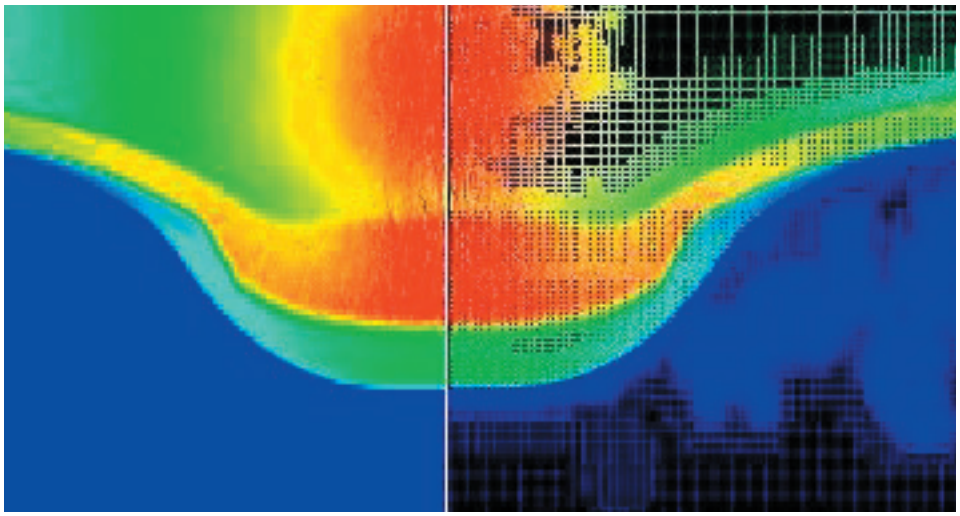
LA-14195-MS
Approved for public release;
distribution is unlimited.

# Milagro Version 2

## An Implicit Monte Carlo Code for Thermal Radiative Transfer: Capabilities, Development, and Usage



Los Alamos
NATIONAL LABORATORY
EST. 1943

The World's Greatest Science Protecting America

Funding provided by the U.S. Department of Energy.

Edited by Keith Pohs, Group IM-1

Cover Photograph: Milagro is a standalone Implicit Monte Carlo (IMC) radiation transport simulation code. Milagro is made up of components that, once verified, are used to build software packages for multiphysics application codes. Here, the IMC software is used with two-dimensional Eulerian adaptive-mesh-refinement hydrodynamics to simulate a hot comet impacting a granite planet.

LA-14195-MS
Issued: February 2006

# Milagro Version 2

An Implicit Monte Carlo Code for Thermal Radiative Transfer: Capabilities, Development, and Usage

Todd J. Urbatsch

Thomas M. Evans

Los Alamos
NATIONAL LABORATORY
EST. 1943

The World's Greatest Science Protecting America

# `Milagro` Version 2
# An Implicit Monte Carlo Code for Thermal Radiative Transfer: Capabilities, Development, and Usage

Todd J. Urbatsch

Thomas M. Evans

Author address:

Transport Methods Group (CCS-4), MS D409, Los Alamos National Laboratory, Los Alamos, NM 87544
*E-mail address*: tmonster@lanl.gov

Transport Methods Group (CCS-4), MS D409, Los Alamos National Laboratory, Los Alamos, NM 87544
*E-mail address*: tme@lanl.gov

*LANL Report Designation:* LA-14195-MS.

ABSTRACT. We have released Version 2 of `Milagro`, an object-oriented, C++ code that performs radiative transfer using Fleck and Cummings' Implicit Monte Carlo 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 Implicit Monte Carlo packages for external customers. `Milagro`-2 represents a design overhaul that allows better parallelism and extensibility. New features in `Milagro`-2 include verified momentum deposition, restart capability, graphics capability, exact energy conservation, and improved load balancing and parallel efficiency. A users' guide also describes how to configure, make, and run `Milagro`2.

# 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, the predominant mesh type being a three-dimensional (3-D), Cartesian, nonuniform, orthogonal, structured mesh. Two-dimensional (2-D) versions of this Cartesian mesh type are used for testing, whereas one-dimensional (1-D) is generally simulated with 3-D and reflecting surfaces. Milagro also runs on an RZ-Wedge mesh, which is a 3-D, Cartesian wedge that models a curvilinear RZ mesh. An external package that utilizes the RZ-Wedge mesh has been released and is in use [2]. 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. The Milagro-2_0_0 release has been redesigned so that parallel, radiation-only calculations are more efficient. We present scaling results for each topology.

In order to facilitate code reuse, we store our reusable classes in a transport library called Draco [3]. The classes that are of general use for Monte Carlo calculations make up the mc component directory in Draco. The classes that are of general use for IMC are stored in the imc component directory. 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 generator, SPRNG [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. Comparisons to analytic solutions are presented.

## 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 nonequilibrium 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].

CHAPTER 2

# Background

## 2.1. Thermal Radiative Transfer Equations

`Milagro` IMC uses the Fleck and Cummings IMC algorithm for radiative transfer [1]. We briefly repeat the radiative transfer equations and the derivation of Fleck and Cummings' IMC algorithm.

We are solving two coupled equations: the equation of transfer and the energy balance equation [16]. The one-group, purely absorbing equation of transfer is

$$\frac{1}{c}\frac{\partial}{\partial t}I + \mathbf{\Omega}\cdot\nabla I = \sigma(\mathbf{x},T)\ \left[B(T) - I(\mathbf{x},\mathbf{\Omega},t)\right]\ , \tag{1}$$

where $I \equiv I(\mathbf{x},\mathbf{\Omega},t)$ is the specific intensity, $B$ is the Planck function, $\sigma$ is the opacity, and $T$ is the material temperature. Equation (1) is correct in the limit of local thermodynamic equilibrium (LTE). The full description of the radiation transfer process requires coupling to the material, whose energy balance is given by

$$\frac{\partial}{\partial t}\mathcal{E}(\mathbf{x},T) = \int_{4\pi}\sigma(\mathbf{x},T)\ \left[I(\mathbf{x},\mathbf{\Omega}',t) - B(T)\right]\ d\mathbf{\Omega}'\ , \tag{2}$$

where $\mathcal{E}$ is the material energy density.

These equations can be solved as they stand [17], but the nonlinearities in the radiation-material coupling can produce instabilities. Fleck and Cummings developed the IMC method, which, in attempting to capture the nonlinear radiation-material coupling, models some of the absorption and reemission as effective scattering. We will briefly go through the derivation of their algorithm. First, we identify the equilibrium radiation energy density, $\phi$,

$$\phi = aT^4\ , \tag{3}$$

where $a$ is the radiation constant. We also note that, in equilibrium, $I = B = \frac{c}{4\pi}\phi$ and, therefore,

$$\int B(T)\,d\mathbf{\Omega} = c\phi(T) = caT^4\ . \tag{4}$$

The nonlinear coupling between the material and the radiation in the problem is captured with the variable $\beta$, which is defined as follows:

$$\frac{\partial\mathcal{E}}{\partial\phi} = \frac{1}{\beta} \tag{5}$$

Holding $\beta$ constant over the timestep effectively linearizes the coupling between the material and radiation during the timestep. Using the preceding definitions in the radiation and material equations, Eqs. (1) and (2), we obtain the following modified equations:

$$\frac{1}{c}\frac{\partial}{\partial t}I + \mathbf{\Omega}\cdot\nabla I = \sigma(\mathbf{x},T)\ \left[\frac{c}{4\pi}\phi(T) - I(\mathbf{x},\mathbf{\Omega},t)\right]\ , \quad \text{and} \tag{6}$$

$$\frac{\partial\phi}{\partial t} = \beta\int_{4\pi}\sigma(\mathbf{x},T)\ \left[I(\mathbf{x},\mathbf{\Omega}',t) - \frac{c}{4\pi}\phi(T)\right]\ d\mathbf{\Omega}'\ . \tag{7}$$

In the equation of transfer, we want a time-centered expression for the radiation energy,

$$\tilde{\phi} = \alpha\phi^{n+1} + (1-\alpha)\phi^n\ , \tag{8}$$

where $\phi^n$ is the radiation energy density at the beginning of a timestep, $\phi^{n+1}$ is the radiation energy density at the end of a timestep, and $\alpha$ is a user-defined variable specifying the implicitness: $\alpha = 0$ is fully explicit and $\alpha = 1$ is "fully" implicit. [Because of approximations in estimating $\tilde{\phi}$ from Eqs. (7) and (8), the Fleck and Cummings method employs quantities evaluated at the beginning of the timestep, and it is not fully

implicit [18].) We substitute the expression for the time-centered radiation energy density, $\tilde{\phi}$, for $\phi$ on the right-hand-side of the modified material equation, Eq. (7). Then we discretize the modified material equation in time by integrating over a timestep, $t^n \leq t \leq t^{n+1}$. To perform the integration, we replace the radiation intensity, $I$, with an initially unspecified time-averaged intensity so that it may come outside of the integral. Next, we solve this equation for $\phi^{n+1}$ and substitute it back into our expression for the time-centered radiation energy density, Eq. (8). This new expression for $\tilde{\phi}$ is used in the modified radiation equation, Eq. (6). Lastly, the time-centered radiation intensity is replaced with the instantaneous radiation intensity producing the following radiation equation:

$$\frac{1}{c}\frac{\partial I}{\partial t} + \mathbf{\Omega} \cdot \nabla I + \sigma I = \frac{1}{4\pi}(1-f)\sigma \int I d\mathbf{\Omega}' + \frac{1}{4\pi} f\sigma c \phi^n \,. \tag{9}$$

Note that Eq. (9) contains an isotropic scattering term even though the medium is purely absorbing. Thus, the Fleck and Cummings' IMC method divides the total absorption opacity into an effective absorption opacity, $f\sigma$, and an effective scattering opacity, $(1-f)\sigma$, using the Fleck factor, $f$:

$$f = \frac{1}{1 + \alpha\beta c \Delta t \, \sigma} \,, \;\; 0 < f \leq 1 \,. \tag{10}$$

Finally, the material energy density is updated using the conservation of energy,

$$\mathcal{E}^{n+1} = \mathcal{E}^n + f \int \int \sigma I \, d\mathbf{\Omega} \, dt - f c \sigma \, \Delta t \, \phi^n \,. \tag{11}$$

We also note that, during a timestep, the time-implicit representation of the Planckian in the Fleck and Cummings method,

$$B \equiv \frac{\tilde{\phi}}{\phi_n} B_n \,, \tag{12}$$

is merely a scaling of the Planckian at the beginning of the timestep [18]. This scaled representation of the Planckian avoids negativities (such as might exist in a Taylor series expansion representation), but, in multigroup, it does not account for any spectral change (as a Taylor series expansion would).

## 2.2. Reproducibility

Since its inception, `Milagro` has had a requirement of reproducibility [19]. Reproducibility means that, for a given input and a given random number seed, a code produces the same solution regardless of parallel scheme or number of processors. Reproducible codes generally have complicated designs and algorithms, longer runtimes, and greater memory requirements. However, reproducibility separates statistical variation from parallelism issues and results in a parallel code that, in the long run, is immensely easier to develop, debug, and verify. Assuming the serial code is verified, a reproducible code is instantly verified for all parallel cases. More practically, verification may be performed with a faster parallel calculation instead of a slower serial calculation. Without reproducibility, every parallel calculation produces a different solution. Verifying that different solutions have the same correct expected value is costly; performing that verification for each new number of processors or new parallel scheme is even more expensive.

There are some arguments against reproducibility for large, time-consuming calculations, which will most likely be run only once. Unfortunately, reproducible code and nonreproducible code are different, and one cannot rely on the other's verification for its own verification. The basis of verification is lost whenever a fully developed reproducible code is abandoned for a nonreproducible code.

Our particular reproducibility is for testing purposes and can be broken by machine precision and roundoff error. Lack of reproducibility in practice does not degrade `Milagro`'s degree of verification. To ensure reproducibility in `Milagro`'s regression tests and verification problems, we operate in a robust part of parameter space by requesting a number of particles that, volume- and energy-weighted, gives approximately whole numbers of particles per cell per processor per source type. Thus, an *X*-*Y*-*Z* infinite-medium, steady-state problem with 100 uniformly sized cells on 1, 2, or 4 processors and with no surface source particles (only emission and census particles) might have 100*4*2=800 particles or any multiple of that (1600, 2400, etc.). The reason for contriving the test problems like this is that different machines can round numbers such as 4.5 differently.

## 2.3. Implicit Absorption

`Milagro` employs an implicit absorption technique when it tracks particles. Implicit absorption is not a new feature in `Milagro` or in general Monte Carlo practice, but we have not yet described it in our documentation. The technique is a form of biasing and is designed to reduce statistical variance and computational effort. As opposed to the discrete nature of analog absorption, implicit absorption is an analytic treatment of the absorption for each particle. It involves a continuous removal of energy-weight from the particle along its path. Sampling a distance to collision involves only scattering (real and effective). Thus, the particle transport is biased to produce longer particle paths. To compensate for this bias, the energy-weight of the particle is exponentially decreased along its path and deposited into the material.

As pointed out to us by H. Grady Hughes, Los Alamos National Laboratory (LANL), Transport Methods Group (CCS-4), the implicit absorption amounts to a modified distance-to-collision probability density function coupled with an application of survival biasing at the collision. The analog particle's distance-to-collision is sampled using the total opacity, which is the sum of the absorption and scattering opacities, $\sigma_t = \sigma_a + \sigma_s$. All opacities are assumed constant for the purpose of this discussion. The analog probability density function for constant opacities is

$$p(x) = \sigma_t e^{-\sigma_t x} \quad . \tag{13}$$

The biased probability density function, which results in larger streaming distances, is

$$p'(x) = \sigma_s e^{-\sigma_s x} \quad . \tag{14}$$

Therefore, in implicit absorption, after a track and before a collision, the particle's energy-weight, EW, must be modified as follows [20]:

$$\mathrm{EW}' = \mathrm{EW}\frac{p(x)}{p'(x)} = \mathrm{EW}\frac{\sigma_t}{\sigma_s} e^{-\sigma_a x} \quad . \tag{15}$$

Then, once the collision has occurred, survival biasing requires that the energy-weight of the particle be multiplied by the scattering ratio:

$$\mathrm{EW}'' = \mathrm{EW}'\frac{\sigma_s}{\sigma_t} = \mathrm{EW}\, e^{-\sigma_a x} \quad . \tag{16}$$

To show that implicit absorption is unbiased, we calculate $E[\mathrm{EW}(s)]$, the expected energy-weight of a particle at a distance $s$ along its path, for both analog and implicit absorption, and show that they are equal. For analog absorption, the particle's energy-weight does not change. The probability that an analog particle survives a distance $s$ without a collision is $e^{-\sigma_t s}$. The probability that an analog particle goes a distance $x$ and then collides is $\sigma_t\, e^{-\sigma_t x}$. Therefore, the probability that an analog particle collides before a distance $s$ and contributes nothing to the expected energy-weight at $s$ is

$$\int_0^s \sigma_t\, e^{-\sigma_t x}\, dx = 1 - e^{-\sigma_t s} \quad , \tag{17}$$

where we note that the probabilities for either making it to $s$ or not making it sum to unity. Thus, for the analog particle, the expected energy-weight at a distance $s$ along its path, $E[\mathrm{EW}_a(s)]$, is

$$E[\mathrm{EW}_a(s)] = 0 \cdot (1 - e^{-\sigma_t s}) + 1 \cdot e^{-\sigma_t s} = e^{-\sigma_t s} \quad . \tag{18}$$

With implicit absorption, the particle's energy-weight is continuously attenuated over its path at a rate of $e^{-\sigma_a x}$. The only discrete events sampled for the implicit particle are scattering events, so the probability that an implicit particle survives a distance $s$ without colliding is $e^{-\sigma_s s}$. Assuming an initial energy-weight of unity, its energy-weight at a distance $s$ is $e^{-\sigma_a s}$. Correspondingly, the probability that it suffers a scattering collision before a distance $s$ and contributes nothing to the expected energy-weight at $s$ is $1 - e^{-\sigma_s s}$. Therefore, the expected energy-weight at $s$ for the implicit particle, $E[\mathrm{EW}_i(s)]$, is

$$E[\mathrm{EW}_i(s)] = 0 \cdot (1 - e^{-\sigma_s s}) + e^{-\sigma_a s} \cdot e^{-\sigma_s s} = e^{-\sigma_t s} \quad , \tag{19}$$

which is equivalent to $E[\mathrm{EW}_a(s)]$.

Implicit absorption will tend to smooth the transfer of energy from radiation to material and reduce the number of required particles. However, the disadvantage of using implicit absorption is that particle weights may drop very low. Computer time is wasted following these low-weight particles. The solution is to truncate the life of low-weight particles. We use the somewhat crude approach of killing a particle when

its energy-weight drops to 1% of its original energy-weight and depositing all of its remaining energy-weight to the material. We call this approach "Full-Clip Russian roulette." In the future, we hope to upgrade this approach to something more sophisticated such as switching to analog tracking. (See the discussion about future work in Chapter 5.)

The time-integrated radiation energy density for a timestep is calculated using an energy-weighted path length (EWPL) tally. Since the energy-weight of a particle is $\mathrm{EW}\, e^{-\sigma_a\, x}$, we have that, over a path length $d$

$$\mathrm{EWPL} = \int_0^d \mathrm{EW}\, e^{-\sigma_a\, x}\, dx \tag{20}$$

$$= \frac{1}{\sigma_a}\left(1 - e^{-\sigma_a\, d}\right) \quad . \tag{21}$$

Earlier IMC efforts neglected contributions to $EWPL$ in voids. However, in a void, the energy-weight is constant and Eq. (20) reduces to

$$\mathrm{EWPL}|_{void} = \mathrm{EW} \cdot d \quad . \tag{22}$$

The energy-weighted path length in voids is treated correctly in `Milagro`.

CHAPTER 3

# Improvements in Milagro-2

## 3.1. Improved Code Design and Activity Timeline

In the time since Milagro was first reported [15], it has undergone redesign. The motivation for redesigning, or refactoring, Milagro was to improve the object-oriented nature of the code and improve the parallel efficiency. We discovered that some of the classes did not fit together as well as we had hoped, and that some of the code was becoming too bulky. Also, the original parallel design of Milagro was overly restrictive; it addressed the issue of a mesh that changes from timestep to timestep, but it was terribly inefficient because the mesh was collapsed to a single processor each timestep. Milagro's improved design parallelizes the mesh and then cycles without collapsing the mesh. Figure 3.1 shows a schematic of the old and new designs. The new design assumes that the mesh will not change over time. Milagro's extensibility

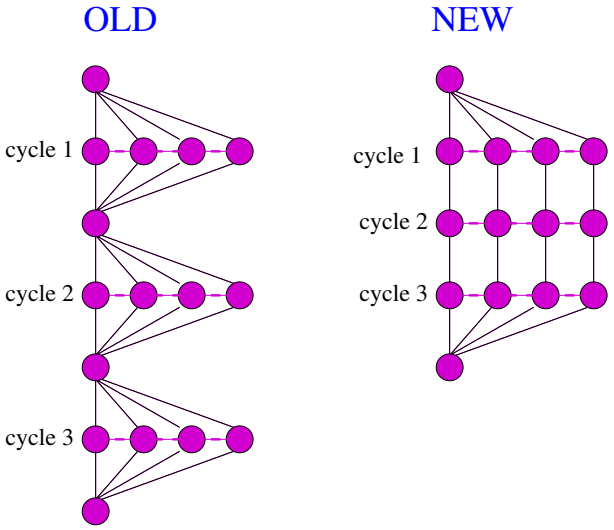



FIGURE 3.1. Schematic of Milagro's new parallel design.

allows future development of a capability to handle a dynamically changing mesh in parallel. However, this capability is not currently needed to satisfy Milagro's intended purpose as a verification tool and research testbed.

Levelized design ensures that a code has no components with physical or link-time cyclic dependencies [21]. Levelized design also allows for rigorous component testing. Tested components may then be used with confidence by higher level components. The levelized design of Milagro's components is shown in Fig. 3.2. The highest level executables represent Milagro templated on various mesh types.

The activity diagram for Milagro, Fig. 3.3, shows when classes are instantiated in Milagro's new design.

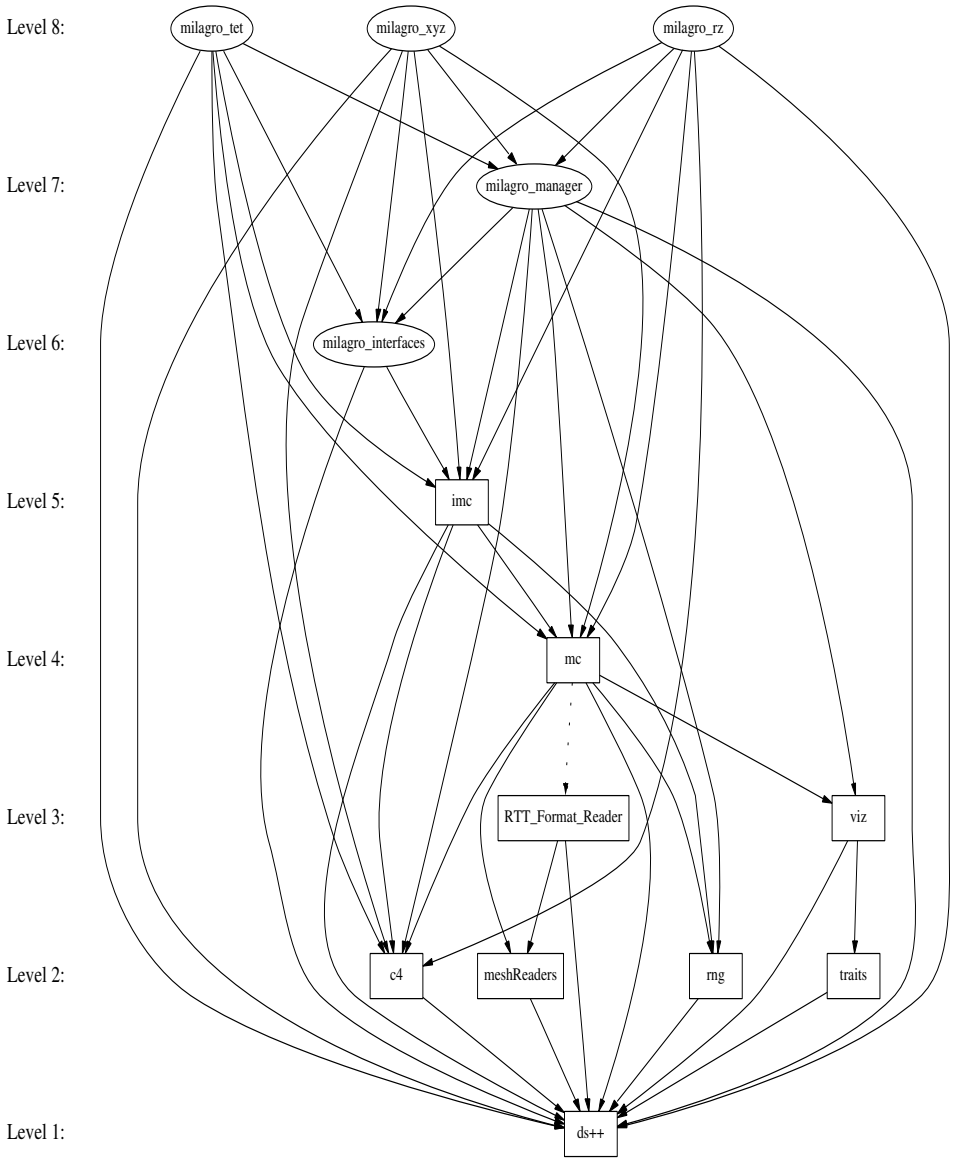


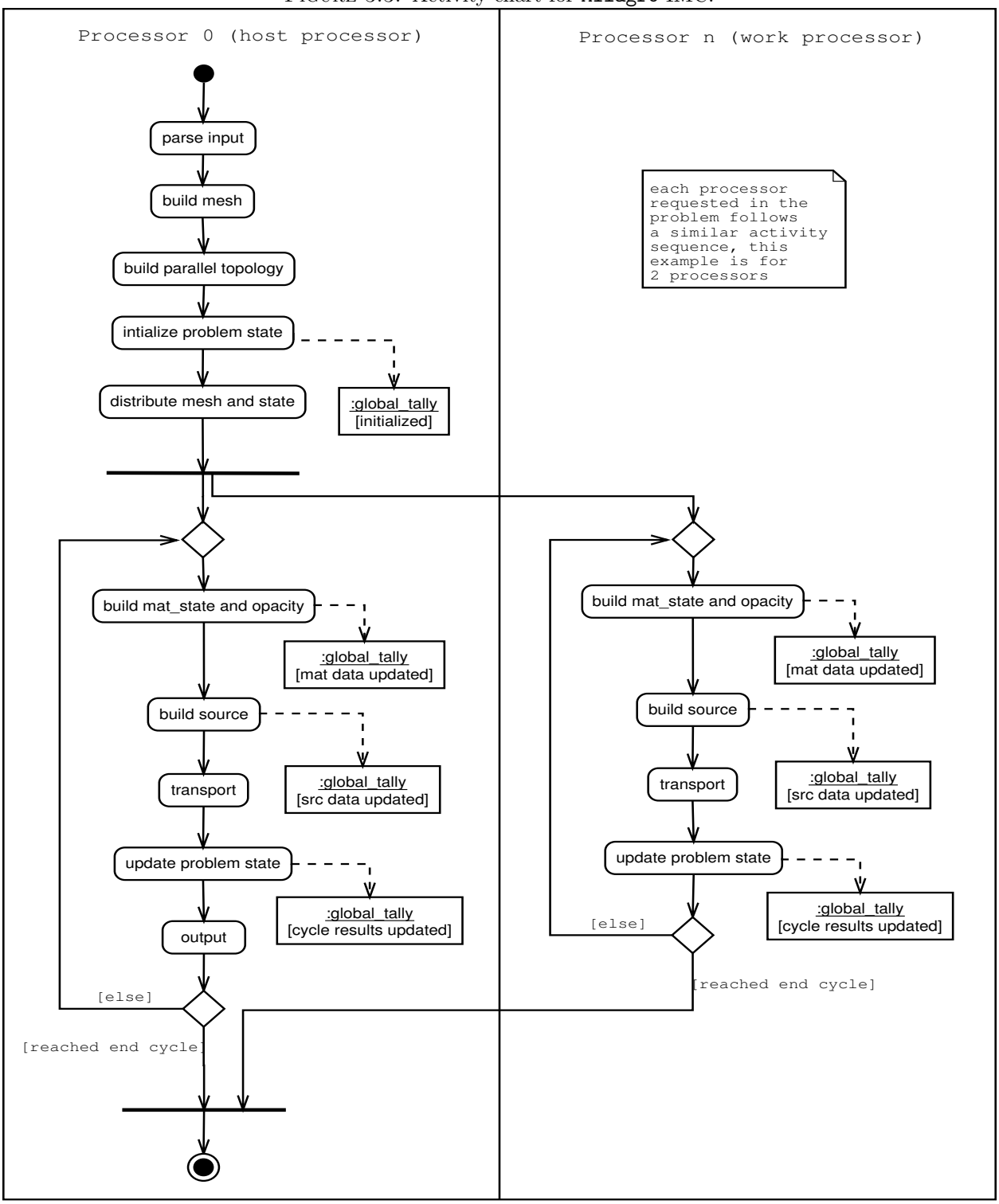

FIGURE 3.2. Levelized design for `Milagro` and its components.

### 3.2. Energy Conservation

`Milagro`'s IMC particles conserve energy both globally and locally. At first glance, energy conservation seems trivial since the Fleck and Cummings IMC method conserves energy [1, 18]. However, there are conservation issues separate from whether a numerical method conserves energy. For example, updating the material temperature while assuming a constant specific heat will affect conservation and stability [22]. The issue we will discuss is that of energy conservation in the particles. The difficulty in conserving particle energy comes from attempts to manage the number of particles while maintaining reproducibility.

At the beginning of each timestep, or cycle, there exist "census" particles that are still alive and in flight from the end of the previous timestep (or from the initial census on the first timestep). The energy-weights of these particles are highly varied and, over time, the number of census particles tends to rise to undesirable values. So at the beginning of each time step, the existing census particles are combed, a process originally

FIGURE 3.3. Activity chart for `Milagro` IMC.




developed by Canfield [23]. Historically, the comb involves stacking the energy-weights of particles end-to-end and running a "comb" with equally spaced prongs through the lineup. The particles' energy-weights that

the comb's teeth hit are retained. The surviving particles are each assigned the same, new energy-weight such that the total energy-weight is conserved. Thus, Canfield's historical comb globally conserves energy, minimizes variance, and controls particle numbers.

The historical comb, in order to be reproducible, requires all the census particles to be lined up in the same order regardless of number of processors or parallelization topology. Though not impossible, this daunting requirement led us to replace the historical comb with a Russian-roulette approach, where each particle samples its own fate based on its own random number state, its own energy-weight, and the desired census energy-weight after combing. While this approach minimizes variance and controls particle numbers, it only statistically conserves global census energy.

We added two techniques that allowed the Russian-roulette approach to conserve energy both globally and locally. First, if all the census particles are Russian-rouletted from a cell, a dead census particle is resurrected and given the cell's census energy. If there are multiple dead census particles, the random stream identification number is used in a consistent way to determine which one comes back to life. (This selection appears to be unbiased, but, even if it is not, the bias should be smaller than entirely neglecting the energy-weight.) Second, after Russian roulette and any necessary resurrection, the weights of the surviving census particles are uniformly readjusted to match the census energy in their respective cells.

These added techniques produce an overall combing strategy that is reproducible, controls the number of census particles, reduces the variance in energy-weights, and conserves global energy and local energy per cell. Its advantage over the historical comb is that it is easily reproducible and locally conserves energy. Its shortcoming is that, by retaining census particles with small energy-weights, it does not fully minimize variance. This combing strategy still requires the user to request an adequate number of particles; it cannot compensate for inadequate sampling.

It was pointed out to us by Jim Morel, CCS-4 LANL, that the comb does not conserve momentum. We may simply tally the change in momentum when a particle is combed into or out of existence. This tally would not change the inaccurate and undue momentum change from the comb, but it would account for the change.

### 3.3. Momentum Deposition

When a radiation package is built to perform the radiation portion of an operator-split radiation-hydrodynamics calculation, the radiation package must provide both energy and momentum deposition to the hydrodynamics code. Momentum deposition was implemented in `Milagro` for this release [24]. From Mihalas and Mihalas [25], the momentum of a photon with energy $h\nu$ and traveling in direction $\mathbf{\Omega}$ is $(h\nu/c)\mathbf{\Omega}$, where $h$ is the Planck constant, $\nu$ is the frequency of the photon, and $c$ is the speed of light. The net radiative momentum transport across a differential surface area, $d\mathbf{S}$, is $(1/c)\mathbf{F}\cdot d\mathbf{S}$, where the radiation flux, $\mathbf{F}=\int\int \mathbf{\Omega} I\, d\nu\, d\Omega$, is the first angular moment of the specific intensity, $I$. The net momentum *deposition*, then, from the radiation to the material is $\sigma\mathbf{F}/c$, where $\sigma$ is the macroscopic cross section [26].

The momentum deposition tally in `Milagro` accumulates momentum deposition whenever a particle is emitted, absorbed, or scattered. The degree to which this tally is analog or implicit corresponds to the degree to which the particle is analog or implicit. In `Milagro`, the absorption of a particle's energy-weight is implicit, so the momentum deposition tally is implicit as well. Conversely, the momentum deposition tally is analog for scattering, emission, and low-weight killing (a point-wise absorption). Table 3.1 lists the events and corresponding quantities accumulated for the momentum deposition tally. The net momentum deposition

TABLE 3.1. Analog Scoring of Momentum Deposition from Radiation to Material, Where $ew$ is the Particle's Energy-Weight and $\Omega$ is a Particular Direction Cosine

| Event | Score |
|---|---|
| volume emission | $-ew\,.\,\Omega$ |
| time-rate absorption | $\Delta ew\,.\,\Omega = (ew_{old} - ew_{new})\,.\,\Omega$ |
| scatter | $ew\,.\,(\Omega_{old} - \Omega_{new})$ |
| kill due to low $ew$ | $ew\,.\,\Omega$ |

from radiation to material per unit time and per unit volume is obtained by dividing the accumulated energy scores from Table 3.1 by the speed of light $c$, the timestep $\Delta t$, and the volume of the cell, $V_c$:

$$\Delta \boldsymbol{p}_{dep} = \frac{1}{c\Delta t\, V_c} \sum_{events} ew_c \cdot \boldsymbol{\Omega} \ . \tag{23}$$

The momentum deposition in `Milagro` was verified on three test problems: a steady-state, infinite medium and two Marshak Waves [24]. The results were compared to the analytic momentum deposition from Mark Gray's Analytical Test Suite[1]. Figure 3.4 shows the momentum deposition into the slab for the Marshak-1D test problem. The Marshak-1D test problem has a delta function source at time zero and zero depth into the slab. The `Milagro` results tend to admit the analytic solution, albeit with some noise.

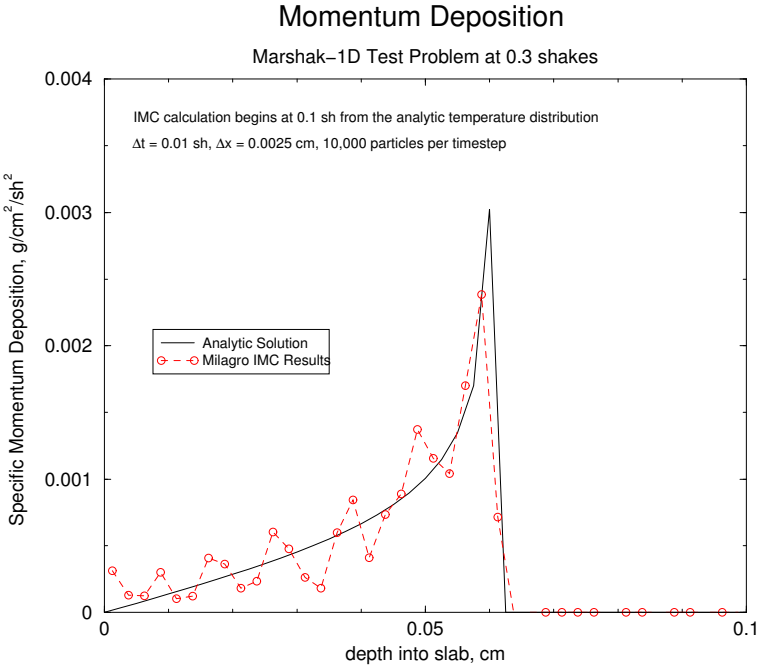



FIGURE 3.4. Momentum deposition in the Marshak-1D test problem.

Differences at the wavefront are due to resolution differences and the fact that our initial cold temperatures are not exactly zero as they are in the analytic calculation.

## 3.4. Parallelism

`Milagro`'s parallel capability is based on message-passing communication. It does not currently utilize threads or shared memory. `Milagro` runs with either of two limiting cases of parallel data distribution: mesh replication (particles distributed) and mesh decomposition (both mesh and particles distributed).

**3.4.1. Philosophies of Serialism and Parallelism.** Both serial speed and parallel scalability are important. However, we tend to place a somewhat higher value on serial speed than on scalability because parallel speedups can be directly improved by serial speedups. It is true that an improvement in serial speed can reduce the workload per processor and therefore decrease parallel efficiency. However, the same logic implies that improved serial speeds can reduce the number of required processors or allow for more particles to be run in the same amount of time. (The reduction in required processors is especially good if the user is part of a large society of users that shares a parallel computer.) In further accordance with this philosophy, we will not artificially decrease serial speed to gain parallel efficiency unless, of course, overall efficiency can be improved.

[1]The Analytical Test Suite is a CCS-4 application used to verify radiation transport packages.

Adhering to that philosophy, we try to avoid increasing the workload per particle. For instance, if every particle needs the same certain piece of information, we will calculate it once and store it. Thus, with the added small cost of some extra memory, computer time doesn't need to be wasted repeatedly calculating the same information for each particle. Sometimes the reduction in a particle's work does not involve a tradeoff with increased serial work; it may only require a more intelligent use of available data. This philosophy is sound considering that, in most calculations, the number of particles is much greater than the number of cells.

**3.4.2. Topologies.** "Topology" is the term we use to associate the mesh and the processors. `Milagro` handles each of two limiting cases of topology: full replication and full domain decomposition. Full replication is the parallel scheme in which each processor has immediate access to the entire mesh. Full domain decomposition is the parallel scheme in which the mesh is divided among the processors, and each processor has access to a unique part of the mesh.

Full replication is ideal for Monte Carlo particle transport simulations because no cross-processor communication is required during transport. Unfortunately, extremely large meshes will not fit on the memory available to a single processor. In those cases, domain decomposition is necessary. Domain decomposition is the preferred topology for deterministic methods, where it is known beforehand how much work each mesh cell requires. In `Milagro`, calculations with a decomposed domain need to be contrived so that the number of cells divides evenly among the processors. This limitation does not hinder our verification efforts, and, moreover, it is not present in our delivered IMC packages.

We have considered, but not yet fully implemented, a general topology that would allow replication of important parts of the mesh and decomposition of less important parts [27].

**3.4.3. Memory Models.** `Milagro` was designed using a distributed memory model. Distributed memory means that each processor has access to only its own memory and must communicate to exchange information. This communication is facilitated through the use of a standard, portable message-passing library definition such as the Message Passing Interface (MPI) [28]. A distributed memory model would be appropriate for a network of single-node workstations.

Some computer systems allow processors to access the same memory and are thus appropriate for shared memory models. Some systems consist of several distributed "boxes," where each box contains several processors that access shared memory within the box. These systems are amenable to a mixed-memory model. The mixed-memory model allows optimization of memory usage by using distributed memory where necessary and shared memory where possible.

There is nothing technically wrong with using a distributed memory model on a shared-memory system, but it can be memory-inefficient. Consider a full replication topology using a distributed memory model on a shared memory system. The entire mesh is replicated on every processor, so the system has redundant copies of the mesh. For sufficiently large meshes, full replication on a shared-memory system may not be possible.

Memory inefficiency is one motivation for utilizing shared memory with something like threads or OpenMP on a shared-memory system. We could emulate a full replication topology by having one copy of the mesh while each thread simultaneously transported particles.

Our ultimate goal is to use a hybrid-memory model on a mixed-memory system. We would fit as much of the mesh as possible on each box and run threads or OpenMP on each box.

**3.4.4. Source.** `Milagro` uses a source builder from the `imc` package. Building the source consists of three successive parts: calculating energies, calculating numbers of source particles, and setting the random number stream offsets in each cell.

The source builder calculates the energy per cell for each of the three species of source particles: volume emission, surface source, and census. The census energy is deterministically calculated from the initial radiation temperature only on the first cycle; thereafter it is known from the previous cycle.

The number of source particles per cell and per species is calculated as an ensemble using an approach similar to the iterations of an eigenfunction calculation. The initial eigenvalue is the "particles per unit energy," which is calculated from the user-requested total number of particles and the total source energy. After each iteration, the sum of the numbers of particles is compared to the user-requested number of

particles. If there are too many particles, the eigenvalue is reduced accordingly, and another iteration is performed. This scheme tends to adequately converge in a few iterations. Once the number of particles in a cell for a given species is known, the corresponding energy-weights are easily calculated. Energy losses due to inadequate sampling are also tabulated.

To retain reproducibility, each source particle maintains its own random number generator, which has a unique identification number (ID). A random number generator ID offset is calculated for each cell and each source species. The offsets are numbered on the global mesh, beginning before the first cycle with the initial census particles. Then, for each cycle, the numbering continues with the volume emission particles and then the surface source particles. Particles due to external sources are lumped into the volume emission.

For full replication topologies, each processor performs exactly the same calculation for global source energies and numbers, all without interprocessor communication. Having knowledge of the number of processors and its own unique processor ID number, each processor calculates its own random number generator ID offsets and local source numbers. For a given source species in a given cell, the global number of particles is spread evenly over all the processors. Leftover particles are distributed to an equivalent number of processors; for each cell, leftover particles are distributed beginning with the first processor available after the previous cell's leftover particles. Thus the particles are very nearly exactly load balanced. This exact load balancing is perturbed somewhat by the census comb and, as always, by the fact that particles take different amounts of time to run. Because the source calculation occurs on every processor, it contributes to the serial fraction of the runtime and can degrade scalability.

For full domain decomposition each processor's cells are unique. The source calculation proceeds according to the serial description, except that communication is required in several places and each processor only calculates the source for its own cells. Communication is required for calculating global energies, global total numbers of source particles, and energy loss tabulations. Calculating the random number generator ID offsets requires temporary global-mesh-sized arrays on each processor and full communication between the processors.

**3.4.5. Asynchronous Transport.** For particle transport in a domain decomposition topology, each processor continuously loops over a set of mutually exclusive options until every particle finishes. Each processor attempts to dynamically prioritize its own source particles and incoming particles from other processors. Initially, the first source particle has the highest priority. After $N_{\mathrm{src}}$ source particles, the communicator is checked to see if any particles have arrived from neighboring domains on other processors. If incoming particles have arrived, they are put into a bank and immediately run to completion without checking for more incoming particles. During transport, particles that leave the processor's domain are buffered and eventually sent to the appropriate processors. When a processor has no more source particles or incoming particles, it deliberately flushes its buffers and checks for more incoming particles. When there appear to be no more incoming particles, the processor makes any updates to the global count of finished particles. When all particles have been completed, the master processor broadcasts the finished status to all the processors[2].

The logic of `Milagro`'s asynchronous transport is demonstrated in the following actual code:

```
// transport particles
while (!finished)
{
  // transport a source particle and any incoming particles
  if (*source)
    trans_src_async(check, bank, new_census_bank);

  // transport an incoming particle from another domain (processor)
  else if (bank.size())
    trans_domain_async(check, bank, new_census_bank);
```

[2]Using the number of finished particles as a criterion for stopping asynchronous transport was suggested by John Fao, Lawrence Livermore National Laboratory, and relayed to us by Jim Rathkopf and Forrest Brown. This criterion seems obvious, but, for some reason, many Monte Carlo practitioners failed to see it!

```
    // if send-buffers are not empty, flush them
    else if (communicator->get_send_size())
      communicator->flush(*buffer);

    // receive particles as second-to-last option
    else if (communicator->arecv_post(*buffer, bank))
    {
      int bsize = bank.size();
      Check (bank.size() > 0);
    }

    // report num_done back to master; see if we are finished
    else
      update();
  }
```

where the transporter functions are defined by the following pseudo-code:

```
trans_src_async(check, bank, new_census_bank)
{
  transport N_src source particles;

  if communicator has incoming domain particles
  {
     put them into the bank;
     while (bank.size() > 0)
       trans_domain_async(check, bank, new_census_bank);
  }
}

trans_domain_async(check, bank, new_census_bank)
{
  transport N_src domain particles;

  if communicator has incoming domain particles
  {
     put them into the bank;
  }
}
```

We have found that setting $N_{\rm src}$ to the buffer size resulted in the formation of too many MPI buffers. For greater robustness and some loss in efficiency, we set $N_{\rm src} = 1$. The optimal value of $N_{\rm src}$ is somewhere between 1 and the buffer size and depends on the number of neighboring processors and load balancing.

**3.4.6. Communication Buffer Size.** The user specifies the size, in particles, of the communication buffer. The buffer size is important for asynchronous transport, when particles must be communicated between processors. The necessity of a buffer size greater than unity is due to the fact that communication costs are generally higher than computing costs and the fact that, up to a certain point, it costs just as much to send small amounts of data as it does larger amounts of data. The optimal buffer size depends upon such quantities as the number of particles, the load balancing, the communication-to-work ratio, and the trends in any of these quantities over a timestep. For example, if most of the particles began on one processor, a small buffer size would get other processors working as soon as possible. However, when the workload is balanced, a small buffer size could result in too many buffers (environment variables may limit the number of buffers).

**3.4.7. Scaling Results.** We investigate the scaling properties of `Milagro` with two different problems. The problems are designed to highlight the differences between the two parallel topologies, domain decomposition and replication. Each problem has a constant amount of work because of the constant number of particles. We consider a fairly large number of particles in order to mitigate the diminishing amount of work as the number of processors increases. Future studies will consider a constant number of particles per processor.

The first problem is a hot, steady-state, infinite medium, which we represent as a cube with reflecting boundary conditions. Figure 3.5 shows the constant-work scaling for this problem. The calculations using

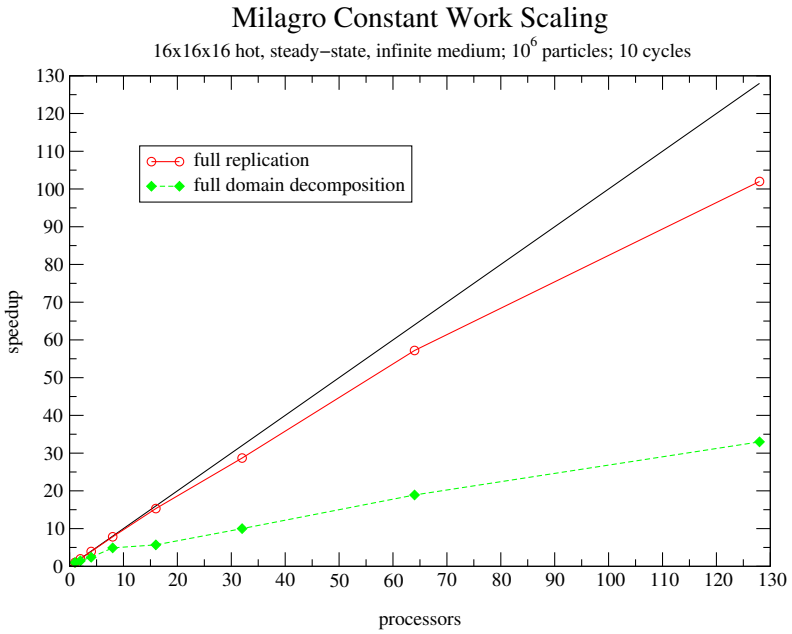



FIGURE 3.5. Constant work scaling for replicated and decomposed spatial domain in a hot, steady-state, infinite medium.

full replication scale well out to 128 processors. The domain decomposition topology did not scale as well. For both topologies, the transport workload is adequately balanced between the processors. However, this problem isolates the in-cycle communication cost incurred in a decomposed domain. As the communication-to-transport ratio approaches zero, the domain decomposition approaches the efficiency of the full replication. The communication-to-transport ratio would decrease if communication speeds were increased or if the medium was optically thick, in which case the Fleck and Cummings' method becomes effectively diffusive. The domain decomposition suffers from an additional degree of freedom in that the user must select a buffer size for communication between processors. In order to balance the size of each message and the number of messages, we varied the buffer size from 100,000 for 2 processors to 20 for 128 processors.

The second problem is basically an early-time Marshak wave. The scaling results are shown in Figure 3.6. The domain decomposition topology performs horribly for two reasons. First, the wave problem is severely load imbalanced because all the work is done by one or two processors. Second, we intentionally decomposed the mesh in an unintelligent way; orthogonal to—not along—the direction of the wave propagation where it does little good. Surprisingly, the full replication also performed poorly. Eye-balling the asymptotic limit of the curve, we use Amdahl's Law [29] to infer that the ratio of parallelizable-to-serial coding is about 24. Indeed, timings of the entire calculation and the transport parts alone also produced that ratio. The cold material absorbs particles quickly, so the actual transport work per particle is miniscule, resulting in degraded scalability. As the radiation propagates into the material, the material will heat up and the parallelizable-to-serial ratio will increase so that this problem scales as well as the first problem.

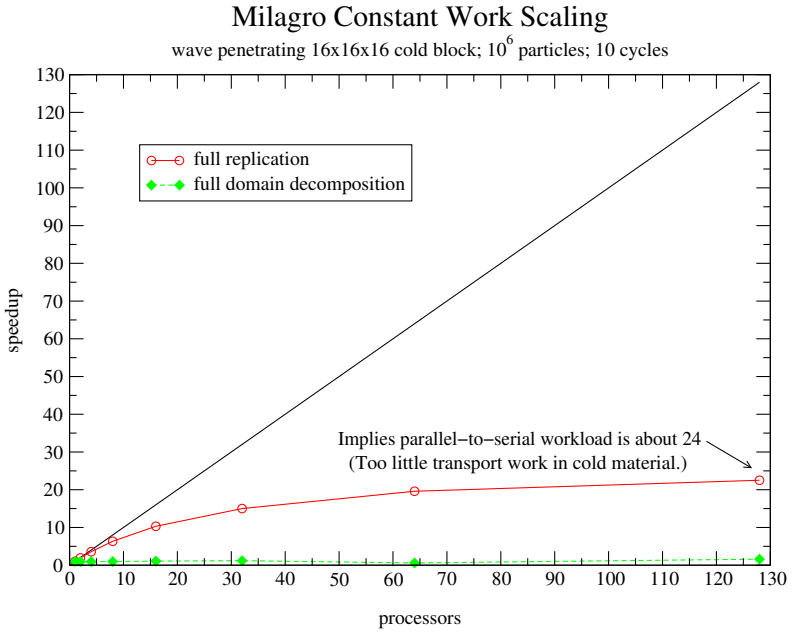



FIGURE 3.6. Constant work scaling for replicated and decomposed spatial domain in a Marshak Wave.

**3.4.8. Utilizing Shared Memory.** In the future, we will try to address the potentially significant fraction of serialism in the full replication parallelism by coupling MPI and OpenMP in a shared- or mixed-memory model. This hybrid parallelization is called OpenMPI [30]. OpenMP threads parallelize computations in shared memory. OpenMP threads are efficient and easy to implement on loops, such as `Milagro`'s loops over cells when it deterministically calculates the source. Applying OpenMP to C++ classes such as `Milagro`'s particle class is less straightforward and, at this point, requires some minor advances in compiler technology (which are forthcoming).

Let us assume that the total workload, $W$, for a `Milagro` calculation can be divided into three parts, $W = S + H + P$, where $S$ is truly serial workload, $H$ is the threadable workload consisting of loops, and $P$ is the particle transport workload which is both MPI-parallelizable and threadable. For MPI parallelism alone, as demonstrated in Sec. 3.4.7, the theoretical maximum speedup (assuming zero communication cost) for $N$ processes (one process per MPI node) is

$$\text{Speedup}_{\text{MPI}} = \frac{S + H + P}{S + H + P/N} \quad . \tag{24}$$

For OpenMP parallelism alone with $T$ threads applied both to loops and to `Milagro`'s particles, the theoretical maximum speedup is

$$\text{Speedup}_{\text{OpenMP}} = \frac{S + H + P}{S + (H + P)/T} \quad , \tag{25}$$

which, for $T = N > 1$, is greater than MPI parallelism. Unfortunately, most shared-memory computers have non-uniform memory access (NUMA), which reduces OpenMP's realized speedups. Hopefully, the randomness of the starting and ending times of `Milagro` particles will buffer the adverse effects of NUMA.

Ideally, a hybrid OpenMPI approach would associate a single MPI processor with a box of processors that utilize shared memory. On each box would exist $T$ threads for both the loops and the particle transport. Thus, for full replication, the theoretical maximum speedup is

$$\text{Speedup}_{\text{OpenMPI}} = \frac{S + H + P}{S + H/T + P/T/N} \quad . \tag{26}$$

A hybrid OpenMPI approach could also be used on a single shared-memory box to dictate some degree of processor affinity, which would combat the effects of NUMA. OpenMPI also would optimize memory usage and efficiency for domain decomposition parallelism in `Milagro`.

## 3.5. Restart

Version 2 of `Milagro` has the ability to restart calculations. The user specifies the frequency with which `Milagro` makes restart dumps (actually, the user specifies the inverse frequency, in cycles). `Milagro` constructs a restart directory whose name is the problem title concatenated with "_restart". For each restart dump, `Milagro` puts two binary files in the restart directory—one with restart data, and one with census data—each appended with the cycle number (e.g., "census.300" and "restart.300").

A separate restart input file is required for actually restarting. This restart input file must contain information about the mesh file, the title, and the restart cycle. Other parameters may also be changed upon restart; they are outlined in Section 6.4.

A restart capability is necessary for run continuations and for recovering from time-limits and crashes. However, `Milagro`'s restarting capability can also be used for acquiring postcalculation graphics and as a mechanism to change the parallel topology in the middle of a calculation.

## 3.6. Graphics

By utilizing existing `Draco` visualization components, `Milagro` now has the capability to provide EnSight graphics. The user specifies how often, in shakes, the graphics data should be dumped. The data is dumped into subdirectories under the directory "<title>_ensight". Graphics data include the geometry data (which the mesh must provide) and the following cell data: density, radiation energy and temperature, and material energy and temperature. The user may also specify graphics regions that are subsets of the full system.

In order for graphics dumps to be made at any point in a calculation, `Milagro`'s Graphics_Manager must be instantiated at the initialization of the calculation so that the graphics subdirectory can be created. We recommend that, in production-type calculations, graphics always be turned on. If graphics are not desired, "graphics_dump:" can be set to a very large value. That way, the subdirectory is created, but dumps are not actually made. If graphics are deemed necessary at a later time, the calculation can be restarted with an appropriate value of "graphics_dump:".

Modification of "graphics_dump:" in a restart calculation is a little tricky. On a restart, the next graphics dump will occur at a time equal to the time of the last graphics dump plus "graphics_dump".

CHAPTER 4

# Verification

## 4.1. Software Development Practices that Enable Verification

Our ultimate goal is to provide our customers with verified IMC packages. Verification ensures that a code correctly solves the equations it intends to solve.

Many of the software practices we use are meant to ease, and sometimes facilitate, code verification efforts. At the lowest level, the function level, we use Design-by-Contract™[1] (DBC) [31], a practice that utilizes C++ assertions to ensure that a function receives the proper information, that it is performing its job as expected, and that it returns the proper data. By using a levelized, object-oriented design, we can make use of component tests. Component tests externally ensure that functions are performing as expected. Levelized design [21] implies that tested components can be confidently used by higher level components. We utilize automatic nightly regression tests so that we can manage code development over time [32]. We also use regression testing as a forum for testing code or physics components from the compiled-code level. At the highest, compiled-code level—the `Milagro` executable—we compare results to analytic and semianalytic test problems. When we build IMC packages, we simulate the customer with a "shunt" so that we can verify the interface to our package. When the customer correspondingly verifies their side of the interface by simulating the IMC package, overall debugging efforts scale algebraically instead of geometrically (i.e., $nx$ instead of $x^n$). Again, without reproducibility, the required verification efforts would be staggeringly more extensive.

## 4.2. Physics Verification

We have compared `Milagro` results to several benchmark problems with analytic solutions. These problems include the following:

- steady-state, infinite, homogeneous medium: several variations
- streaming problems
- constant material volume source
- time and space equilibration
- Marshak Waves: four variations
- Su/Olson nonequilibrium transport benchmarks: two variations

Both the streaming problems and the steady-state, infinite, homogeneous medium problems are uninteresting, but they are excellent tools for verifying some of the necessary (but not sufficient) correctness of the code. `Milagro`'s nightly regression tests include 30 infinite medium problems and 31 streaming problems. We will not present any of these simple test results here.

Figure 4.1 shows the `Milagro` temperature results compared to an analytically derived, thermal equilibrium temperature for a constant material energy volume source in an infinite medium. See the the `Milagro-1_2_0` release note for a detailed description of this problem [12].

While performing research on hybrid methods, we had an opportunity to run `Milagro` on a space-time equilibration problem [33]. This problem is a 1-D infinite medium modeled as a finite medium with reflecting boundary conditions and a linearly sloped initial temperature distribution. Over time, the temperature equilibrates to a constant temperature that can be computed using the conservation of energy. Figure 4.2 shows the time-dependence of the temperature in the left-most cell and the right-most cell. `Milagro`'s time-

[1] "Design by Contract" is a trademark of Interactive Software Engineering.

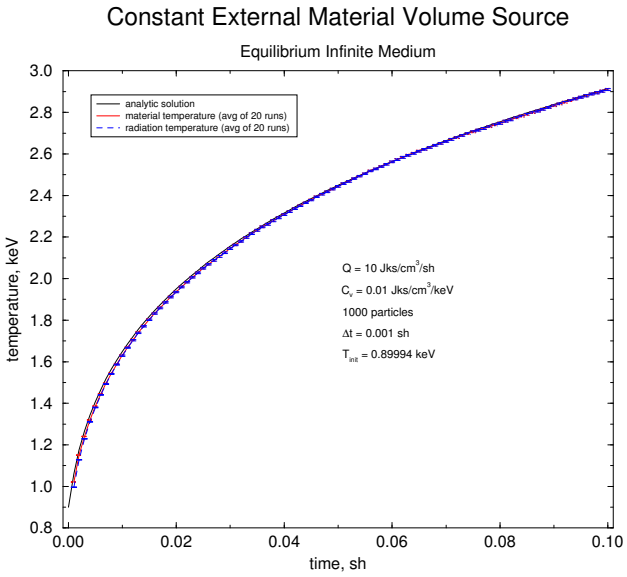



FIGURE 4.1. The equilibrium temperature for a constant external material volume source.

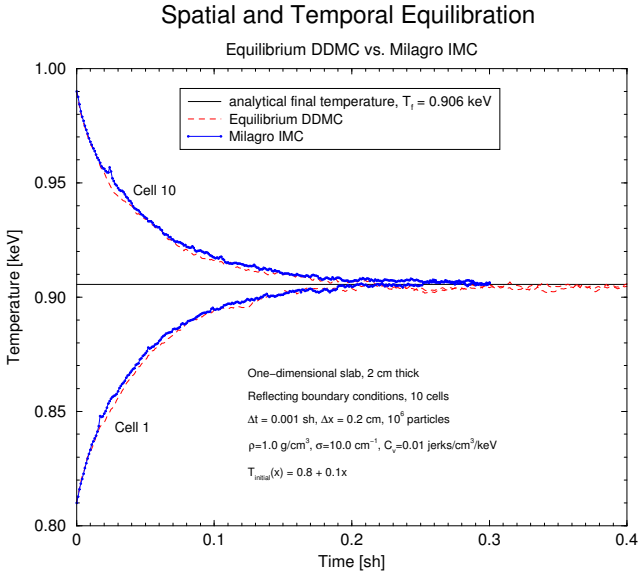



FIGURE 4.2. The equilibrated temperature for a finitely modeled infinite medium with a linear initial temperature distribution.

equilibrated results agree with the analytic results, and the space- and time-equilibrated results agree with the new method's results.

The Marshak wave analytic test problems [7, 8] make a very useful set of 1-D verification problems. All the Marshak wave problems model radiation propagating through a cold slab of homogeneous material with an opacity that is proportional to $T^{-3}$. We consider only the 1-D slab results for the Marshak problems.

One variant of these analytic problems, Marshak-1D, has a delta function source in space and time. In the Marshak-1D problem, we avoid the complications of modeling a delta function by starting the calculation

at 0.1 shakes with analytic data from Mark Gray's Analytic Test Suite[2]. Figure 4.3 shows the `Milagro` results for the material temperature compared to analytic results from the Analytic Test Suite.

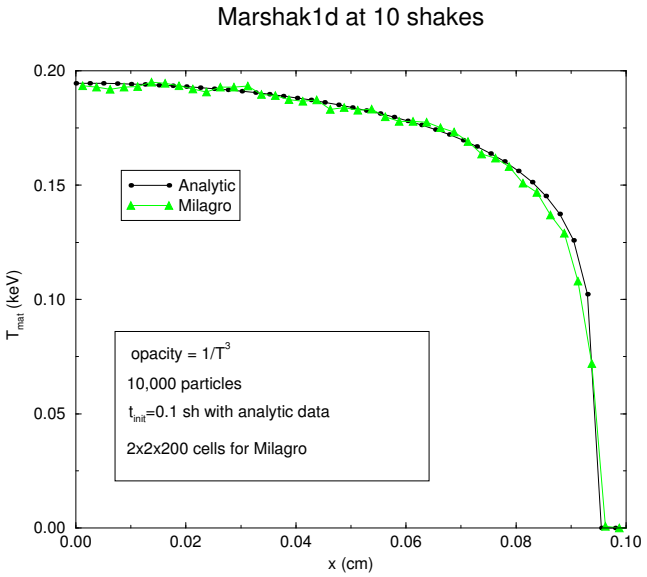



FIGURE 4.3. `Milagro` results for the Marshak-1D problem.

Another variant, Marshak-2B, has a constant 1 keV blackbody radiation flux impinging on the cold slab. The Marshak-2B problem has an absorption/emission coefficient of $100/T^3$ cm$^2$/g. Figure 4.4 shows how `Milagro` results for the Marshak-2B problem compared to analytic data that was obtained from a fourth order Runge-Kutta code written by Don Shirk, Diagnostics ApplicationsGroup (X-5), LANL.

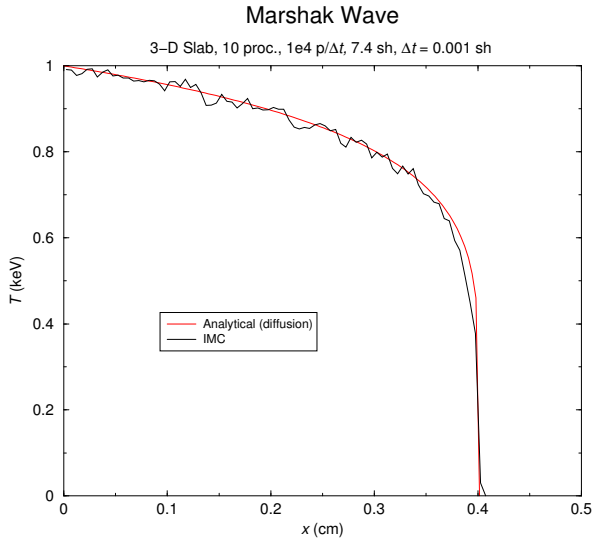



FIGURE 4.4. Marshak-2B results from `Milagro`.

The Marshak-2A problem is exactly the same as the Marshak-2B problem except that it has an absorption/emission coefficient of $10/T^3$ cm$^2$/g. It turns out that this small absorption/emission coefficient violates

[2]The Analytical Test Suite is a CCS-4 application used to verify radiation transport packages.

the equilibrium diffusion approximation and results in poor analytic results. `Milagro`'s results, shown in Fig. 4.5, compared well with results from deterministic transport codes.

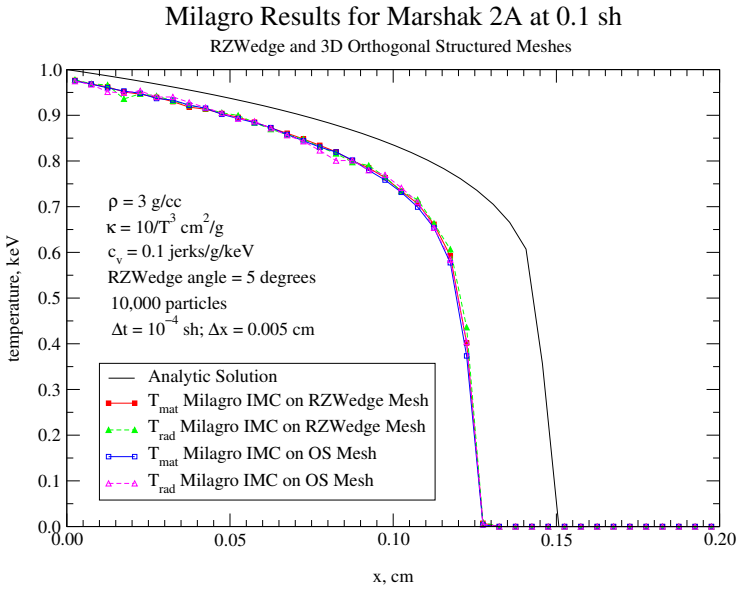



FIGURE 4.5. Marshak-2A results from `Milagro`.

The analytic solutions to the Marshak waves assume that the material energy is much greater than the radiation energy, $\rho\, c_v >> 4aT^3$. In the case of Marshak-2B, this equality is at most $0.3 >> 0.05488$. In order to improve the validity of this assumption, we recast Marshak-2B with a specific heat capacity that is an order of magnitude larger such that the inequality is $3.0 >> 0.05488$, which is an order of magnitude more valid. `Milagro`'s material temperature for this modified problem, Marshak-2B', is plotted against Mark Gray's analytic results in Fig. 4.6.

Su and Olson have determined analytic transport solutions for nonequilibrium radiative transfer [9]. Their problem is half-space and contains a radiation source of finite space and duration. The Su/Olson problem is totally linear because it has specific heats proportional to the cube of the material temperature, $c_v \propto T^3$. One version of the problem has no scattering; another has 50% scattering. We compare material and radiation energies from `Milagro` on both the orthogonal structured (OS) mesh and the RZWedge mesh to the analytic solution for both the purely absorbing and the 50% scattering cases in Figs. 4.7 to 4.10.

The material temperature updates in `Milagro` normally assume that the specific heat is constant over a timestep (wrong for the Su/Olson problem) and can be evaluated at the beginning of the timestep (inaccurate when specific heat is not constant and unstable when the specific heat is near zero). For the Su/Olson problem where the specific heat is nonconstant but known, `Milagro` now analytically updates the material temperature [22].

### 4.3. Comparison Problems

In addition to problems with analytic solutions, there exist problems whose numerical solutions represent the correct solution with a large degree of confidence. These problems do not serve the same role as analytic solutions, but they provide some low-level confidence in the transition from verification to validation. One such problem is the solution published by Olson, Auer, and Hall [10]. They consider a variant of the Marshak wave and present their "best" results, which came from a Variable Eddington Factor (VEF) method that iterated on the time-implicit opacities. We mocked up the problem along the $x$-direction with $\Delta x = 0.01$ cm and one 10-cm-thick cell in each of the transverse directions. We used a timestep of $0.01ct$ cm, which relates

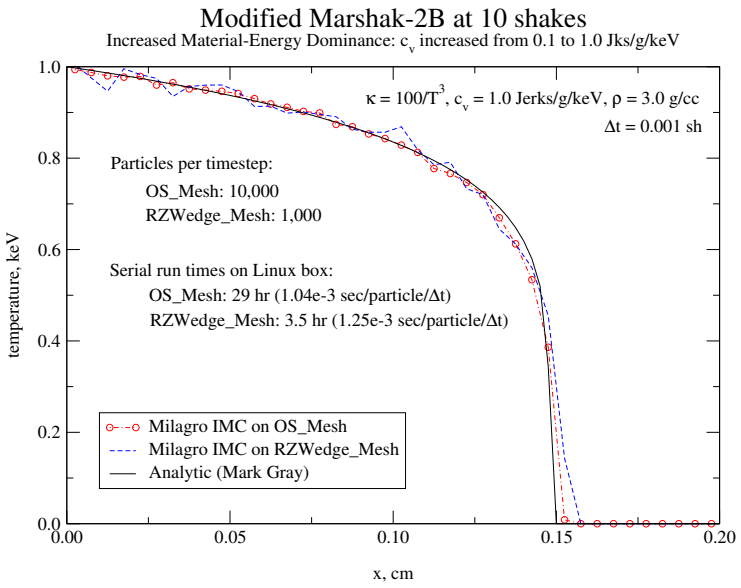



FIGURE 4.6. Marshak-2B' problem modified with an increased specific heat.

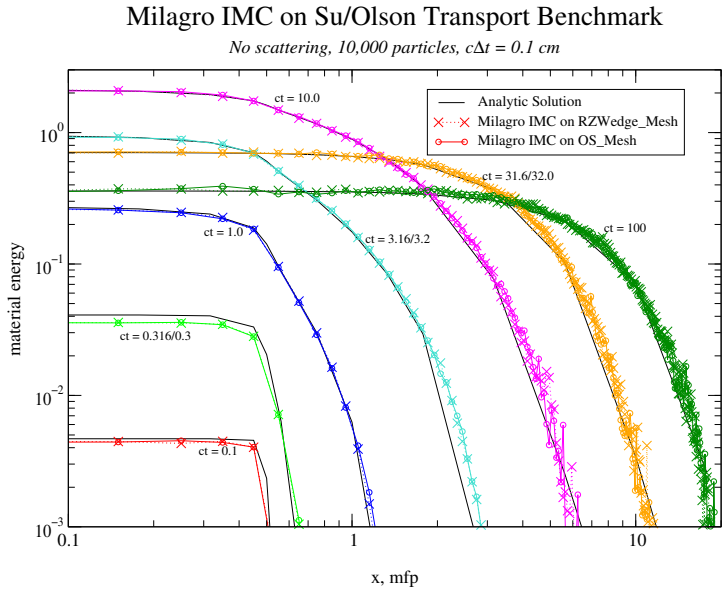



FIGURE 4.7. `Milagro` material energies for the purely absorbing Su/Olson nonequilibrium benchmark.

to about $1/3 \times 10^{-4}$ sh. The initial number of particles was 1000, and it ramped up at a rate of $10^5$ particles/sh with a ceiling of 30,000. The material had a density of 0.38214 g/cm$^3$, an absorption/emission coefficient of $2.61684/T^3$, a specific heat of 0.14361 jks/g/cm$^3$, and an initial temperature of 0.056234 keV. Figure 4.11 shows the material and radiation temperatures from `Milagro` and the VEF method for times of 1, 3, 10, 30, and 100$ct$ cm. The agreement is good. Differences in the wavefronts are probably due to the fact that `Milagro` uses time-explicit opacities.

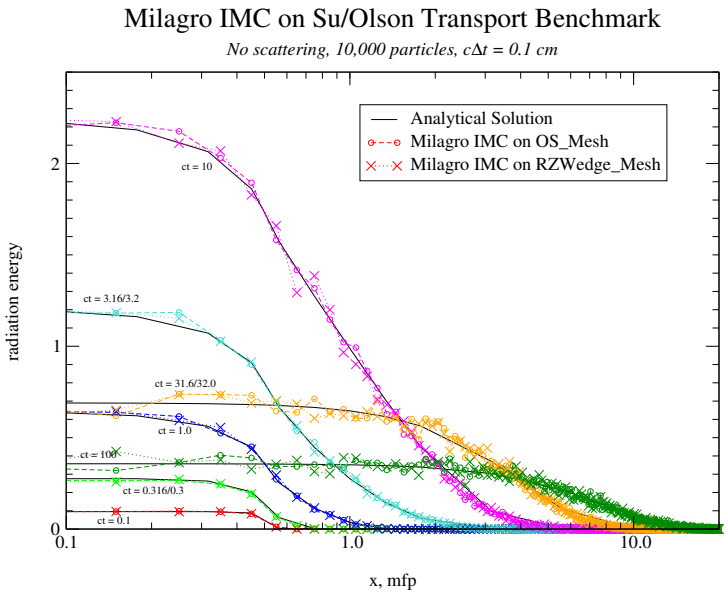



FIGURE 4.8. `Milagro` radiation energies for the purely absorbing Su/Olson nonequilibrium benchmark.

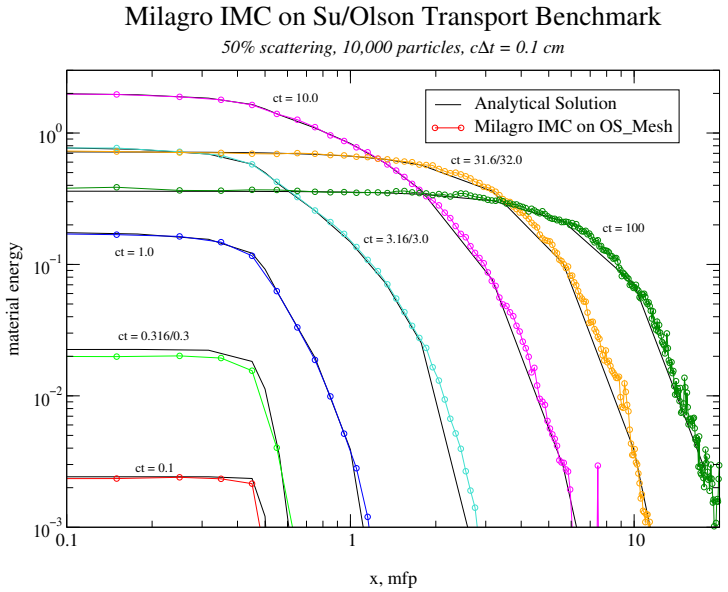



FIGURE 4.9. `Milagro` material energies for the Su/Olson nonequilibrium benchmark with 50% scattering.

In the spirit of furthering code-to-code comparison efforts, we presented results for a 2-D Cartesian dogleg problem in the `Milagro-1_1_0` release [11]. We proposed the dogleg problem because it has some geometrical complexities but does not require excessive amounts of computational effort. The OS mesh is shown in Figure 4.12. Radiation enters the pipe at $x = 0$. The pipe runs from $x = 0$ to $x = 2.5$ cm with a radius of $y = 0.44311346$ cm, then it opens up in the radial direction. Radiation flows down the pipe,

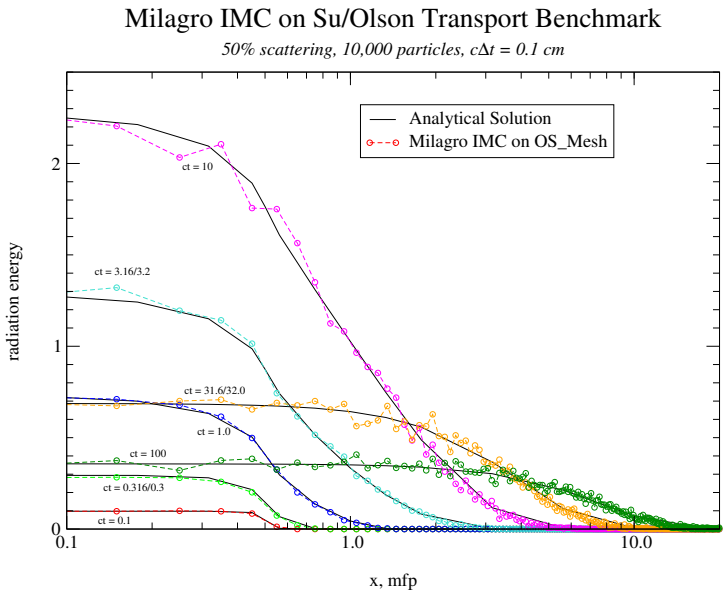



FIGURE 4.10. `Milagro` radiation energies for the Su/Olson nonequilibrium benchmark with 50% scattering.

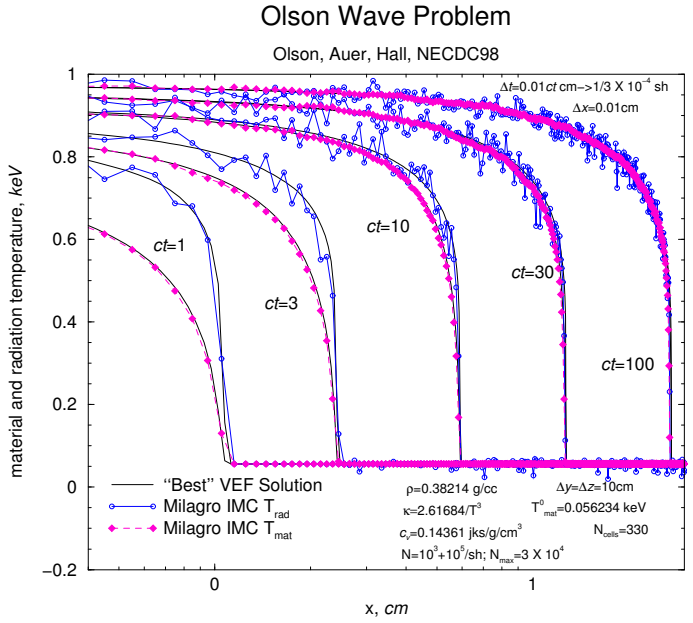



FIGURE 4.11. Radiation and material temperatures from MILAGRO and a VEF method for the Olson Wave.

hits a blocking wall at $x = 3$, and propagates radially outward. The penetration of radiation into the pipe walls requires fine zoning. Boundary conditions are reflecting on the lower $z$ face and both $y$ faces. Both $x$ faces and the high $z$ faces have vacuum boundaries. The surface source of 500 eV resides on the $x = 0$ line from $y = 0$ to $y = 0.44311346$. There are three edit cells where we monitor the radiation and material

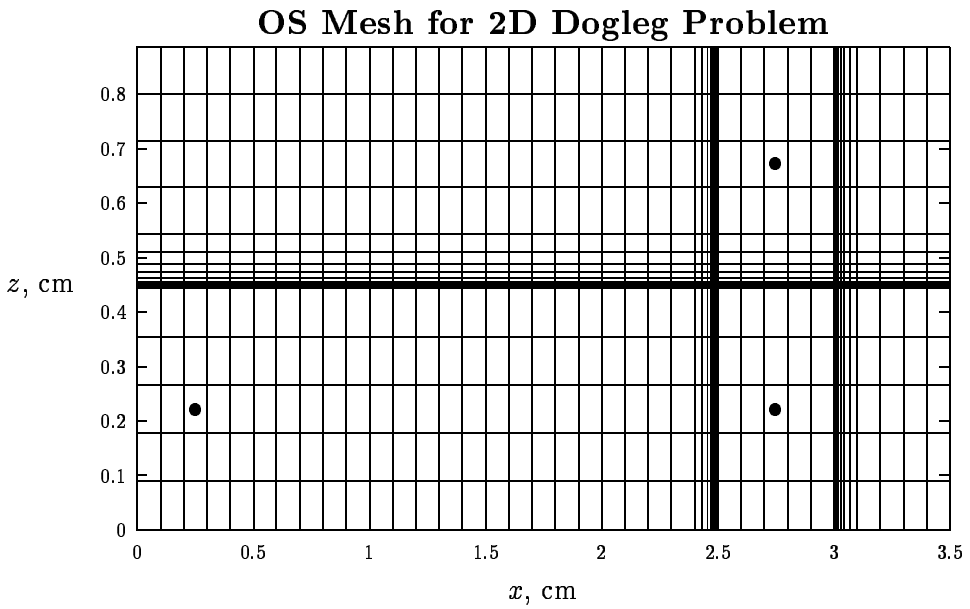



FIGURE 4.12. OS mesh for the 2-D dogleg test problem.

temperatures; they are indicated in Figure 4.12 by the large dots and ordered left-to-right and in ascending order.

We ran this problem in *X*-*Y*-*Z* geometry with the *y* dimension made up of one thick cell. Any thickness would work, but a thicker cell reduces the number of reflections. We used a timestep of 0.001 sh up to 0.01 shakes and a timestep of 0.01 thereafter. We used 10,000 particles, which was a sufficient number. Running an independent calculation out to 0.1 shakes, we saw that the statistical variation in the temperatures was much smaller than variations due to size of the timestep (as seen at 0.01 shakes where there is a data point from two calculations that had differently sized timesteps). The results are shown in Figure 4.13.

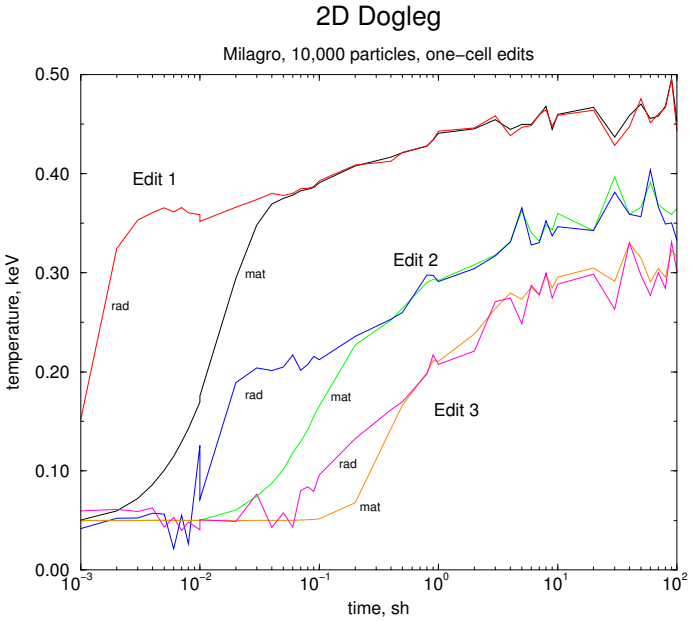



FIGURE 4.13. Temperatures for three locations in the 2-D dogleg problem.

CHAPTER 5

# Conclusions and Future Work

We have presented the new release of `Milagro`-2, an object-oriented, C++ code that performs radiative transfer using Fleck and Cummings' IMC method. `Milagro` is a part of the `Jayenne` program and is used as a stand-alone driver code to verify its underlying classes. `Milagro`-2 has been redesigned to allow for better parallelism and greater extensibility. Some of the new features in `Milagro`-2 include momentum deposition, a restart capability, a graphics capability, exact energy conservation, and improved load balancing. We have successfully demonstrated templating on mesh type. `Milagro` is templated on both an OS, Cartesian mesh type and a 3-D wedge mesh that models RZ geometry. We have presented `Milagro` results using both mesh types. A tetrahedral mesh has also been written and tested, but it has not yet been incorporated into `Milagro`. Finally, this document includes a users' guide that describes how to build and run `Milagro`.

Many items remain to be researched and implemented into `Milagro` and its underlying classes. For example, when a particle's energy-weight drops below 1% of its original energy-weight, it is killed, and all of its remaining energy-weight is deposited into the material. In some problems, the hardwired 1% undesirably affects the coupling between the radiation and material [12]. We plan to investigate some options for replacing this hardwired cutoff including:

- automatically determining a cutoff using physics parameters;
- replacing the current "full-clip Russian roulette" with a regular Russian roulette; or
- at the cutoff, switching to analog tracking instead of tracking with implicit absorption.

The last option, which was suggested by Tom Booth of LANL, seems the most feasible. The first option—determining a variable cutoff—would probably utilize the opacity and the Fleck factor and would be at least cell-dependent. It is not clear that regular Russian roulette would be feasible because its binary operation would be incompatible with the three places for low energy-weight to go: radiation, material, and ether. This could have even more adverse effects on the coupling than the "full-clip" Russian roulette.

Another issue that we may look into is to refrain from sourcing emission particles from a material that is below a floor temperature.

We may look into Forrest Brown's [of LANL's Code Development Group (X-3)] preferred stratified sampling instead of combing.

The currently used random number generator from the SPRNG library has a very large memory load per census particle. We may investigate other types of generators in the SPRNG library.

We plan to couple the IMC with EqDDMC [34] to speed up the IMC in diffusive regions. Coupling transport and diffusion will also require us to sample angles from an intensity that is linear in angle when particles escape diffusion regions.

Other topics such as hybrid methods, enhanced sources, and new methods remain to be investigated. Recent research has shown that the Carter and Forest method [35] might be a suitable replacement of Fleck and Cummings' IMC method [36].

CHAPTER 6

# 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. Here, we describe how to build and run Milagro and the input it requires.

## 6.1. Building Milagro

Building Milagro requires checking out both Milagro and Draco from our cvs repositories. In a working directory, execute the following command:

```
cvs co milagro
```

You may check out all of Draco in a similar fashion, but it is more efficient to check out only the parts of Draco that Milagro needs. We provide a script that checks out only the pertinent parts of Draco. To run it, go down to the Milagro source directory, and execute the script as follows:

```
./get_draco
```

Return to your working directory,

```
cd ..
```

where you will have both the Draco and Milagro source subdirectories. Make a target, or build, directory whose name is descriptive of the type of build. For example, to make a parallel executable, one might call the build directory "parallel":

```
mkdir parallel
```

Enter the build directory "parallel",

```
cd parallel
```

and make two build subdirectories, "milagro" and "draco":

```
mkdir draco milagro
```

Now, both Draco and Milagro need to be configured and compiled. Go into the Draco build directory,

```
cd draco
```

and configure Draco with the following command:

```
../../draco/configure --prefix=/home/tmonster/working/parallel
  --with-c4=mpi --with-dbc=0 --with-opt=1
  --with-sprng-lib=/users/tmonster/lib/sprng0.5/sgi64
  --with-sprng-inc=/users/tmonster/lib/sprng0.5/SRC
```

where the prefix specifies the build (or target) location, with-c4 specifies if parallelism is on and what type it is, with-opt specifies the level of optimization, and with-dbc specifies the level of Design-By-Contract with a decimal equivalent of a three-digit binary number: 0 means no checking is performed except for "Insist" which is always on; default is 7 which means everything ("Require", "Ensure", and "Check") is on. Run configure with "--help" to get a list of available options.

The location of the SPRNG (random number generator) libraries can be set with the environment variables SPRNG_INC_DIR and SPRNG_LIB_DIR, in which case the configure arguments would simply be --with-sprng-inc and --with-sprng-inc.

Once configured, Draco can be compiled:

```
gmake
```

Now, repeat the procedure for Milagro. Go up one directory and back down to the Milagro build directory

```
cd ../milagro
```

and configure Milagro with the following command:

```
../../milagro/configure --prefix=/home/tmonster/working/parallel
  --with-c4=mpi --with-dbc=0 --with-opt=1
  --with-sprng-lib=/users/tmonster/lib/sprng0.5/sgi64
```

Then compile `Milagro` with

```
gmake
```

The component tests in a directory may be run with

```
gmake check
```

The `Milagro` regression tests are run by executing "gmake check" in target directory

```
WORKING_DIR/TARGET/milagro/src/milagro_xyz/test
```

where, in the examples here, `WORKING_DIR` is `/home/tmonster/working` and `TARGET` is `parallel`.

The `Milagro` executable is stored in

```
WORKING_DIR/TARGET/bin/milagro_xyz
```

The version of `Milagro` that is templated on RZWedge_Mesh is configured, built, and executed the same way as `Milagro` on the Orthogonal Structured Mesh, except that the name "`milagro_xyz`" is replaced with "`milagro_rzwedge`"

### 6.2. Command Line Input

The `Milagro` executable that runs on a 3-D, Cartesian mesh is called `milagro_xyz`. The executable that runs on the RZWedge Mesh is called `milagro_rz`. For the purpose of this discussion, the two executable names can be interchanged. Usage is as follows:

`milagro_xyz --input` *filename* `--verbose --core --restart` *filename* `--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. These filenames must be less than 20 characters long. 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.

### 6.3. 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. The mesh file, if it exists separately from the input file, must contain the following blocks, where the title block specifies the coordinate system,

```
coord: ["xy", "xyz", or "rz" (all lower or all upper case)]
end-title
··· initialization block ···
end-init
··· mesh block ···
end-mesh
··· abbreviated source block: surface source position information ···
end-source
```

where we note that the mesh file must have an abbreviated source block that contains the mesh-specific, surface-source position information: "num_ss", "sur_source", and, optionally, "num_defined_surcells" and "defined_surcells".

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:

title: *mytitle* [default "Milagro"] ← *must be < 20 chars*
mesh_file: *mymeshfile* ← *must be < 20 chars*
end-title
··· material block ···
end-mat
··· 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:

title: *mytitle* [default "Milagro"] ← *must be < 20 chars*
coord: ["xy", "xyz", or "rz" (all lower or upper case)]
end-title
··· initialization block ···
end-init
··· mesh block ···
end-mesh
··· material block ···
end-mat
··· source block ···
end-source

**6.3.1. 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 an "$r$" 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 has a vacuum boundary condition 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.

**6.3.2. Mesh Block.** 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.

**6.3.3. 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 specify the density [g/cm$^3$], the absorption/emission opacity, the isotropic scattering opacity [cm$^2$/g], the initial temperature [keV], and the specific heat.

The opacities may be calculated according to a few simple analytic opacity models that are specified by the keyword "analytic_opacity:". The four possible values of "analytic_opacity:" are as follows:

- "straight", in which case the opacity on the "mat:" line is entered in units of [cm$^2$/g];
- "tcube", in which case the opacity is proportional to the inverse cube of the material temperature and is entered as a coefficient in units of [cm$^2$-keV$^3$/g];

- "tlinear", in which case the opacity is proportional to the inverse of the material temperature and is entered as a coefficient in units of [cm$^2$-keV/g]; and
- "opacity", in which case the opacity is entered in units of a cross section [cm$^{-1}$]. Every material must follow the same analytic opacity model.

An additive component of the opacity is also available with the keyword "analytic_opacity_offsets:" with an entry following for each material. The units of the offsets are consistent with the analytic opacity model, which is to say that the offsets are to have units of [cm$^2$/g] for "analytic_opacity:" equal to "straight", "tcube", and "tlinear" and units of [cm$^{-1}$] for "analytic_opacity:" equal to "opacity". For example, the offsets allow specification of such coefficients as $\kappa = \kappa_0 + \kappa_1/T$.

The specific heats may also be calculated according to a few simple analytic models. The models are specified with the keyword "analytic_sp_heat:". The three possible models for "analytic_sp_heat:" are as follows:

- "straight", in which case the specific heat on the "mat:" line is specified in units of [jerks/(g keV)];
- "tcube" in which case the specific heat is directly proportional to the cube of the material temperature and is entered as a coefficient in units of [jks/(cm$^3$ kev$^4$)]; and
- "dedt" in which case the specific heat is entered as a heat capacity in units of [jerks/keV].

Every material must follow the same analytic specific heat model.

The material block also contains the Fleck implicitness factor, $\alpha$, 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.

In the example material block that follows, the first material (in the low $y$ plane) has a density of 3 g/cm$^3$, opacity of 1.0 $T^{-3}$ [cm$^{-1}$], scattering opacity of zero, initial temperature of 0.2 keV, and a specific heat of 0.05 jerks/g/keV. The second material (in the high $y$ plane) has a density of 1.5 g/cm$^3$, opacity of 10.0 $T^{-3}$ [cm$^{-1}$], scattering opacity of 5 cm$^{-1}$, initial temperature of 0.3 keV, and a specific heat of 0.1 jerks/(g keV).

```
num_zones: 8                                  ← always required
zonemap: 1 1 2 2 1 1 2 2          ← mat 1 in low y; mat 2 in high y
num_materials: 2
mat: 1    3.0 1.0 0.0 0.2 0.05
       2    1.5 10.0 5.0 0.3 0.1
analytic_opacity: tcube
analytic_sp_heat: straight
implicitness: 1.0
end-mat
```

**6.3.4. Source Block.** The source block contains physical source input, runtime parameters, and edit specifications.

Available source options are an external material volume source, an external radiation source, and a blackbody 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 blackbody 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.

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 runtime parameters are specified in the source block with the following keywords:

- "delta_t:" the timestep, $\Delta t$, in shakes,
- "max_cycle:" the number of cycles, or timesteps, to run,
- number of particles to run this timestep = $\min(N_p^{\mathtt{max}}, N_p^{\mathtt{nom}} + t_{\mathtt{prob}} \frac{dN_p}{dt})$
  - "npnom:" the nominal number of particles, $N_p^{\mathtt{nom}}$,
  - "npmax:" the maximum number of particles, $N_p^{\mathtt{max}}$,
  - "dnpdt:" the rate of change of particles per shake, $\frac{dN_p}{dt}$;
- "capacity:" the number of cells per processor,
- "buffer_size:" the size, in particles, of the buffer for communication and census dumping, and
- "seed:" the random number seed, a positive integer.

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. An example source block follows.

```
timestep: 0.001       ← in shakes, constant
max_cycle: 100        ← number of cycles to run
npnom: 100            ← nominal number of particles per cycle (or timestep)
npmax: 1000           ← maximum number of particles per cycle (or timestep)
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
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.
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.
capacity: 66          ← each processor contains all the cells
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
```

## 6.4. 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:", and "restart_cycle:". Again, the title and filenames 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
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 timestep.
- max_cycle: Maximum problem cycle.
- 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: Timestep.
- 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.

## 6.5. Troubleshooting

**Building:**

When building `Draco` and `Milagro` on machine theta, the MPI libraries sometimes cannot be found even though the modules are loaded. If that happens, you need to explicitly specify the appropriate directory in the configure arguments:

```
--with-mpi-lib=/opt/mpt/mpt_1.3.0.3/usr/lib64
```

**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?

# 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. M. Evans and T. J. Urbatsch, "The Wedgehog Implicit Monte Carlo package." In development, 2001.

[3] 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.

[4] 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.

[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 appoximate 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] G. C. Pomraning, *The Equations of Radiation Hydrodynamics*. Oxford: Pergamon Press, 1973.

[17] J. A. Fleck, Jr., "The calculation of nonlinear radiation transport by a Monte Carlo method," in *Computational Methods in the Physical Sciences* (B. Alder and S. Fernbach, eds.), vol. 1, p. 43, New York: McGraw-Hill, 1963.

[18] E. W. Larsen and B. Mercier, "Analysis of a Monte Carlo method for nonlinear radiative transfer," *Journal of Computational Physics*, vol. 71, pp. 50–64, 1987.

[19] T. J. Urbatsch and T. M. Evans, "Reproducibility in parallel Monte Carlo codes," Technical Memo. XTM–99–022 (U), Los Alamos National Laboratory, apr 12 1999. LA–UR–99-1826.

[20] L. L. Carter and E. D. Cashwell, *Particle-Transport Simulation with the Monte Carlo Method*. ERDA critical review series, Technical Information Center, Office of Public Affairs, U.S. Energy Research and Development Administration, 1975. TID-26607.

[21] J. Lakos, *Large-Scale C++ Software Design*. Reading, MA: Addison-Wesley, Inc., 1996.

[22] T. J. Urbatsch and T. M. Evans, "Analytic temperature updates in milagro for $T^3$ specific heats," Technical Memo CCS-4:01-12(U), Los Alamos National Laboratory, March 12 2001. LA-UR-01-1427.

[23] E. Canfield, "private communication." Aug. 1998.

[24] T. Urbatsch and T. Evans, "Momentum deposition in IMC codes," Research Note X-6–RN(U)-00–12, Los Alamos National Laboratory, May 2000. LA-UR-00–2183.

[25] D. Mihalas and B. W. Mihalas, *Foundations of Radiation Hydrodynamics*. New York: Oxford University Press, 1984.

[26] R. M. Roberts, "3-T diffusion with material motion corrections," web publication, Los Alamos National Laboratory, August 5 1999. LA-UR-99-4438.

[27] T. J. Urbatsch and T. M. Evans, "Strategy for parallel Implicit Monte Carlo," Research Note XTM-RN(U)-98-018, Los Alamos National Laboratory, May 1998. LA-UR–98–2263.

[28] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA; London, England: The MIT Press, 1996. 3rd printing.

[29] D. R. Butenhof, *Programming with POSIX Threads*. Reading, MA: Addison-Wesley, Inc., 1997. ISBN 0-201-63392-2.

[30] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers of Harcourt, Inc., October 2000. ISBN 1-55860-671-8.

[31] B. Meyer, *Object-Oriented Sofware Construction*. Upper Saddle River, NJ: Prentice Hall, second ed., 1997.

[32] T. J. Urbatsch and T. M. Evans, "Software testing in Milagro, Draco, and the Jayenne project," Research Note XTM–RN(U)–99–018, Los Alamos National Laboratory, June 28 1999. LA–UR–99–3482.

[33] T. M. Evans, T. J. Urbatsch, and H. Lichtenstein, "1-D equilibrium discrete diffusion Monte Carlo," Research Note X-6:RN(U)00-10, Los Alamos National Laboratory, April 25 2000. (LA-UR-00-1996).

[34] T. M. Evans, T. J. Urbatsch, and H. Lichtenstein, "1-D equilibrium discrete diffusion Monte Carlo," presented (by H. G. Hughes) at the MC2000 - International Conference, (23-26 October, 2000) Lisbon, Portugal, Los Alamos National Laboratory, July 2000. LA-UR-00-3371.

[35] L. L. Carter and C. A. Forest, "Nonlinear radiation transport simulation with an implicit Monte Carlo method," Tech. Rep. LA-5038, Los Alamos National Laboratory, January 1973.

[36] W. R. Martin and F. B. Brown, "Comparison of Monte Carlo methods for nonlinear radiative transport," in *Poceedings of the American Nuclear Society Topical Meeting: International Conference on Mathematical Methods to Nuclear Applications*, American Nuclear Society, September 9-13 2001. Salt Lake City, Utah.


Los Alamos
NATIONAL LABORATORY
EST.1943