LA-UR-00-6118
Approved for public release;
distribution is unlimited.

Title: **MILAGRO Implicit Monte Carlo: New Capabilities and Results**

Author(s): Todd J. Urbatsch and Thomas M. Evans

Submitted to:

http://lib-www.lanl.gov/la-pubs/00357106.pdf

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

FORM 836 (10/96)

MILAGRO Implicit Monte Carlo: New Capabilities and Results (U)

Todd J. Urbatsch and Thomas M. Evans
CCS–4, MS D409,
Los Alamos National Laboratory, Los Alamos, NM 87545

Milagro is a stand-alone, radiation-only, code that performs nonlinear radiative transfer calculations using the Fleck and Cummings method of Implicit Monte Carlo (IMC). Milagro is an object-oriented, C++ code that utilizes classes in our group's (CCS–4) radiation transport library. Milagro and its underlying classes have been significantly upgraded since 1998, when results from Milagro were first presented. Most notably, the object-oriented design has been revised to allow for optimal stand-alone parallel efficiency and rapid integration of new classes. For example, the better design, coupled with stringent component testing, allowed for immediate integration of the full domain decomposition parallel scheme. (It is a simple philosophy: spend time on the design, and debug early and once.) Milagro's classes are templated on mesh type. Currently, it runs on an orthogonal, structured, not–necessarily–uniform, Cartesian mesh of up to three dimensions, an RZ-Wedge mesh, and soon a tetrahedral mesh. Milagro considers one-frequency, or "grey," radiation with isotropic scattering, user-defined analytic opacities and equation-of-state, and various source types: surface, material, and radiation. Tallies produced by Milagro include energy and momentum deposition. In parallel, Milagro can run on a mesh that is fully replicated on all processors or on a mesh that is fully decomposed in the spatial domain. Milagro is reproducible, regardless of number of processors or parallel topology, and it now exactly conserves energy both globally and locally. Milagro has the capability for EnSight graphics and restarting. Finally, Milagro has been well verified with its use of Design-by-Contract™, component tests, and regression tests, and with its agreement to results of analytic test problems. By successfully running analytic and benchmark problems, Milagro serves to integrally verify all of its underlying classes, thus paving the way for other service packages based on these classes (U).

Keywords: Implicit Monte Carlo, radiation transfer, parallel computing, non-linear transport, object-oriented scientific computing

Introduction

Milagro, an Implicit Monte Carlo (IMC) code written in C++, was first presented in 1998 [1]. Milagro is the stand–alone driver for our IMC classes, which are used in all of our IMC packages. Thus, Milagro provides an important role as both a methods development platform and a stand-alone verification/validation tool. Milagro has undergone massive improvements, which we will highlight. Most notably, it has been redesigned for better C++ class interaction and better parallel performance. Milagro and the classes it uses are templated on mesh-type. In 1998, we only had one mesh-type, and therefore, we had not proven that the concept of templating on mesh-type worked. We have extended Milagro's

repertoire to include an RZ–Wedge Mesh, thus demonstrating the usefulness of templating on mesh type. Milagro has been verified to a much larger extent: we have significantly increased our use of Design–by–Contract™[1] (DBC) [2] and we have increased the number and coverage of our component tests. We will describe these and other advancements.

Software Development Philosophy

Our goal is to provide well–verified IMC software packages to users. Verification is the assurance that the software is actually solving the equations we intend the software to solve. In practice, verification efforts rely heavily on good software management. Verification is not directly dependent on software quality, but poor software quality can render the task of verification untenable or impossible. Unverified software may or may not produce incorrect results; the developer and user simply do not know. We have found that our software practices have somewhat higher up front costs, but they have the benefit of making verification easier and reducing maintenance costs down the road.

The first part of software quality is the software design. We use an object–oriented design [2] to allow for modularity, code reuse, and extensibility. Levelized design—where each component depends only on lower level components—makes component testing possible. Milagro's levelized design is shown in Fig. 1.

We make our jobs easier by using such practices as version control (cvs [3]), reproducibility, and regression testing. Version control stabilizes the source code and serves as a safety net. Reproducibility allows us to rapidly verify new IMC packages that use the same underlying classes. Regression testing lends some measure of software stability. In addition to stability, we have designed our regression tests to perform low–order verification and component testing as well. We have different types of regression tests: some test specific, specialized features from the high, code–compiled level and some test the integrated features of multiple components. The regression tests are executed automatically each night.

In the way of actual verification, we begin with Design–by–Contract [2] at the class and line–of–code level. Design–by–Contract is a multi–level set of compiler–directed assertions that check the validity of data at the beginning, during, and at the end of a class. Next, we externally test all components. Component tests ensure that each component does what is expected. With levelized design and component tests, higher level components may rely assuredly on lower level components. The final software product may be viewed as the ultimate component, which is tested by rigorous comparison to analytical results.

When we need to interface an already–verified IMC package to a new customer, we simulate the customer in order to verify the interface. We call this simulated customer a "shunt." By using shunts, we come closer to fully debugging our package and its interface before we actually interface to the customer. Thus, assuming the customer has verified the coding on their side of the interface and their share of the interface, the actual interfacing should be fairly painless.

We have also enjoyed some success with pair programming (two people, one keyboard). Bugs are caught faster, and different perspectives produce better code on the first attempt. Initial quality is important because, once code is written, it takes a very deliberate effort to revisit a piece of code. We have found that pair programming produces better results than

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

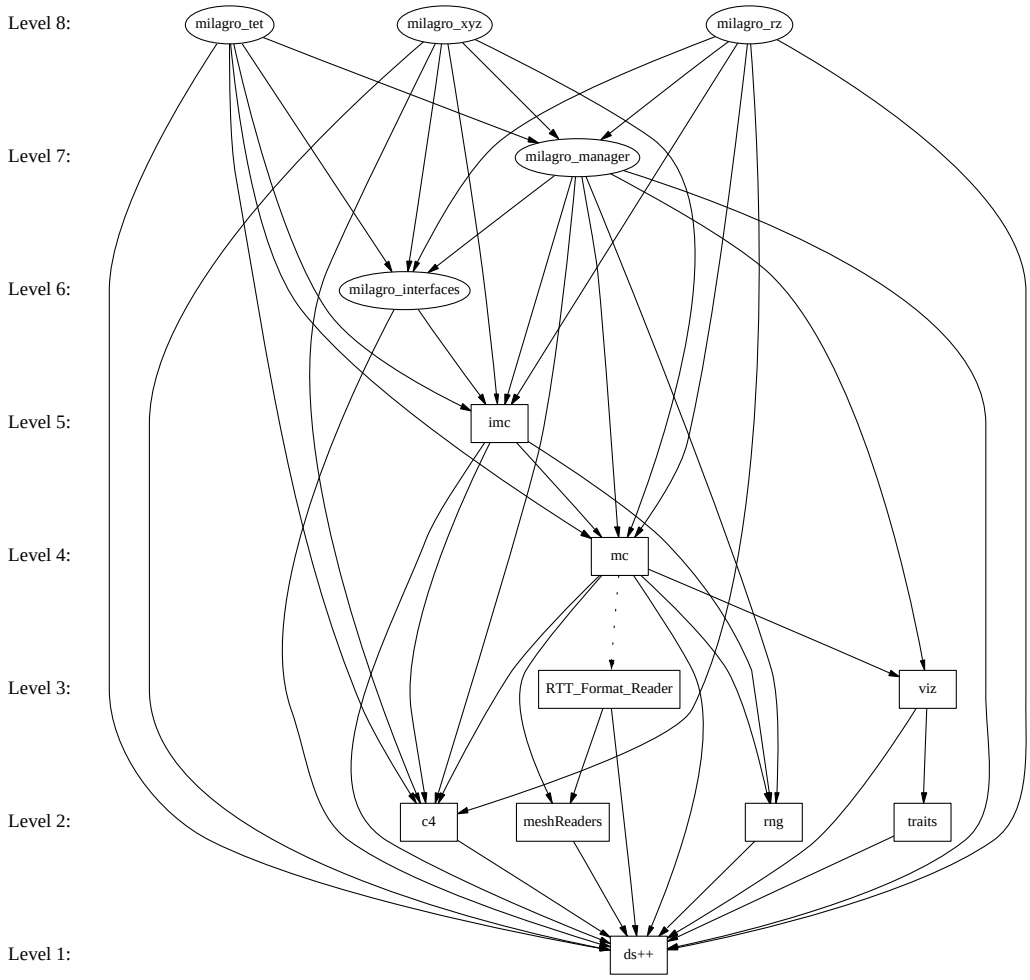


Figure 1: Levelized design for Milagro IMC.

the seemingly viable approach of having one person write a class and another person write the component test for the class.

We have found that customer acceptance of staged deliverables is a crucial component of delivering a package to a customer. From a code developer's point of view, it makes much practical sense to deliver a product to the customer in stages, beginning with the simplest and increasing in complexity. This view goes along with the philosophy of finding bugs at the earliest, least complicated time. Unfortunately, the earlier deliverables may be unacceptable, or viewed as unacceptable, by the customer, and the valuable developer/customer relationship is nonexistent. As always, improved communication is the best solution. We have also found it practical to sensibly build up a user–base by first targeting customers who need or are willing to accept a package with limited capability.

Improved Object-Oriented Design

In the time since Milagro was first reported [1], it has undergone redesign. The motivation for redesigning, or refactoring, Milagro was twofold: 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 we discovered 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 simply was not efficient 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 2 shows a schematic of the old and new designs.

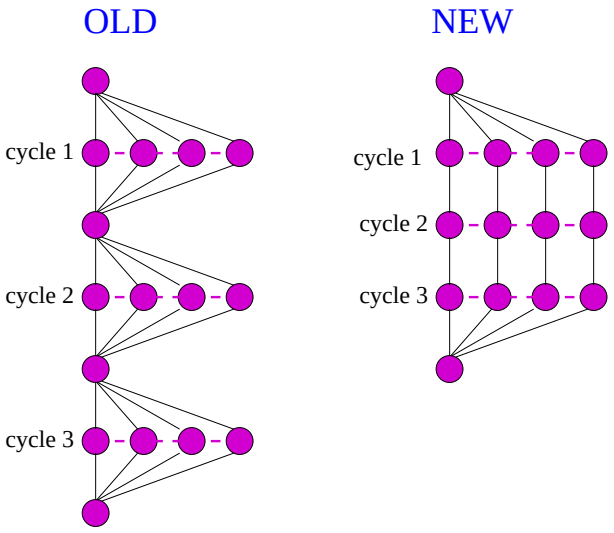
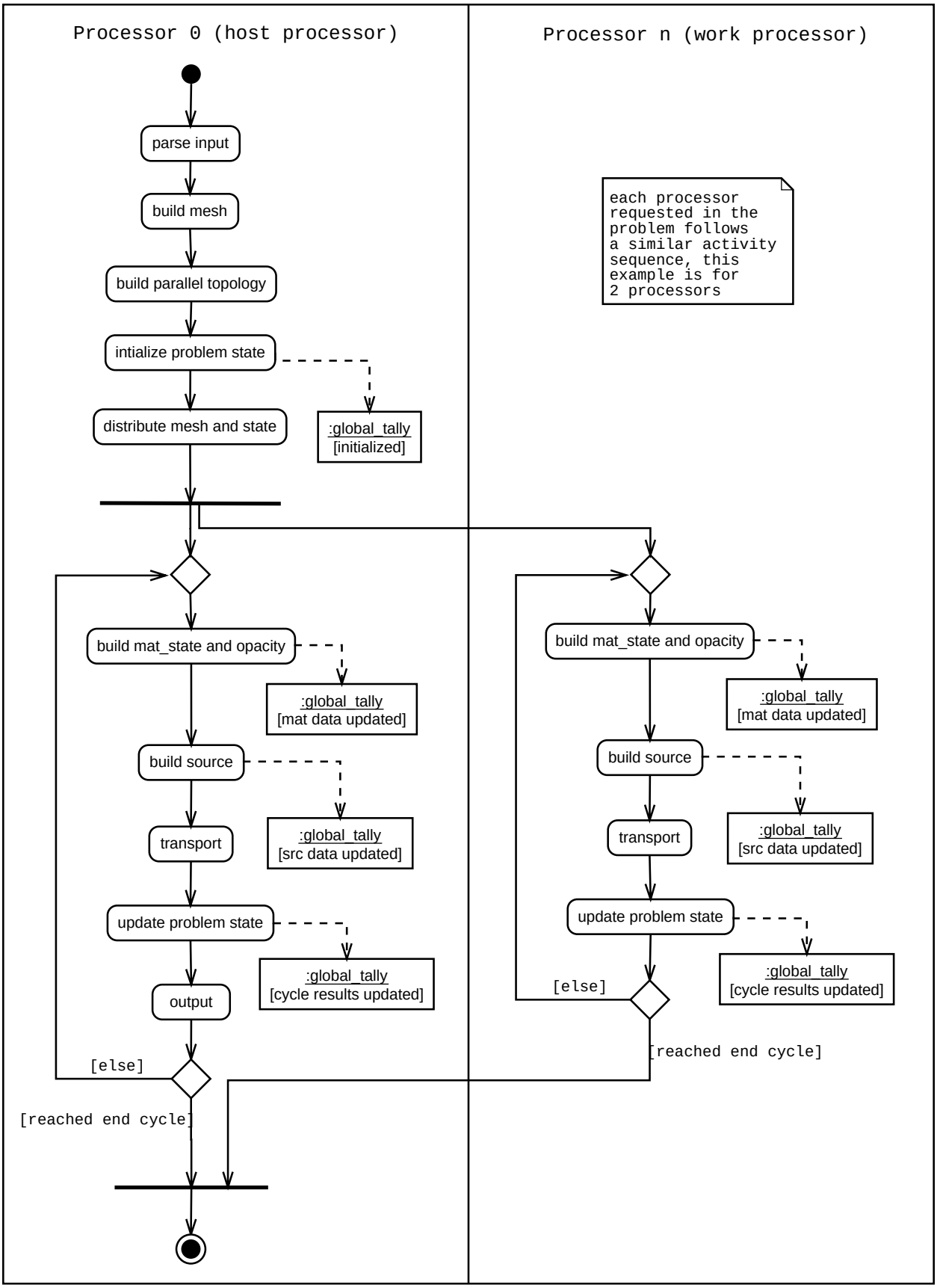


Figure 2: Schematic of Milagro's new parallel design.

The activity chart for Milagro, Fig. 3, shows when classes are instantiated in Milagro's new design.

Figure 3: Activity chart for Milagro IMC.



Mesh Types

Milagro and the classes it uses are templated on mesh type. By templating on mesh type, Milagro can run on any mesh as long as the mesh type provides the required services. The same goes for any other IMC package that uses the same classes that Milagro uses. For a long time, Milagro only ran on an orthogonal, structured Cartesian mesh. Now, however, we have run Milagro on another mesh type, the RZ–Wedge Mesh. Thus, we have demonstrated that templating on mesh type actually works.

Orthogonal, Structured Cartesian Mesh The Orthogonal, Structured Mesh was the first mesh on which Milagro performed IMC calculations. Although 3-D is what we mostly use, 2-D and 1-D are available for component and regression testing. New for Milagro is the capability for ratio zoning. The orthogonal, structured mesh type could always handle any size cells, but the input parser needed upgrading to input the information and calculate the cell parameters. Also, an Adaptive Mesh Refinement (AMR) capability has been implemented, but it has not been tested.

RZ–Wedge Mesh The RZ–Wedge Mesh is the second mesh–type on which Milagro has been templated. Traditionally, 2-D IMC calculations have been performed in the actual R-Z geometry, where particle paths are hyperbolic. We find that representing the RZ mesh as a three-dimensional, XYZ wedge is more attractive. The distance–to–boundary calculation are simpler for a wedge than for true RZ geometry. The radial momentum in true RZ geometry requires extra sampling because the radial direction continuously changes over a particle's path. With the wedge, momentum is tallied in three dimensions and then converted to RZ geometry at the end of a timestep. The disadvantage of the RZ–Wedge is that, although volume is conserved, there is a geometric error because curvature is not exactly modeled. Fortunately, as the user–defined wedge angle decreases, the error diminishes. Unfortunately, as the wedge angle decreases, the computing time increases because of the increased frequency of reflection. Thus the user must evaluate the tradeoff in order to intelligently select a wedge angle.

The RZ–Wedge Mesh also allows for AMR, as shown in Figure 4. The RZ–Wedge Mesh has been verified through the usual Design–by–Contract™ (DBC) [2], component testing, and verification problems.

Tetrahedral Mesh A tetrahedral mesh–type has been written by Grady Hughes of CCS–4, Los Alamos National Laboratory. It provides all the required services (distance–to–boundary, face normals, etc.), but it has not been incorporated into Milagro yet. One of the services is graphics dumps, as seen in Figure 5. The tetrahedral mesh has been verified to provide its services correctly.

Parallelization Schemes

Milagro and the classes upon which it is based run on two different types of parallel topologies: replication and domain decomposition. Replication is where the entire mesh is replicated on every processor. Domain decomposition, the choice of deterministic transport methods, is where the mesh is split among all available processors, i.e., no mesh cell is ever

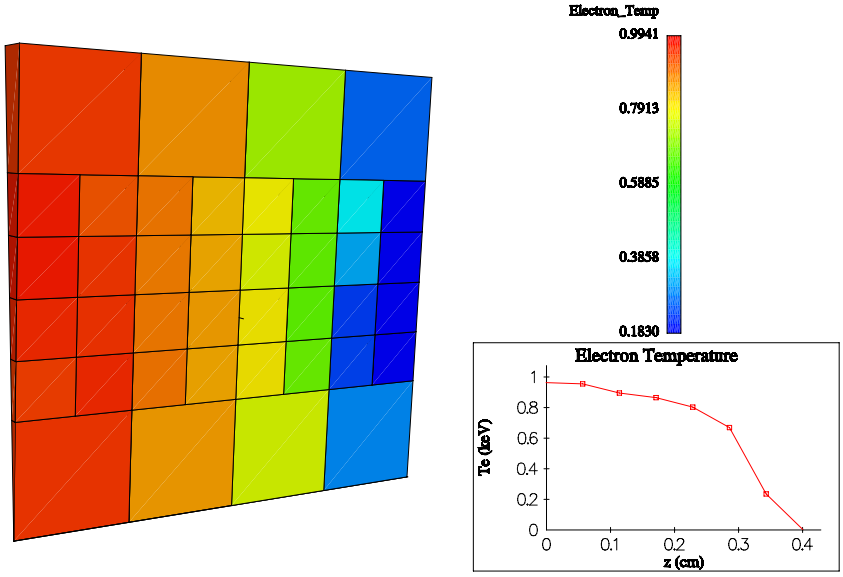


Figure 4: RZ–Wedge Mesh with AMR.

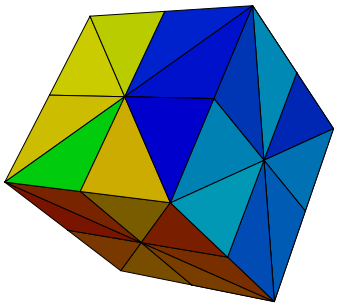


Figure 5: Tetrahedral Mesh.

replicated. Currently, all of the parallel schemes described here run in a message-passing architecture. We are currently implementing threaded schemes based on OpenMP, but these are not in a finished state.

Reproducibility eased the testing of our redesign of Milagro. In one case, we inadvertantly introduced an off–by–one error in the random number stream ID. Without reproducibility, we may have never found that bug.

A key component in each parallel scheme is the IMC source builder, which had to be rewritten in our new design. Some improvements were made along the way. In the replication case, for instance, the particles were made to be exactly load balance (while maintaining reproducibility).

Figure 6 shows the constant–work scaling for a steady–state, infinite medium. The workload is uniform across the cells in this problem. Full replication scales reasonably well. Due to the communication costs during particle transport, the domain decomposition case performs worse than the replication.

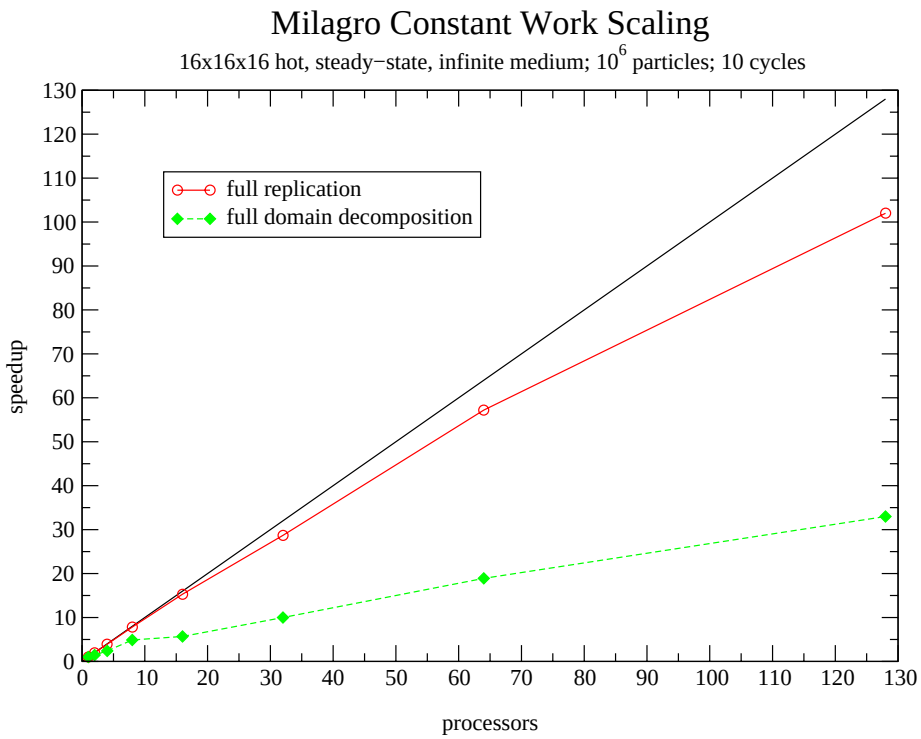


Figure 6: Constant work scaling for replicated and decomposed spatial domain.

New Features

Restart Milagro now has a restart capability, which serves the traditional role of recovering from time–limits and crashes. Restarting is also a crude way to change parallel topologies. Consider a mesh with a large number of cells where the transport work is initially limited to only a small number of cells. In this case, the deterministic workload (calculations for the source and tallies and parallel communication) would be larger than the IMC work. Instead of replicating the deterministic work, it would be more efficient to use a decomposed domain. Later in the calculation, however, the transport workload could become larger than the deterministic workload. At that point, it could be advantageous to stop the domain decomposed problem and restart it with a fully replicated mesh.

Graphics Capabilities Milagro now has the ability to make EnSight [4] graphics dumps, as can be seen in several figures in this paper.

Energy Conservation in the Census Comb The census particles are the particles that are still alive at the end of a time step. 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 due to Canfield [5]. 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 prongs hit are retained and reassigned a new energy-weight

that is the same as every other surviving census particle. Thus, the comb globally conserves energy in addition to minimizing variance and controlling particle numbers.

In Milagro, we place a large premium on reproducibility, which is to say that Milagro gives the same answer regardless of number of processors or parallelization scheme. The historical comb, in order to be reproducible, requires all the census particles to be lined up in the same order regardless of processor count 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. This was the state of Milagro before now.

Recently, 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 seems unbiased, but, even if it is, the bias should be small since the census energy in the cell is presumably small.) Second, after Russian roulette and any necessary resurrection, the weights of the surviving census particles are 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, conserves global energy, and conserves 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.

Momentum Deposition We have implemented momentum deposition from the radiation to the material in the particle and tally classes that Milagro uses. The momentum deposition tally is analog since it is tied to particle events. However, since the IMC particle absorption is implicit (not analog), the corresponding momentum deposition tally has an implicit component. Once implemented, the momentum deposition was verified to be correct on problems with analytic solutions from Mark Gray's Analytical Test Suite[2]. It was verified for a steady–state, infinite, homogeneous medium and for a few different Marshak waves. The momentum deposition for the Marshak–1D wave is shown in Figure 7.

Extended Source Capability Milagro has capabilities for a surface source, an external material energy source, and an external radiation energy source.

Originally, Milagro's surface source could only be applied to an entire system face. Now it has been upgraded to apply to a user–defined set of cells on a system face. The capability is easily extensible to a host–driven IMC package, where the host specifies the surface source cells.

The time–implicitness in Fleck and Cummings' IMC method splits an external material

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

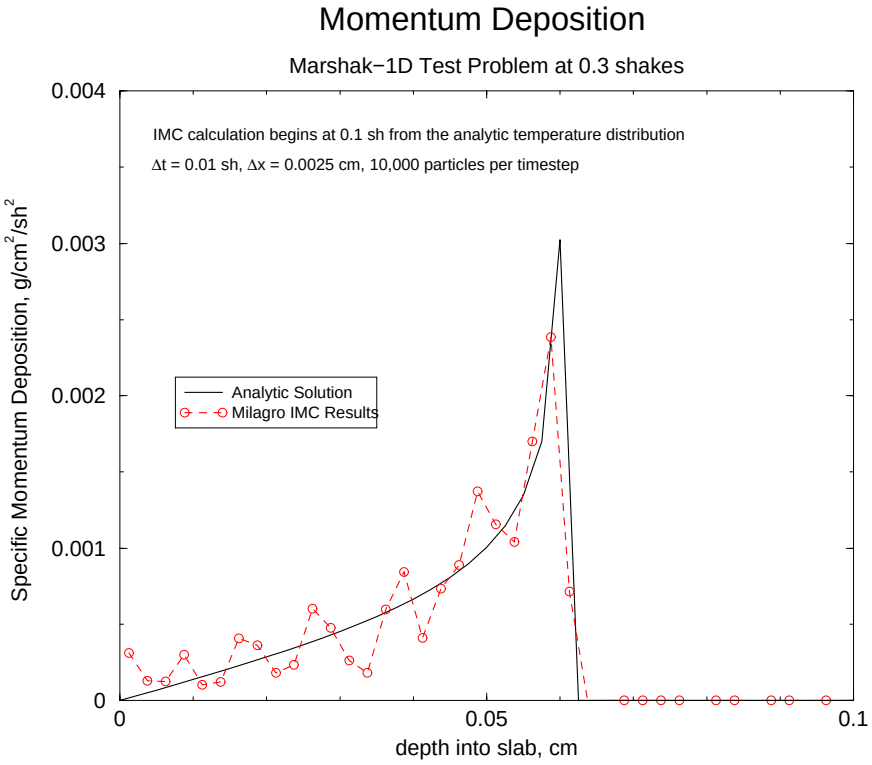


Figure 7: Milagro's momentum deposition for the Marshak–1D wave (delta function source in time and space).

source into a "time–explicit" portion in the material equation and a "time–implicit" portion in the radiation equation. We discovered that we had not included the "time–explicit" portion. After correcting the oversight, the external material source was verified to be correct by comparing the IMC results to the time dependence of the analytic solution of the diffusion equation in an infinite, homogeneous medium with a constant, external material source.

Status of Random Number Generators

We have enjoyed success with the Scalable Parallel Random Number Generators (SPRNG) library [6]. One shortcoming is that the random number stream identification and number of streams, or generators, are represented in the interface by a single (4-byte) integer. In practice, then, we are limited to slightly over two billion random particles since we associate a unique random number stream to each particle. The SPRNG vendors are working to replace the single integer representations with an array of integers. In the meantime, we have implemented a workaround where stream identification numbers over two billion are wrapped seamlessly back to zero. This workaround may produce unwanted correlations if the number of particles per timestep approaches a significant fraction of two billion.

Verification Efforts

Our verification of Milagro and its underlying classes is an ongoing process. As discussed in the section on Software Development Philosophy, verification begins at the line–of–code level and progresses from the lowest–level component to the final, compiled code. We compare Milagro results to analytic solutions and to solutions from other codes.

Analytic Problems Our basic Milagro verification problems are simple, degenerate, and terribly uninteresting. Our workhorse degenerate analytic problem is a steady–state, homogeneous, infinite medium, which we model as a cube or wedge with specularly reflecting boundaries. With these problems, we can test, in a degenerate state, more complicated capabilities, such as multiple materials (different specifications of the same material), shifting about the origin, region specification, and so on. We can test time and space equilibration in these infinite medium problems. We can also replace one of the reflecting boundaries with a surface source at the same ambient temperature. For further testing, we may move the surface source to any of the six (or three, in the case of a wedge) boundaries. Some problem variants will exactly replicate, some will be only statistically equivalent, and some will vastly differ. In one case, we took the infinite medium, applied a linearly sloped initial temperature it, and watched it equilibrate in space and time [7]. It indeed converged to the correct final temperature.

Beyond degenerately simple problems, we compare Milagro results on both the 3–D Orthogonal Structured Mesh and the RZWedge Mesh to analytic solutions:

- Marshak Waves [8, 9] (3 variants)
- Su/Olson non-equilibrium transport benchmarks [10] (2 variants)
- Olson Wave [11]

As an example, in Fig. 8, we show a plot of Milagro's radiation energy on the RZWedge Mesh for the Su/Olson benchmark with 50% scattering. For other test problems, Milagro's results are similar in their agreement to the analytic results.

Code–to–Code Comparison Code–to–code comparisons may or may not verify the code; hopefully they build *some* confidence. For instance, we ran one variant of the Marshak wave, and our results differed significantly from the analytic solution. However, results from other transport codes differed from the analytic solution, as well. After discussions with Marv Alme, X–3, LANL, we now believe that the analytic solver makes approximations that break down in this particular problem and, thus, give an incorrect solution.

Conclusion

We have redesigned Milagro, the Implicit Monte Carlo (IMC) radiative transfer code, and upgraded its parallel performance, improved its object–oriented design, and extended its capabilities. We have shown good scaling for replicated mesh topologies and, as expected, poor scaling for domain–decomposed mesh topologies. We have added an RZ capability; exact global and local energy conservation; momentum deposition, which was verified against analytic results; restart; and graphics capability. While upgrading Milagro and delivering

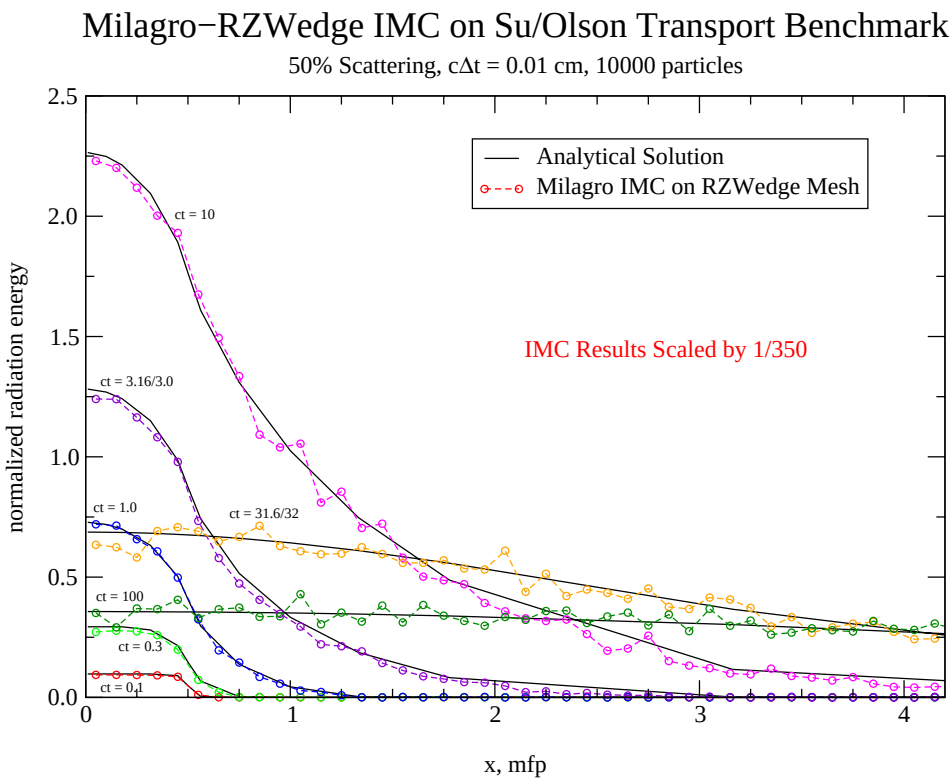


Figure 8: Radiation energy from Milagro on an RZWedge Mesh for the Su/Olson benchmark with fifty percent scattering.

IMC packages, we have continued to learn and apply more effective software quality practices, which greatly ease our verification efforts. Verification is a continuing effort that begins at the line–of–code level and transverses from the lowest–level component to the compiled code–level and beyond to package interfaces and shunts, which are simulations of the customer code.

References

[1] 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.

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

[3] P. Cederqvist, *Version Management with CVS*. Signum Support AB, 1993. v1.10.

[4] CEI, Inc., Morrisville, NC 27560, *Creator of EnSight, EnLiten & EnVideo*. www.ensight.com.

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

[6] 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.

[7] 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, lanl, July 2000. LA-UR-00-3371.

[8] 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.

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

[10] 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.

[11] 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.