ramp time step to this cycle number.
- buffer_size: Size of communications buffer.
- graphics_dump: Frequency of graphics dump.

Be wary of drastically changing particle numbers on a restart: you may instigate a step function in particle energy-weights that could undesirably propagate a statistically outlying quantity or effectively lose information.

2.5. Output

While a run is in progress, `Milagro` prints out status updates after of each cycle. Information on the number of particles transported on each processor and the total wall-clock time per cycle are output to standard error. Upon completing a print cycle (dictated by the "print_frequency:" keyword in the source block), `Milagro` will print out an update on the current state of the problem to standard out. If one (or both) of these streams is not redirected to a file, it will print out to the command line interface/terminal instead.

Each print cycle update is given under the header "`RESULTS FOR CYCLE` <cycle number>". The total problem time is given at the beginning of the results block, followed immediately by a "`Material State`" table. The table lists each edit cell (all cells if "edit_cells" has not been set) and a series of entries describing its current state: "T-mat" gives the temperature of the material, in keV; "E-mat" is the energy of the material, in Jerks; "T-rad (path)" and "E-rad (path)" are the average energy over the last time step, evaluated at the middle of the time step, expressed in keV and Jerks, respectively; "E-rad (cen)" is the cell-centered census particle energy, in Jerks; "Evol-net" is the cell-centered net volume energy, in Jerks; "dE/dT" represents the heat capacity of the material in the given cell, in Jerks/keV; and the "Mom-dep" columns give momentum deposition values in each dimension, in $g/cm^2/sh^2$.

Following the "`Material State`" table is a section labeled "`Energy Conservation Check`". It gives the results of several self-consistency energy checks, tracking any energy gained or lost from sampling errors and numerical roundoff. "Cycle energy check" and "accumulated energy check" are an evaluation of the total computational error for the latest cycle and the problem so far, respectively. They are the difference between the end-of-cycle internal energy and the beginning-of-cycle energy plus the in-problem energy flux. If the code encounters no sampling errors, this value will be zero; otherwise, it becomes the excess energy in Jerks. These error values are also given as fractions of the end-of-cycle energy.

The cycle and accumulated "energy loss" fields represent any energy lost (or gained) erroneously. "Energy to census" and "number to census" list the energy and corresponding particles sent to the census database at the end of the last cycle. The initial and ending "internal energy" fields represent all energy in the cell at the beginning and end of the cycle, and the initial and ending "material energy" fields specify the material energy alone. With the exception of "number to census", all of these values are in Jerks.

The "energy check by source type per cycle" section tracks total energy and energy loss by particle type. The "census" column represents census particles from the previous cycle; "volume" is the volumetric particles; and "surface" represents the surface source particles.

The "`HYBRID IMC`" section gives information on what (if any) hybrid diffusion technique is being used. "Transport" corresponds to a regular IMC run with no hybrid diffusion, "Random Walk" and "DDMC" are fairly self-explanatory.

The final block, "`Random Number Streams`" reports what entries from the random number stream have been used over the course of the last print cycle.

Upon completion of the final cycle, `Milagro` will state the total run time for the simulation and then automatically exit.

2.6. Keywords

In the following tables, input file keywords known by `Milagro` are listed by block. Milagro expects the blocks provided in a specific order, but keywords within each block are unordered. Table **??** shows the expected ordering of these input blocks. In Milagro, a colon is used to separate keywords and values.

Table 2.6. Order of Blocks for Milagro Input Files.

Block	Table
Title	See Table 2.7
Initial	See Table 2.8
Mesh	See Table 2.9
Material	See Table 2.10
Tally Surfaces	See Table 2.11
Source	See Table 2.12

Table 2.7. Title Block Keywords.

Keyword	Value(s)	Comment
Lines preceding the keyword title are treated as a comment block.		
title	string	A descriptive string without spaces.
coord	r|rz|xyz	
mesh_type	amr	Only need to be specified if AMR meshes are required.
mesh_file	string	File name where the mesh is defined. When selected, `coord` does not need to be specified.
end-title	-	Signals the end of the title block.

Table 2.8. Initial Block Keywords.

Keyword	Value(s)	Comment
num_[rxyz]coarse	integer	Number of coarse zones in each dimension. Coarse zoning is used for specifying material properties. You must specify an entry for each dimension used by the simulation.
lo[rxyz]_bnd	reflect|vacuum	Boundary condition for low [rz] faces. You must specify a value for each boundary in the simulation.
high[rxyz]_bnd	reflect|vacuum	Boundary condition for high [rz] faces
end-init	-	Signals the end of the init block.

TABLE 2.9. Mesh Block Keywords.

Keyword	Value(s)	Comment
wedge_angle_degrees	0.0-90.0	RZ only: the geometry is actually a 3D wedge shape. This is the number of degrees subtended by the wedge.
cone_angle_degrees	0.0-90.0	R only:
[rxyz]coarse	real real	You must specify two real values that correspond the minimum and maximum coordinate values (cm) for each dimension.
num_[rxyz]fine	integers	The number of fine zones per coarse zone for each dimension. The number of entries must correspond to the number of coarse zones specified by the `[rz]coarse` entry.
coarse_cell_size	real	AMR only
low[xyz]_val	integer	AMR only
refined	int int	AMR only
end-mesh	-	Signals the end of the mesh block.

2.7. Troubleshooting

Input files:

Is there a space after the colon following a keyword? If there is not a space, the data will not be read.

If the mesh-file is separate from the input-file, does it contain all the necessary information?

2.8. Contacts

Please contact the developers for assistance with `Milagro`. Future enhancements in `Milagro` are heavily influenced by customer needs. Requests for capabilities may be made to the following contacts:

- Rob Lowrie, Monte Carlo Team Leader, CCS-2, MS D413, <*lowrie@lanl.gov*>
- Ed Dendy, CCS-2 Group Leader, CCS-2, MS D413, <*dendy@lanl.gov*>

TABLE 2.10. Material Block Keywords.

Keyword	Value(s)	Comment
num_zones	integer > 0	Typically a unique material, density and/or initial temperature is assigned to each zone.
num_materials	integer > 0	The number of materials that will be defined in this block.
zonemap	integers > 0	num_zones entries must be provided. Each value assigns a material index to a zone.
num_opacities	integer > 0	The number of opacity definitions in this block.
num_eos	integer > 0	The number of equation-of-state models defined in this block.
num_opacity_models	integer > 0	The number of opacity models defined in this block.
num_groups	integer > 0	The number of frequency (energy) groups requested for this problem.
group_bounds	real > 0.0	The frequency edges (Hz) for the frequency groups. You must provide num_group+1 entries.
opacity_model	special	Defines an opacity model. The specification is {index} {constant|polynomial} {parameter list}. For the constant model, the parameter list is a single real value. For the polynomial model, five real values must be provided.
opacity	special	Create an opacity set that defines either an absorption or a scattering opacity for all groups. The format must be {index} model {absorption|scattering} {list of opacity model indices}. There must be num_group+1 entries in the list of opacity model indices. The units of opacity are cm^2/g.
eos	special	Create an EOS set that defines the heat capacity. The format must be {index} {analytic} {constant} {parameter list}. For a constant heat capacity, the parameter list is a single real value with units $jk/keV/cm^3/g$.
mat	special	Define a material consisting of absorption and scattering opacity models, an eos model, a density value and an initial temperature. The format must be {index} {string descriptor} {density (g/cm^3)} {initial temperature (keV)} {absorption opacity index} {scattering opacity index} {eos index}.
implicitness	0.0-1.0	
end-mat	-	Signals the end of the material block.

TABLE 2.11. Tally Surfaces Block Keywords.

Keyword	Value(s)	Comment
num_surfaces	int > 0	The number of tally surfaces defined in this block.
surface	-	
num_bins	int > 0	The number of angular bins for the tally.
bin_cosines	-1.0 < reals < 1.0	The angular bin edges (cosines). `num_bins+1` values must be provided.
end-surfaces	-	Signals the end of the surfaces block.

TABLE 2.12. Source Block Keywords.

Keyword	Value(s)	Comment
timestep	real > 0.0	Simulation timestep in sh.
num_ss	integer	The number of surface source definitions in this block.
sur_source	lo[rxyz] | hi[rxyz]	The location of the surface source must be the entire surface on one face of the mesh. `num_ss` values must be provided.
sur_temp	real > 0.0	The temperature of the surface source (keV). `num_ss` values must be provided.
ss_dist	normal|cosine	The angular distribution for the surface source.
num_defined_surcells	integers	`num_ss` values must be provided.
defined_surcells	integers	
rad_temp	reals > 0.0	The initial radiation energy distribution will be Planckian based on this temperature (keV). `num_zones` values must be provided.
vol_source	reals > 0.0	Volume based radiation sources (jks/cm^3/sh). `num_zones` values must be provided.
npnom	int > 0	The initial number of particles in the simulation.
npmax	int > npnom	The maximum number of particles allowed in the simulation.
dnpdt	int	The rate of change per timestep for the number of particles used in the simulation (num/sh).
capacity	int > 0	The number of cells per processor. This value should be larger than `num_cells` for serial and full replication. It must be contrived for stand-alone domain decomposition.
max_cycle	int > 0	The simulation will be stopped after running this number of cycles.
hybrid_diffusion	0-2	0:Use normal IMC transport. 1:Use the Random Walk approximation. 2:Use the DDMC approximation.
print_frequency	int	After running this many cycles, provide a status update to standard out.
buffer_size	int > 0	The number of particles that must be collected before in-cycle communication between processors is triggered.
cuttoff_temp	real > 0.0	-
seed	int > 0	The random number seed.
cnum_ss	int	
csur_source	low[rxyz]	
csur_temp	reals	(keV)
css_dist	cosine	
end-source	-	Signals the end of the source block.

Bibliography

[1] J. A. Fleck, Jr. and J. D. Cummings, "An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport," *Journal of Computational Physics*, vol. 8, pp. 313–342, 1971.

[2] T. Evans, "The Draco system for XTM transport code development," Research Note XTM-RN(U)-98–046, Los Alamos National Lab., 1998. LA-UR-98–5562.

[3] D. Ceperley, M. Mascagni, and A. Srinivasan, "SPRNG: Scalable Parallel Random Number Generators." NCSA, University of Illinois, Urbana-Champaign, Nov. 1997. www.ncsa.uiuc.edu/Apps/SPRNG.

[4] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, "Parallel random numbers: as easy as 1, 2, 3," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), pp. 16:1–16:12, ACM, 2011.

[5] T. J. Urbatsch and T. M. Evans, "The Jayenne IMC project plan," Research Note XTM-RN(U)-98-019, Los Alamos National Laboratory, May 1998. LA-UR–98–2262.

[6] T. Urbatsch and T. M. Evans, "Release notification: MILAGRO-1_0_0," Research Note XTM:RN(U)99-016, Los Alamos National Laboratory, June 4, 1999. LA-UR-2948.

[7] A. G. Petschek, R. E. Williamson, and J. K. Wooten, Jr., "The penetration of radiation with constant driving temperature," Technical Report LAMS–2421, Los Alamos Scientific Laboratory, July 1960.

[8] Y. B. Zel'dovich and Y. P. Raizer, *Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena*. New York: Academic Press, 1966.

[9] B. Su and G. L. Olson, "An analytical benchmark for non-equilibrium radiative transfer in an isotropically scattering medium," *Annals of Nuclear Energy*, vol. 24, no. 13, pp. 1035–1055, 1997.

[10] G. L. Olson, L. H. Auer, and M. L. Hall, "Diffusion, P1, and other approximate forms of radiation transport," in *Proceedings of the Nuclear Explosives Code Development Conference*, (Las Vegas, NV), Oct. 1998. LA-UR-98–5237.

[11] T. Urbatsch and T. Evans, "Release notification: MILAGRO-1_1_0," Research Note X–6:RN(U)-99-033, Los Alamos National Laboratory, October 26 1999. LA-UR-99-5694.

[12] T. Urbatsch and T. Evans, "Release notification: Milagro–1_2_0," Research Note X-6:RN(U)–99–037, Los Alamos National Laboratory, November 12 1999. LA–UR–99–6087.

[13] T. Urbatsch and T. M. Evans, "Release notification: Milstone–1_0_0," Research Note XTM:RN(U)–99–017, Los Alamos National Laboratory, June 28 1999. LA–UR–99–3199.

[14] T. J. Urbatsch and T. M. Evans, "Milstone shunt for the marshak 1D problem," Research Note XTM-RN(U)-99-024, Los Alamos National Laboratory, August 6 1999. LA-UR-99-4420.

[15] T. M. Evans and T. J. Urbatsch, "MILAGRO: A parallel Implicit Monte Carlo code for 3-d radiative transfer (U)," in *Proceedings of the Nuclear Explosives Code Development Conference*, (Las Vegas, NV), Oct. 1998. LA-UR-98–4722.

[16] B. Gough, *GNU Scientific Library Reference Manual*. Network Theory Ltd., 2009.

[17] "CMake." http://www.cmake.org, 2011.

[18] K. Thompson, T. Evans, and R. Roberts, "The Draco Build System," Technical Memorandum CCS-2:12-43(U), Los Alamos National Laboratory, 2012.

[19] "OpenMPI." In Development, http://www.openmpi.org/, 2011.

[20] T. M. Evans and T. J. Urbatsch, "An interface specification for Wedgehog and RAGE," Research Note CCS-DO–005, Los Alamos National Laboratory, December 7 2000. LA–UR–00–5856.