LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-411198

Enhancements to the Combinatorial Geometry Particle Tracker in the Mercury Monte Carlo Transport Code: Embedded Meshes and Domain Decomposition

G. M. Greenman, M. J. O'Brien, R. J. Procassini, K. I. Joy

March 11, 2009

International Conference on Mathematics, Computational Methods & Reactor Physics
Saratoga Springs, NY, United States
May 3, 2009 through May 7, 2009

Disclaimer

International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)
Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009)

ENHANCEMENTS TO THE COMBINATORIAL GEOMETRY PARTICLE TRACKER IN THE MERCURY MONTE CARLO TRANSPORT CODE: EMBEDDED MESHES AND DOMAIN DECOMPOSITION

Gregory Greenman, Matthew O'Brien and Richard Procassini
Lawrence Livermore National Lab
Mail Stop L-95, P.O. Box 808
Livermore, CA 94551
United States of America
greenman1@llnl.gov, mobrien@llnl.gov and spike@llnl.gov

Kenneth Joy
Computer Science Department
University of California at Davis
Davis, CA 95616
United States of America
kijoy@ucdavis.edu

ABSTRACT

Two enhancements to the combinatorial geometry (CG) particle tracker in the ***Mercury*** Monte Carlo transport code are presented. The first enhancement is a hybrid particle tracker wherein a mesh region is embedded within a CG region. This method permits efficient calculations of problems with contain both large-scale heterogeneous and homogeneous regions. The second enhancement relates to the addition of parallelism within the CG tracker via spatial domain decomposition. This permits calculations of problems with a large degree of geometric complexity, which are not possible through particle parallelism alone. In this method, the cells are decomposed across processors and a particles is communicated to an adjacent processor when it tracks to an interprocessor boundary. Applications that demonstrate the efficacy of these new methods are presented.

Key Words: Monte Carlo, particle transport, combinatorial geometry, embedded mesh, domain decomposition

1 INTRODUCTION

Two enhancements to the combinatorial geometry (CG) particle tracker in the ***Mercury*** [1],[2] Monte Carlo transport code are presented. These permit the efficient calculation of problems with a high degree of geometric complexity. The first enhancement is a hybrid version of the particle tracker. In this method, a mesh region is embedded within a CG region. This hybrid scheme permits efficient calculations of problems with contain both large-scale heterogeneous and homogeneous regions. The second enhancement is a second form of parallelism within the CG tracker. While the CG tracker has been limited to particle parallelism in the past, this new method adds spatial parallelism via domain decomposition. This permits calculations of problems with a very large degree of geometric complexity, which are not possible through particle parallelism alone. In this method, the cells, as well as the particles, are decomposed across pro-

cessors and a particles is communicated to an adjacent processor when it tracks to an interprocessor boundary.

1.1 Motivation for Embedding Meshes within a Combinatorial Geometry

In problems where the problem geometry contains large regions with homogeneous material properties, a combinatorial geometry (CG) representation is most useful for representing such a geometry. There would be no superfluous facet crossing events from a homogeneous mesh cell of one material into another mesh cell of the same material. In a combinatorial geometry, the zonal boundary crossings will only be between the user-defined cells.

Unfortunately, such a representation is not the most efficient when the problem geometry is highly heterogeneous; in which case a mesh representation is more appropriate. One may also wish to opt for a mesh representation when the transport problem in question is coupled to a thermal-hydraulics solver which are usually mesh based. The question addressed here is whether both geometric representations can be used simultaneously.

One answer is to use a CG representation in the regions of the problem geometry for which it is the best representation. Likewise, one should use a mesh representation in those areas of the problem geometry in which a mesh representation is optimal. One then needs to be able to "stitch" the two representations together wherever they meet and ensure the continuity of the transport solution across the boundaries of these regions. Additionally, it is highly desirable that the transport code be responsible for determining the complex connectivity between mesh and combinatorial geometry regions, and not burden the code user with the specification of what can be a very complex geometric interface.

1.2 Motivation for Spatial Domain Decomposition of a Combinatorial Geometry

Previous methods of parallelizing a combinatorial geometry Monte Carlo particle transport code were implemented using a method known as particle parallelism. In this method, the geometry information was stored redundantly on each of the processors, while the particle workload was divided among the processors. This method claims to be "embarrassingly parallel" in the sense that the processors can run independently of each other, until the end of the calculation, when a final answer is calculated by summing results from each of the processors.

The question arises as to how to use this form of parallelism when the geometry is of a complexity that it cannot be stored within the memory of a single processor. Our approach relates to spatial parallelism via domain decomposition, where the geometry is partitioned into spatial domains which are assigned to processors. The number of spatial domains is chosen such that the memory required for each domain will fit within the available memory on each processor. As a particle streams from one domain to another, it must be communicated from one processor to another. The technique of domain decomposition is commonly used in parallel finite-difference or finite-element physics simulations running on a mesh. There are well known techniques for partitioning a mesh into domains. In contrast, there is is no underlying "mesh" in a combinatorial geometry, which consists only of cells and their bounding surfaces.

2 THE EMBEDDED-MESH PARTICLE TRACKER

In the ***Mercury*** implementation of this hybrid scheme of representing the problem geometry, the mesh representation "overlays" and takes precedence over the combinatorial geometry, with one exception. The mesh representation can have a surrounding layer of a "background" material. This background material is used in lieu of the actual geometry that surrounds the mesh representation. In this case, it is useful to have a way to exclude such material regions from the mesh; and have the combinatorial geometry description take precedence. In the current implementation, the user is able to specify a list of materials that are to be excluded from the mesh for the determination of the boundary between the mesh and the combinatorial geometry.

Subject to the above prescriptions, the code finds all zonal facets that form the mesh / combinatorial geometry boundaries. It then determines a location that is offset slightly from the facet in the outward direction and executes the combinatorial geometry "Where am I?" routine to find the CG cell that adjoins the boundary. Once the code determines which mesh cells connect to which CG cells, that connectivity information is used to populate both the mesh and CG adjacency data structures. As a result, the code has the requisite connectivity going in each direction: from mesh to combinatorial geometry and combinatorial geometry to mesh. Therefore, ***Mercury*** can seamlessly track simulation particles from one geometric representation to the other and back again.

It is desirable that the implementation of this hybrid geometric scheme be able to exploit the capabilities provided by large multiprocessor supercomputers. The ***Mercury*** mesh tracker exploits both domain decomposition and domain replication multiprocessing paradigms. In its current implementation, ***Mercury*** adds a full description of the combinatorial geometry to each processor. Since every process has the complete combinatorial geometry model, when a particle transitions from mesh to CG, the processor that was assigned the particle's "previous" mesh domain can continue to track the particle, since that processor has a complete description of the CG.

However, when a particle transitions from combinatorial geometry to mesh, it may enter a mesh domain that is assigned to a different processor than the one that is currently tracking the particle. In such a circumstance, it is necessary to communicate the particle to the processor assigned to the mesh domain that the particle is entering for further tracking. Due to this requirement, any processor that is assigned a domain on the mesh / combinatorial geometry boundary must be able to communicate with any other processor that owns a mesh domain on the boundary.

In problems that have only domain decomposed mesh geometry, each mesh domain has, at most, 6 neighboring domains for a 3-D Cartesian mesh: domains to the "left", "right", "backward", "forward", "down" and "up". Hence, a given domain needs, at most, 6 sets of domain to domain communication paths and communication buffers. The communication patterns implied by the hybrid embedded geometry are much more dense and interconnected. As a result, the domain connectivity of the CG scales super-linearly with the number of mesh domains.

In order to avoid the proliferation of inter-processor communications paths, ***Mercury*** is able to realize the desired communications interconnectivity by "piggy-backing" on the regular 6 nearest neighbor connectivity one finds in a 3-D logically Cartesian model. The code employs a dynamic load balancing algorithm to optimally allocate mesh domains to processors. Once the optimal

allocation of domains to processors and the requisite communication pathways are established, one can view the inter-processor communications network as a directed graph. One then asks the question "What is the shortest path from any one node (processor) to any other node?" This is a solved problem in the field of applied mathematics. An optimal search algorithm, known as Dijkstra's Algorithm, has already been discovered. Once the pathways are defined, each processor employs Dijkstra's Algorithm to find the shortest path to every other processor. Then each processor stores the identity of the first processor on the optimal path in a "routing table".

Whenever a processor needs to pass a particle to any other processor, even if it has no direct connection to that processor; it looks up the desired destination processor in its routing table and finds the identity of a processor that it does have a direct connection to; that is the first step on the optimal path. When a processor receives a particle on the communications network, if first checks if it is the final destination processor for that particle. The destination processor information is also captured as part of the mesh / combinatorial geometry connectivity step described above. If the particle is destined for the processor that just received it, that processor adds the particle to its particle vault for tracking. If the particle is destined for some other processor, the processor that just received it consults its own routing table to find the next processor on the optimal path and passes the particle along to that processor. Eventually, the particle reaches the destination processor.

3 APPLICATION OF THE EMBEDDED MESH PARTICLE TRACKER: THE MITR BNCT FACILITY

An example of an application in which this hybrid-geometry Monte Carlo transport capability can find use is in the calculation of neutron doses in the Boron Neutron Capture Therapy (BNCT) facility at the MIT Nuclear Reactor Laboratory [3],[4]. In this facility, (see Figure 1) thermal neutrons from the core of the MITR-II nuclear reactor pass through a channel in the reactor's graphite reflector. When the cadmium shutter is open, neutrons are permitted to impinge on an array of depleted reactor fuel elements in order to generate high energy neutrons. The spectrum of this beam of fast neutrons is further filtered to provide epithermal neutrons for the BNCT irradiation facility. Epithermal neutrons are chosen for this application because they produce a lower dose to the patient than would thermal neutrons.

Our embedded-mesh model of this facility (see Figure 2) represents the reactor core (shown as the large red volume in Figures 1 and 2) with a 3-D Cartesian mesh region, while the remainder treatment facility is represented by a 3-D combinatorial geometry region. The size of the core, in comparison to the entire facility, would render a fully-meshed representation of the facility extremely inefficient. In addition, this hybrid approach to particle tracking would allow for multi-physics simulations of the core region (such as couple thermal-hydraulics and neutronics on the mesh), while solely modeling neutron transport through the remainder of the facility.

Our model of the BNCT facility employs a point detector tally, which is defined at the center of the small-radius beam exit of the conical collimator. The radiotherapy fluence and spectrum have been computed using ***Mercury*** and these preliminary results compare favorably to measured values published by MIT [4]. In particular, for a reactor core power of $P_{core} = 5\,MW$ (and assuming a useful energy release per fission of $190\,MeV$), the power within the ex-core con-

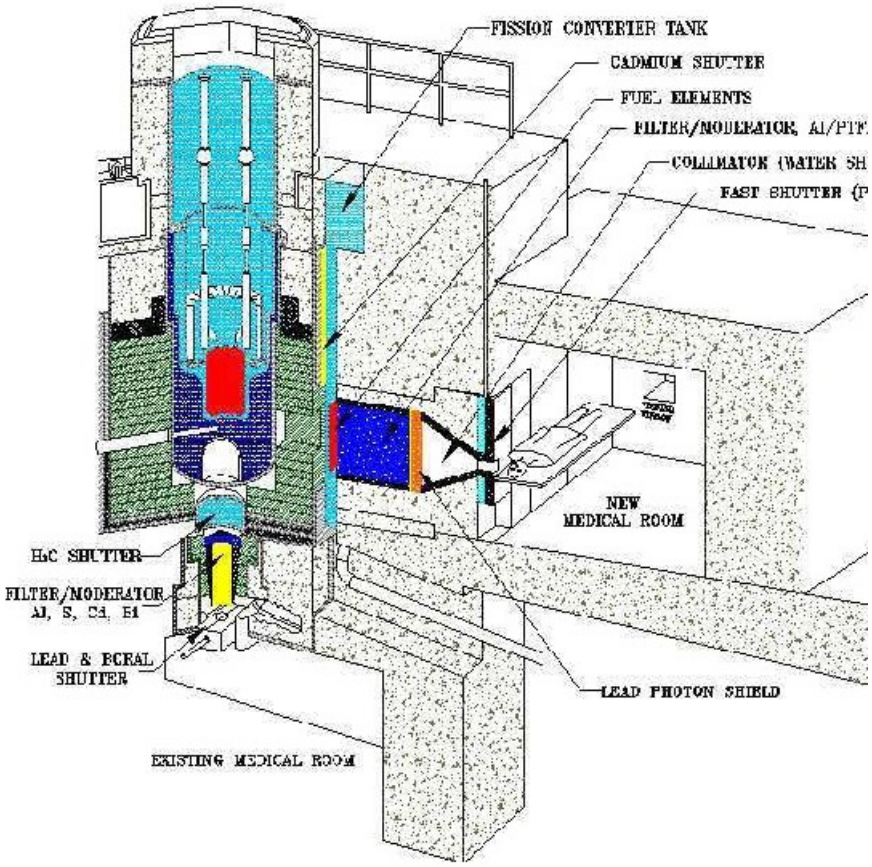


Figure 1. Schematic of the MIT BNCT facility.

verter fuel assemblies is quoted by MIT to be $P_{conv}^{MIT} = 83\,kW$, while our calculation obtains a converter power of $P_{conv}^{LLNL} = 83.8\,kW$.

Of course, the interesting quantity to compare from a treatment perspective is the fluence at the exit of the conical collimator. The MIT researchers quote a treatment fluence of $\psi^{MIT} = 4.6\times10^{9}\,cm^{-2}s^{-1}$, and our continuous-energy cross section calculation produces $\psi^{LLNL} = 5.8\times10^{9}\,cm^{-2}s^{-1}$. While our preliminary results for the fluence are *~26.1%* too large, we continue to examine our model and the drawings provided by MIT to determine if we have accurately modeled the conical collimator and shield region. In addition, we have noticed a large sensitivity of the computed fluence to the choice of evaluated cross section set. In particular, the ENDF/B-VI data set (which was used for this calculation) includes three resonances in ^{27}Al which are not included in the ENDF/B-VII data set. A ***Mercury*** calculation using ENDF/B-VII data yielded a therapy fluence that is more than *60%* too large relative to the MIT data.

Our calculated current spectrum at the beam exit is presented in Figure 3. Our results are in good agreement with the smallest magnitude spectrum provided in Figure 4 of [4]. In particular,

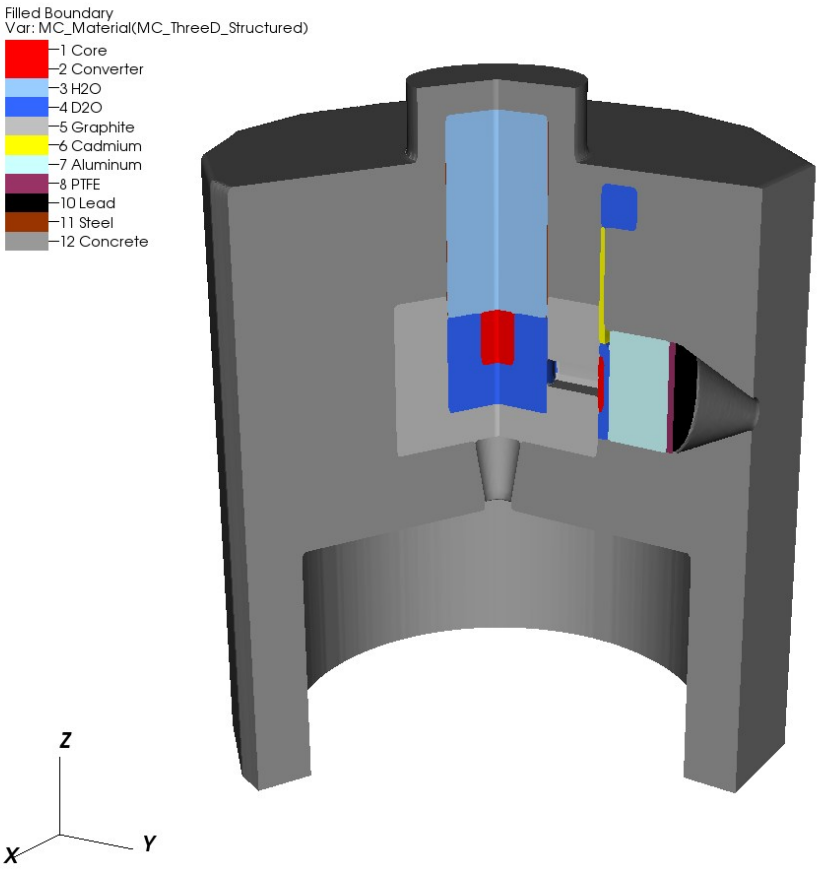


Figure 2. The *Mercury* embedded-mesh (mesh + combinatorial geometry) model of MIT BNCT facility

each spectrum rises rapidly between $10^{-2}\,eV$ and $1\,eV$, exhibit near constant intensity between $1\,eV$ and $10^{4}\,eV$, and falls off rapidly between $10^{4}\,eV$ and $10^{6}\,eV$.

4 DOMAIN DECOMPOSITION OF THE PARTICLE TRACKER

The ***Mercury*** implementation of the combinatorial geometry particle tracker employs linear (1st order) and quadric (2nd order) surfaces, such as planes, spheres, ellipsoids, cylinders, cones, etc. These surfaces as represented by a list of the coefficients in the implicit equation satisfied by the points on the surface:

$$f(x,y,z) = \sum_{\substack{0 \le i+j+k \le 2 \\ i,j,k \ge 0}} a_{ijk} (x - x_0)^i (y - y_0)^j (z - z_0)^k = 0 \qquad (1)$$

The surfaces are used to define volumes by considering the points that are "interior"/"below" the surface $\{(x,y,z) : f(x,y,z) < 0\}$, or "exterior"/"above" the surface $\{(x,y,z) : f(x,y,z) > 0\}$.

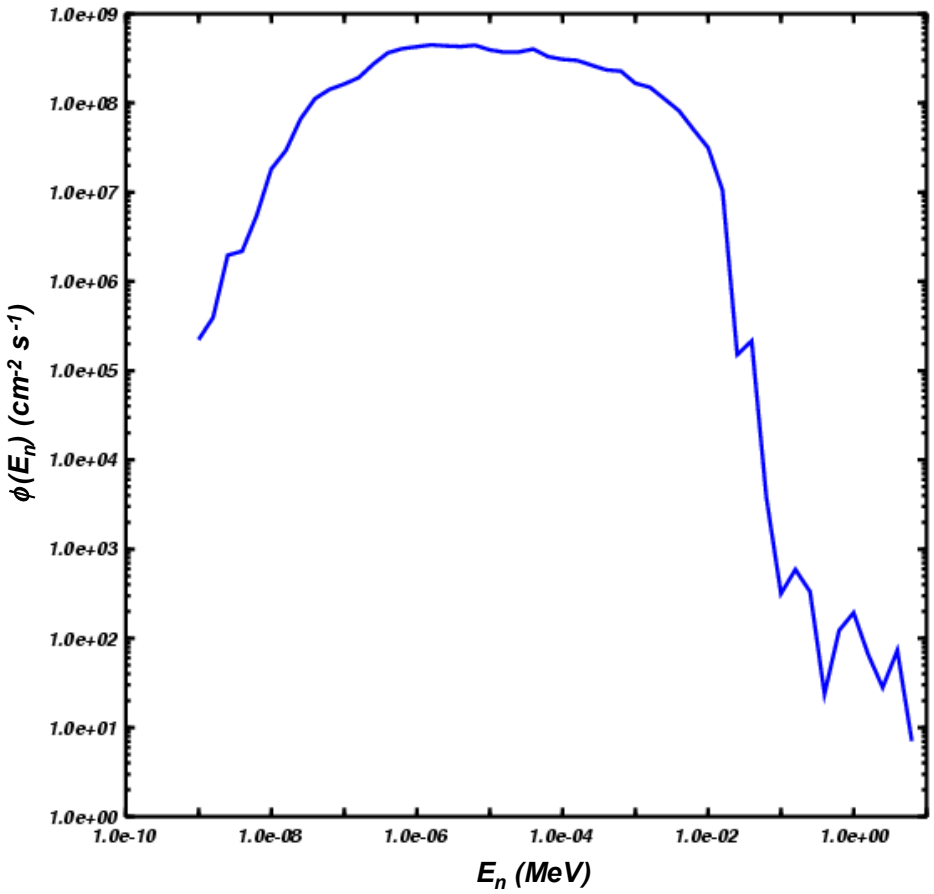


Figure 3. The treatment spectrum of neutrons at the beam exit location within the MIT BNCT facility as calculated by *Mercury*.

Combinatorial geometry cells are defined via the logical aggregation of surfaces using logical operations such as *AND, OR, NOT* to form more complex volumes. An example of the procedure for defining combinatorial geometry cells is shown in Figure 4. Two spherical surfaces, *sphere1* and *sphere2*, are defined. The cells are defined as:

$$cell1 = \text{insideOf}(sphere1)\ \text{AND}\ \text{outsideOf}(sphere2) \tag{2}$$

Using only these simple primitives, one can construct very complicated geometries, such as the National Ignition Facility (NIF) [5] target chamber and support structures shown in Figure 5. The NIF model contains ~2300 surfaces and ~10800 cells.

4.1 Domain Decomposition: Mesh vs. Combinatorial Geometry

It is useful to draw some distinctions between mesh-base and CG-based domain decomposition. In the case when the underlying discretization of your geometry is mesh based, the connectivity of the mesh cells is an integral part of the description of the mesh. If the mesh is topologically Cartesian, then an implicit connectivity of the mesh known by using indexing and striding to move in the *i, j* or *k* directions. If the mesh is unstructured, then a data structure is provided that

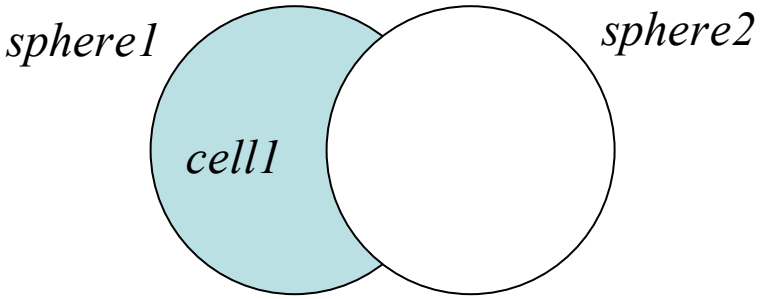


Figure 4. A simple example of creating a CG cell (*cell1*)that is inside of the *sphere1* surface and outside of the *sphere2* surface.

defines the adjacent faces of every face of every cell. This creates an underlying graph that is partitioned into domains.

In the case that the underlying discretization of the problem geometry is CG based, then there does *not* exist any connectivity information about the adjacency of any of the CG cells. As a particle exits a bounding surface of one cell and enters an adjacent cell, it does not (a priori) know what cell it will enter. The algorithm implemented in ***Mercury*** dynamically learns the connectivity of the combinatorial geometry as particles track through it.

The first time a particle exits a cell by crossing a bounding surface, the algorithm loops over all other cells and asks the question "Are my coordinates in the current cell?". Once the answer to the question is "Yes!", the adjacent cell information is saved in a connectivity table to be checked on subsequent particle surface crossings. Therefore, at initialization time, when the domain decomposition of the CG is defined, there is not any connectivity information for the CG

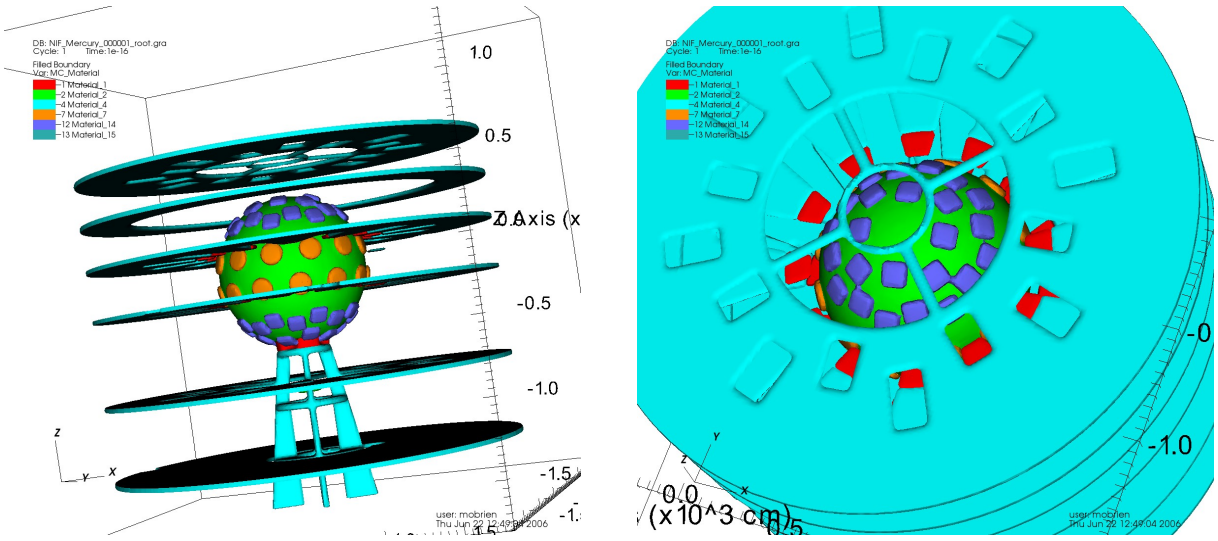


Figure 5. A combinatorial geometry model of the National Ignition Facility (NIF) target chamber and support structures.

cells. As a result, there is no underlying cell-face-neighbor adjacency graph, and graph partitioning algorithms cannot be use to perform the domain decomposition.

To overcome this data limitation, our method employs a technique that relies on the geometric position and extent of each cell as shown in Figure 6. After calculating a bounding box for each cell, the bounding box is used to decide if a given CG surface or cell should exist on a given domain. The user specifies a Cartesian domain decomposition of their problem by defining the positions of decomposition planes normal to the three coordinate axes. Thus only local geometry information is stored on each domain and a scalable algorithm is obtained. Using this method, the domains are themselves "boxes", since they are created from the Cartesian product of boundary planes normal to each of the three coordinate axes. Therefore, the test for membership of a cell within a domain is a simple axis-aligned box-box intersection test. Figure 7 shows a simple 16 way domain decomposition of a collection of spheres, which are color coded by the domain / processor they are assigned to.

The implications of the domain decomposition strategy for particle tracking and I/O in mesh-based and CG-based problems are compared in Table I.

4.2 Combinatorial-Geometry Domain Decomposition Algorithms

The ***Mercury*** Monte Carlo transport code already supported mesh-based domain decomposition [6] when it was decided to add CG-based domain decomposition. As a result, our team leveraged the particle streaming communication that had already been implemented for the mesh-based method to use with the new CSG domain decomposition. Particle streaming communication is the MPI-based communication that occurs when particles cross a domain boundary and need to be sent to an adjacent domain on another processor, in order to continue tracking on the other processor.

4.2.1 What data is distributed across domains?

As the geometric description of a problem gets larger, the following sets of data can grow arbitrarily long: (a) the complete list of surfaces, (b) the list of surfaces that bound each cell, (c) the complete list of cells, and (d) the list of templated (cloned) surfaces and cells. Therefore, a method for distributing this data across processors is required.

Since every object in a CG problem is defined by logical operations on surfaces, the total number of surfaces can be very large. Rather than storing the entire list of surfaces redundantly on every

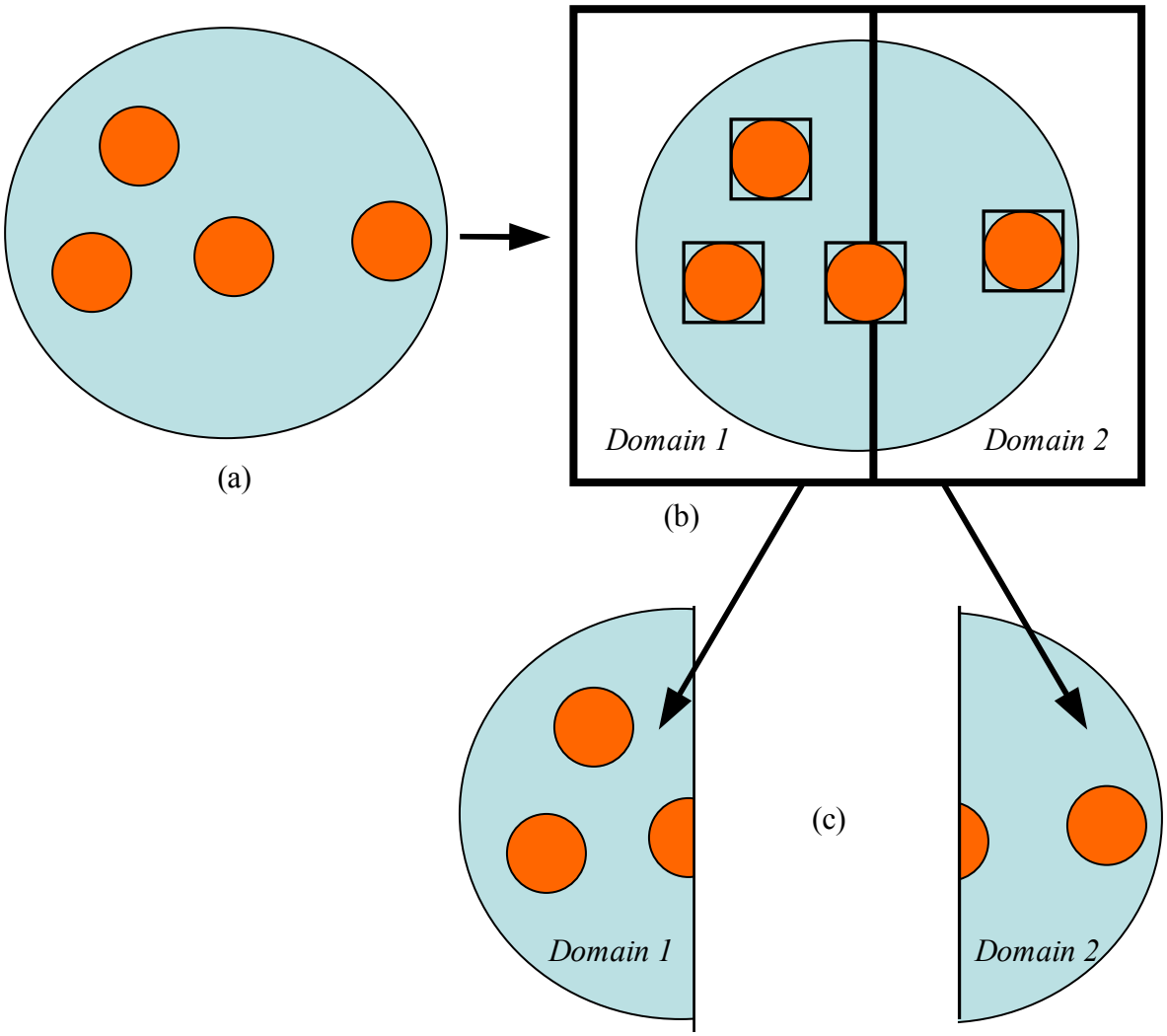


Figure 6. Example of combinatorial geometry domain decomposition: (a) the user defines the global CG problem, without regard to domain decomposition, (b) the user defines the Cartesian domain decomposition by specifying the positions of planes normal to the three coordinate axes, while the code automatically calculates bounding boxes for all cells which are used to test for intersection with each domain, and (c) the code automatically creates *Domain 1* and *Domain 2* and assigns the correct cells to each domain

processor, the approach used in ***Mercury*** is to only store locally those surfaces whose bounding box intersects the bounding box of local domain. The same is true for the cells in the problem: each processor only stores local cells, according to the portion of space that it has been assigned.

The code has a user interface feature known as templates, which is a simple means of defining repeated structures. A user defines a template to be a list of surfaces and cells, and then instantiates the template as many times as they would like, each instantiation having a different translation and/or rotation. For example, one could create a template of a "house", and then instantiate and translate a house template several times to create a neighborhood. This list of templates can

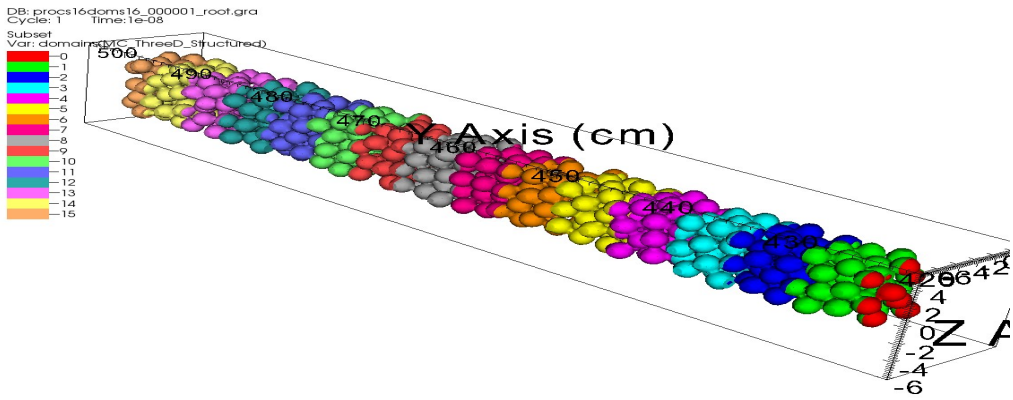


Figure 7. Domain decomposition of a collection of spheres into 16 domains. The CG cells are color coded according to the domain / processor they are assigned to.

Table I. Comparison of the available information and the underlying algorithms for mesh-based vs. CG-based domain decomposed particle tracking.

Event	*Mesh*	*Combinatorial Geometry*
Cell Boundary Crossing	Adjacent cells known	Adjacent cells not explicitly known / Must check adjacent candidate cells
Domain Boundary Crossing	Adjacent domains known	Adjacent domains known
Input	Input description is already domain decomposed within the binary dump file	Must decide which processor each surface/cell should be assigned to (Need to domain decompose user CG input)
Output (Graphics)	Each processor writes output for its domains, and a master file describes how to assemble the pieces	Each processor writes the portion of space it owns, explicitly introducing domain boundary surfaces for cells on domain boundaries, and a master file describes how to assemble the pieces

also get very large. Therefore, ***Mercury*** calculates bounding boxes for templates and only instantiate them on domains whose bounding box intersects the template's bounding box.

4.2.2 Scalability issues

In the case of a mesh, the geometry is defined by a mesh generator which domain decomposes the mesh and stores the domains in separate files. Domains are then assigned to processors, such that each processor only has knowledge of its local domains, and no processor has knowledge of the global description of the geometry. That is in contrast to the CG, where the user defines the problem geometry using input commands that specify the surfaces and cells for the entire problem. In the interest of scalability, the domain decomposition algorithm employed in ***Mercury*** requires that each processor filter out the parts of the global CG problem description that are not assigned to it, based upon residence in the domain bounding box.

Scalability issues occur only at initialization time and are limited to (a) the entire CG description is obtained from an input text file which must be read into memory at once, (b) the entire list of surfaces/cells is read in, and a surface/cell is kept on the local domain only if its bounding box intersects the domain's bounding box, and (c) the entire list of surfaces that define a cell are read in, and only those surfaces that intersect the domain that the cell is on are kept locally.

4.2.3 Scalability solutions

After initialization, when the cells are partitioned according to their domain bounding box, each processor only stores domain-local information. Therefore, the algorithm is scalable. However, one problem remains to be solved: How does one initialize the CG geometry locally, such that each processor only has to deal with domain-local geometry and not the global geometry?

It is possible to treat the CG input in a manner that is similar to how mesh geometry is treated: the definition of surfaces, cell and templates could be decomposed into separate files, and each processor only reads input for the domains that are assigned to it. However, this solution has not been implemented. It has the disadvantage of requiring more work of the user, since they would have to split up their CG input file into several files, each file containing geometry in some specified bounding box.

If a large cell count arises due to repeated hierarchical structures, our approach achieve scalability through the input "template" mechanism. For example, suppose a user wants to model a city composed of 1,000 houses. They can create a template of a house, which might consist of 2,500 cells. Assume that each CG cell requires about 7 kilobytes of memory. The total memory requirement for this geometry is:

$$(1{,}000 \text{ houses/city}) * (2{,}500 \text{ cells/house}) * (7\text{K/cell}) = 17.5\text{GB/city} \quad (3)$$

In most cases, 17.5 GB is more memory than exists on most single processors. However, domain decomposition permits the distribution of the geometry (required memory) across multiple processors such that the entire problem can be run. Input templates are only instantiated on processors that contain domains which intersect the template's bounding box, hence scalability is achieved through the use of input templates.

4.2.4 Calculating the bounding box of a cell

Bounding boxes are required for both the surfaces and cells in order to domain decompose the CG. The surfaces in ***Mercury*** are assumed to be just quadric surfaces, specified by coefficients a_{ijk}, such that $i,j,k \geq 0$ and $0 \leq i,j,k \leq 2$, and a translation (x_0, y_0, z_0). These surfaces are created from user input, where the user specifies the type of surface, such as:

> *Plane_X, Plane_Y, Plane_Z, Plane, Sphere, Ellipsoid, Cylinder_X, Cylinder_Y, Cylinder_Z, Cylinder, Cone_X, Cone_Y, Cone_Z, Cone,* etc.

In addition to storing the surface coefficients and translation, the code also stores an enumerated type describing the type of the surface. Given the type of the surface, the code calculates its bounding box.

For example, a plane normal to the X-axis, has a surface equation:

$$x + a_{000} = 0 \tag{4}$$

Axis-aligned bounding boxes are also stored, which are specified by the minimum and maximum coordinates, in this case:

$$Min = (-a_{000}, \infty, \infty), \quad Max = (-a_{000}, \infty, \infty) \tag{5}$$

Note that it is possible to have infinite extent in any or all of the coordinate directions. This is the case for an unbounded surface, such as a plane that is not normal to any of the coordinate axes. When an unbounded surface is intersected with any domain, there will always be an intersection so unbounded surfaces will be assigned to all processors.

Another example bounding box calculation is that of a spherical surface, which has the surface equation:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 + a_{000} = 0 \tag{6}$$

In this case, the axis-aligned bounding box is given by:

$$\begin{aligned} Min &= (x_0 - \sqrt{-a_{000}}, y_0 - \sqrt{-a_{000}}, z_0 - \sqrt{-a_{000}}) \\ Max &= (x_0 + \sqrt{-a_{000}}, y_0 + \sqrt{-a_{000}}, z_0 + \sqrt{-a_{000}}) \end{aligned} \tag{7}$$

Once every surface has an axis-aligned bounding box, it is used to filter out non-local surfaces. Since every domain also has a bounding box, each domain only keeps the surface whose bounding boxes intersect the domain's bounding box.

As the CG cells are built up from surfaces, the code calculates cell bounding boxes from surface bounding boxes. A CG cell is recursively defined as a tree of CG cells, with an operator defined on the children of a parent cell. There are two binary operators (*and*, *or*), and one unary operator (*not*). All CG cells are classified either as parent or leaf cells. Parent cells have children, while leaf cells do not. We require all cells to be bounded, so `not(cell)` is unbounded, thus we return an infinite bounding box for that case.

4.2.5 Cell parsing

A simple filtering method has been implemented when parsing the CG cell data from the user input file. Since the current domain decomposition strategy is Cartesian in nature, every domain has an axis-aligned bounding box. The bounding box algorithm described above is used to calculate a bounding box for each cell. Each domain inserts a new cell onto its list of cells if the cell's bounding box intersects with the domain's bounding box. This also implies that cells which straddle domain boundaries are inserted into multiple domains.

4.2.6 Locate coordinate

One of the most fundamental algorithms that a Monte Carlo transport code must implement is: "Given a point in space, which cell contains that point?" The modification to this algorithm that is required to support domain decomposition is trivial, since an algorithm existed that works for the case of no domain decomposition. In order to use the existing algorithm, a domain filtering step as also implemented, as shown in Figure 8. If the point in question is outside of the domain in question, that domain can immediately reject ownership of the particle. If the point in question is inside of the domain in question, then proceed with the existing algorithm. During execution of the the *Is-Point-In-Cell* routine, if there is more than 1 domain, then the algorithm ensures that the input particle is inside of the input domain. If that test passes, continue as before; otherwise the particle is definitely not on the input domain.

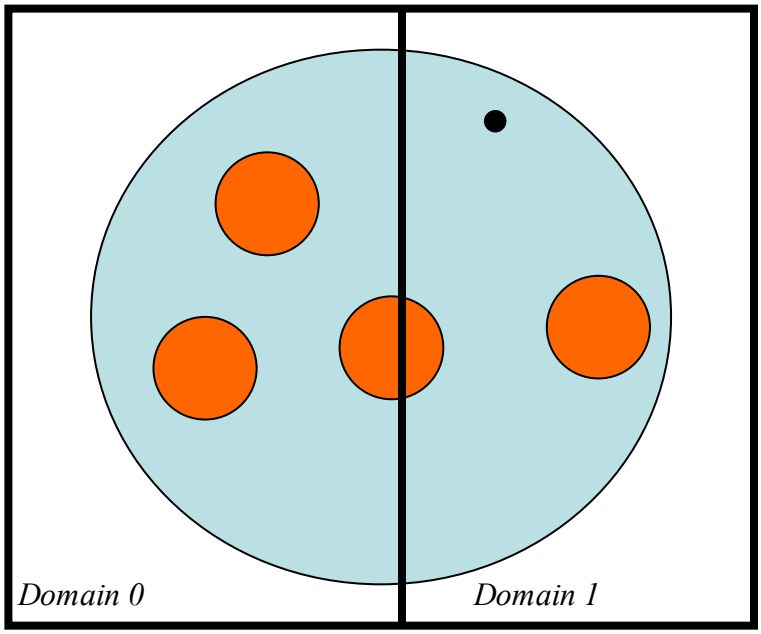


Figure 8. The bold black lines are domain boundaries. The black dot illustrates the position of a particle. The particle is outside of *Domain 0*, hence it can immediately reject ownership of the particle. The particle is inside of *Domain 1*, so it must proceed as usual to test to see which cell the particle is in.

4.2.7 Nearest facet

Another necessary algorithm to implement in a Monte Carlo particle tracking code is : "What is the nearest facet to the current particle position based upon its direction of travel?". This algorithm is isomorphic to ray tracing. As a particle streams through a CG cell, it will eventually reach the boundary of the current cell and cross into the next cell. Given the particle's position and velocity, the *Nearest-Facet* routine calculates the distance to all of the bounding surfaces of the cell, and it selects the nearest boundary surface that the particle will cross. In the case of a domain decomposed CG particle tracker, the existing (serial) nearest facet algorithm is employed with one modification. The distance to the next nearest domain boundary interface must also be determined. If the distance to the nearest domain boundary interface (d_1) is less than the distance to the nearest cell boundary surface (d_2) , then the particle must be communicated to the adjacent domain (see Figure 9).

The ***Mercury*** code had already domain decomposition for mesh-based problems. As a result, it already had the required infrastructure to buffer and communicate particles among adjacent domains. Therefore, once it is determined that the particle should undergo a CG domain boundary crossing, the existing infrastructure is used to communicate the particle from its current domain to the adjacent domain.

5 APPLICATION OF THE DOMAIN-DECOMPOSED PARTICLE TRACKER: THE LIFE ENGINE

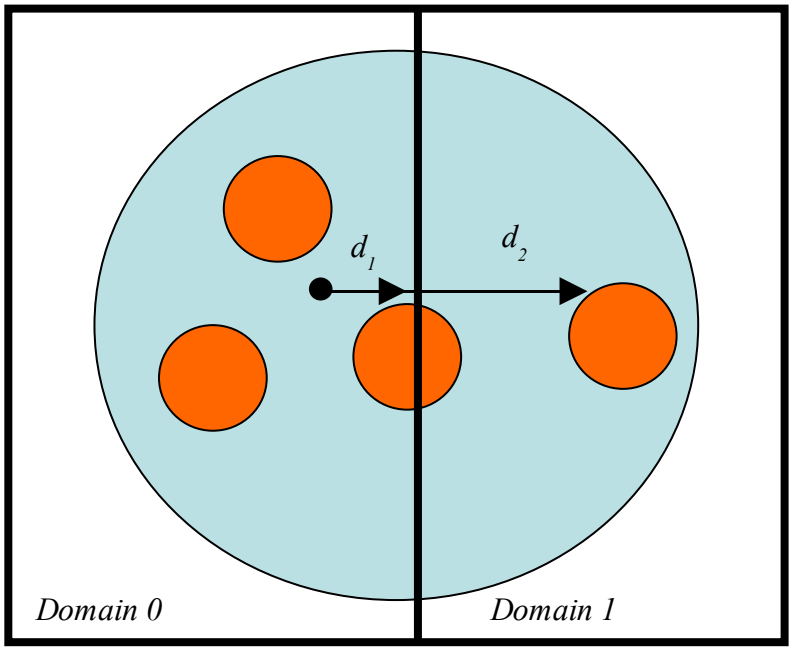


Figure 9. This example shows that the distance to the domain boundary crossing (d_1) is closer than the distance to the nearest facet (d_2) , hence the particle will be communicated from *Domain 0* to *Domain 1*.

Lawrence Livermore National Laboratory (LLNL) is in the final stages of constructing the National Ignition Facility (NIF), the world's largest and most powerful laser. One possible application of a NIF-like laser, is to use it for fissile material transmutation and electricity production. This is the idea behind the Laser Inertial Fusion-Fission Energy (LIFE) engine [7]. The lasers are fired on a tiny D-T capsule within a holhraum in the center of the target chamber. This causes the capsule to compress and undergo nuclear fusion, releasing large quantities of high energy neutrons. These neutrons stream out of the holhraum and into a layer of fissionable fuel material within the spherical, subcritical LIFE engine. This fuel, which can be derived from many sources including spent reactor fuel and weapons grade material, is in the form of packed pebbles, which themselves are comprised of several thousand TRISO pellets. These fuel pebbles undergo fission by the high-energy fusion neutrons and release heat which is used to generate electricity.

A detailed simulation of neutron transport in this facility requires a very large and complex geometric description. To model only a very small portion (a 1º by 1º solid angle) of the fuel layer of the LIFE engine requires ~5.6 million CG cells (see Figure 10):

$$569 \text{ pebbles} * 2445 \text{ pellet/pebble} * 4 \text{ layers/pellet} = 5.6 \text{ Million CG cells} \quad (8)$$

A full 4π model the LIFE engine would require billions of CG cells.

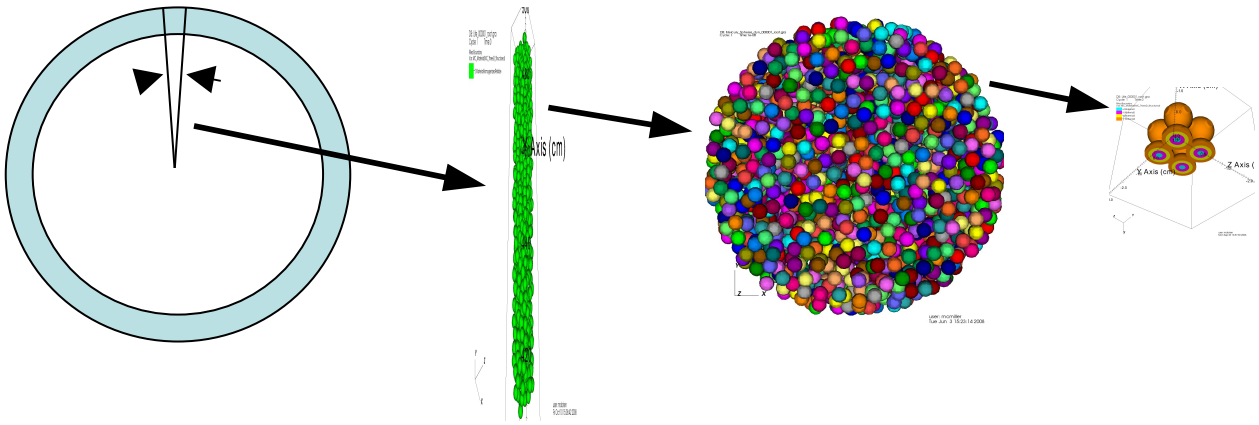
Our 1º by 1º solid angle wedge model of the LIFE engine calculates the neutron scalar flux distribution in 175 binned energy groups. Therefore, each CG cell requires at least 175 double precision floating point numbers, in addition to the data structure fields for describing cells and surfaces. The memory requirement for this "small" LIFE problem is 36GB of memory for the geometry alone, while additional memory is required for the particles. This is typically more memory than any one processor has, so the problem must be distributed across processors if it is to be solved. The ***Mercury*** code is in a unique position to solve extremely large scale, detailed problems like this.

5.1 Preliminary Results

An initial test of the domain-decomposed particle tracker focused on transporting neutrons though a single LIFE pebble. For this initial test, the 4 layers in each TRISO pellet were homogenized. As a result, this single pebble is comprised of 2446 CG cells (as shown in Figure 11): 2445 TRISO pellet cells and 1 pebble filler cell.

Table II shows the time required for ***Mercury*** to calculate a *k* eigenvalue on the single LIFE pebble. The number of particles was the same for each calculation, but the number of domains and processors was varied. To begin the analysis of the data, look at the 1 processor data in the second column. One notices that as the number of domains is increased from 1 to 8, the calculation actually runs faster. This is due to a localization of geometry to each domain, as shown in Figure 12. The calculation avoids non-local distance to facet calculations which are "impossible, since not all of the CG cells are on the current domain.

As more domains are added to a problem (for a given processor count), it further localizes the geometry. However, there is a competing effect of calculating the distance to the new domain



(a)	(b)	(c)	(d)
LIFE Target Chamber 1° by 1° wedge	569 Pebbles	2445 TRISO Pellets per Pebble	4 Layers per TRISO Pellet

Figure 10. The hierarchical layout of the "small" model of the LIFE engine: (a) the target chamber (inner radius 423 cm, outer radius 504 cm), (b) a 1° by 1° wedge of pebbles contains 569 pebbles surrounded by Flibe coolant (made of Li, Be and F), (c) each pebble (1cm radius) is contains 2445 TRISO pellets surrounded by pyrolytic carbon, and (d) each TRISO pellet (radius of 497 μm) has 4 layers (Layer 1: U^{238}, O, C; Layers 2 and 3: C; Layer 4: SiC).

boundaries that are introduced. For example, on 1 processor, increasing the number of domains from 16 to 64, results in the run time increasing from 686 seconds to 732 seconds. In this case, the cost of tracking to 48 additional domain boundaries outweighed the cost savings associated with localization of the geometry.

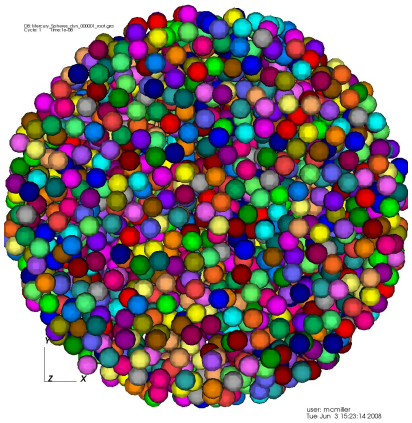
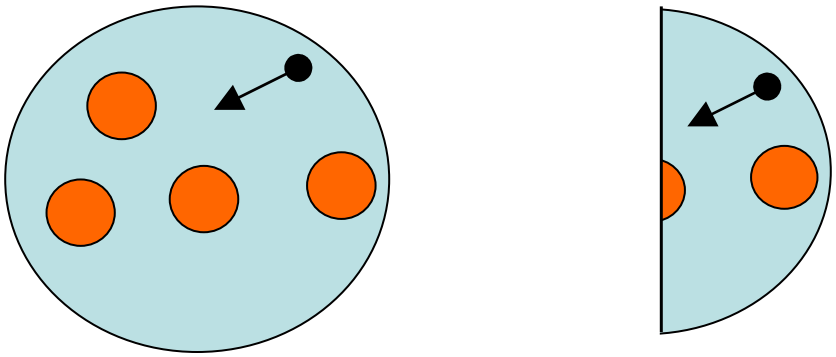


Figure 11. A single LIFE pebble containing 2445 homogenized TRISO pellet CG cells.

Table II. The time required to complete a *k* eigenvalue calculation in the homogenized-TRISO LIFE pebble for various domain and processor configurations.

Time Required to Complete k Eigenvalue Calculation (sec)					
	1 processor	***2 processors***	***4 processors***	***8 processors***	***16 processors***
1 domain	848	427	226	131	74
2 domains	736	235	148	82	52
4 domains	668	190	65	34	20
8 domains	659	162	57	20	12
16 domains	686	214	113	32	12
64 domains	732	207	116	37	18

Now examine the data for 16 processors. When the problem is run with 1 domain, the traditional method of parallelizing Monte Carlo CG transport calculations, each processor has is assigned all of the geometry, and the particle workload is divided evenly among the processors. This configuration requires 74 seconds to complete When the calculation is divided into 16 domains on 16 processors, it only requires 12 seconds, which is more than a factor of 6 speedup! Once again, this performance improvement is due to localization of the geometry. When a particle is inside of the pebble filler cell, it must calculate the distance to the 2445 TRISO pellets surfaces without domain decomposition. However, when the problem is decomposed into 16 domains, it only has to calculate the distance to ~2445/16 = ~153 surfaces. Competing with the speedup arising from localization of the geometry, the particle interprocessor (interdomain) communication leads to an overhead that results in the calculation running slower on 64 domains than 16 domains. This example illustrates that domain decomposition can result in shorter run times than particle parallelism: compare the timings for 16 processors, 1 domain (74 seconds for particle parallelism) to 16 processors, 16 domains (12 seconds for spatial parallelism).



(a) Without Domain Decomposition

(b) With Domain Decomposition

Figure 12. (a) Without domain decomposition, a particle in the pebble filler (blue) cell must calculate the distance to 2445 (orange) surfaces, which is very expensive. (b) With domain decomposition, a particle in the pebble filler cell must calculate the distance to only local surfaces on this domain, which is significantly faster.

5.2 Dynamic Load Balancing

The ***Mercury*** code has an existing dynamic load balancing algorithm [6] which is independent of the underlying geometry discretization (mesh or CG). When the number of processors is greater than the number of domains, the code will assign multiple processors to domains. In this case, the particle workload will be shared evenly among the processors that are working on a particular domain. This is a hybrid domain-decomposition (spatial parallelism) and domain-replication (particle parallelism) model.

For example, consider once more the "small" model of the LIFE engine shown in Figure 13. The figure shows the 569 pebbles in the 1º by 1º wedge. These spherical cells are domain decomposed into 16 domains (that are stacked in a vertical bed in the figure), but run on 64 processors. In this time dependent calculation, neutrons are injected into the system at the bottom with an upward directed. Initially, each domain will have 64 (processors)/16 (domains) = 4 processors assigned to it. After each cycle of the calculation, the code observes how much work each domain required during that cycle, and redistributes the processors proportional to the workload of each domain.

In this figure, the pebbles are color coded according to the number of processors that are assigned to the domain that the pebble resides in: red / light blue / dark blue means 17 / 4 / 1 pro-

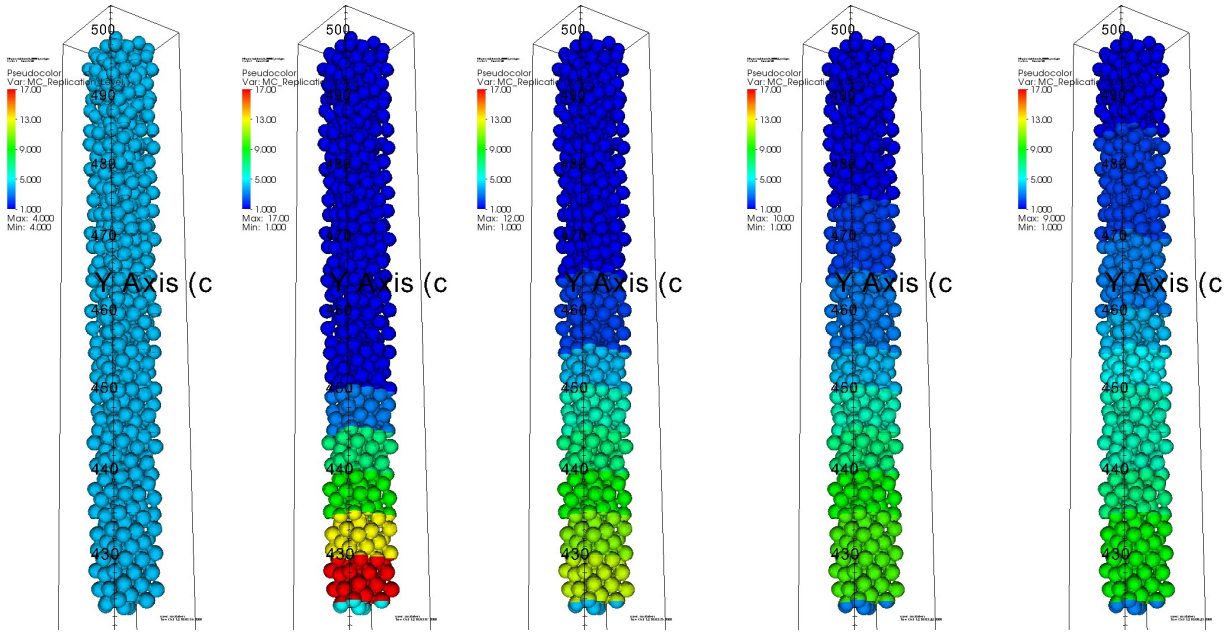


Figure 13. Time evolution of the number of processors assigned to each domain in the "small" model of the LIFE engine. The problem is run with 64 processors and 16 domains. The number of processors assigned to each domain varies dynamically in proportion to the particle workload on the domain. The fuel pebbles are color coded by the number of processors assigned to the domain that the cell resides in.

cessors per domain. It clearly indicates that number of processors assigned to a domain increases between *Cycle 0* and *Cycle 1* in response to the particle workload imbalance. As the particles move upwards in time through the domains, the code reassigns processors to domains, approaching a more uniform distribution (compare *Cycle 1* to *Cycle 4*). In a sense, the processors are "transporting" along with the particles. By allowing the number of processors assigned to each domain to vary dynamically, ***Mercury*** obtained a factor of 1.4 speedup over a uniform, static assignment of 4 processors to each domain.

6 SUMMARY AND CONCLUSIONS

This paper has presented two enhancements to the existing combinatorial geometry particle tracker in the ***Mercury*** Monte Carlo transport code. These modifications allow us to efficiently model problems with a high degree of geometric complexity.

A hybrid particle tracker, in which a mesh region is embedded within a CG region, permits efficient calculations of problems which contain both large-scale heterogeneous regions (modeled with a mesh) and homogeneous regions (modeled with a CG). One application of this methodology is to have ***Mercury*** simulate particle transport in both the mesh-based and CG-based regions, while other physics, such as thermal-hydraulics or structural mechanics, is run in conjunction with particle transport on the mesh.

Our initial application of the embedded-mesh particle tracker is to model neutron transport within the Boron Neutron Capture Therapy (BNCT) facility at the MIT Nuclear Reactor Laboratory. Transport within the reactor core has been modeled on a Cartesian mesh, while the balance of the facility has been modeled with combinatorial geometry. Modeling this system completely via a mesh would have been a daunting challenge, since the core constitutes only a small portion of the facility, and the remainder of the facility includes large homogeneous volumes and several heterogeneous features. Our preliminary results agree favorably with the calculations performed by the MIT reactor designers.

In addition, a second form of parallelism has been added to the CG particle tracker. As a result, our CG tracker now supports particle parallelism via domain replication, as well as spatial parallelism via domain decomposition. ***Mercury*** can now perform calculations of problems with a very large degree of geometric complexity which cannot be solved through particle parallelism alone, due to the large memory requirements of the CG. Our new method decomposes both the cells, as well as the particles, across processors. Therefore, particles are communicated to an adjacent processor when they track to an interprocessor boundary.

The initial application of the domain decomposed CG tracker has been to model a small-solid-angle section of the Laser Inertial Fusion-Fission Energy (LIFE) engine (a subcritical, laser-driven pebble-bed "engine") that is being designed at LLNL. While modeling the neutronics of the entire LIFE engine is far beyond the scope of current supercomputers and transport codes, our initial models of a small portion of the facility have produced rather promising results. Static k eigenvalue calculations of a single pebble (which contains 2445 homogenized TRISO pellets) has shown that domain-decomposed CG particle transport can be faster than use of particle parallelism alone. Using a simple implementation of domain decomposition in which axis-aligned

bounding boxes are defined for surfaces, cells and domains, then localizing the geometry by intersecting bounding boxes and filtering out non-local geometry, has resulted in significant speedups. For example, the serial calculation of this pebble required *659 sec*, while the domain-decomposed calculation with 8 domains and 8 processors required only *20 sec* (superlinear speedup of *32.95* using 8 processors).

A more complicated problem that has been modeled is a 1º by 1º wedge of the LIFE engine. This "small" problem works out to be comprised of 5.6 million CG cells, which would require 36GB of memory for the geometry alone. This amount of memory is typically not available on each processor of current supercomputers, and therefore, particle parallelism alone is not capable of solving this problem. Finally, we have investigated the use of both domain decomposition (spatial parallelism) and domain replication (particle parallelism) in a time dependent version of this problem. Since the neutrons are introduced at the low-radius end of the pebble bed, and move upward through bed in time, the system starts out in a rather load imbalanced state, and approaches a more balanced state as time progresses. By enabling dynamic domain replication, the system is able to better balance the load over multiple cycles, resulting in a speedup over the static assignment of processors to domains.

ACKNOWLEDGMENTS

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

REFERENCES

1. R. J. Procassini, et al., "*Mercury User Guide (Version c.2)*", Lawrence Livermore National Laboratory, Report UCRL-TM-204296, Revision 1 (2008).
2. "Mercury Web Site", Lawrence Livermore National Laboratory, http://www.llnl.gov/mercury (2009).
3. L-W. Hu and J. Bernard, "The MIT Research Reactor as a National User Facility for Advanced Materials and Fuel Research", *IGORR-TRTR Joint Meeting*, Gaithersburg, Maryland, September 12-16 (2005).
4. O. K. Harling, K. J. Riley, et al., "The Fission Converter-Based Epithermal Neutron Irradiation Facility at the Massachusetts Institute of Technology Reactor", *Nucl Sci Eng*, **140**, pp. 223-240 (2002).
5. "The National Ignition Facility: Ushering in a New Age for Science", Lawrence Livermore National Laboratory, https://lasers.llnl.gov/programs/nif (2009).
6. M. J. O'Brien, J. M. Taylor and R. J. Procassini,, "Dynamic Load Balancing of Parallel Monte Carlo Transport Calculations", *The Monte Carlo Method: Versatility Unbounded In A Dynamic Computing World*, Chattanooga, Tennessee, April 17–21 (2005).
7. "LIFE: Clean Energy from Nuclear Waste", Lawrence Livermore National Laboratory, https://lasers.llnl.gov/missions/energy_for_the_future/life (2009).