Load Balancing Unstructured Meshes for Massively Parallel Transport Sweeps in PDT

Tarek Ghaddar
Chair: Dr. Jean Ragusa
Committee: Dr. Jim Morel, Dr. Bojan Popov

Nuclear Engineering Department
Texas A&M University
College Station, TX, 77843-3133

I. Introduction

The steady-state neutron transport equation describes the behavior of neutrons in a medium, and is given by Eq. 1:

$$\vec{\Omega}\cdot\vec{\nabla}\psi(\vec{r},E,\vec{\Omega})+\Sigma_t(\vec{r},E)\psi(\vec{r},E,\vec{\Omega})=\int_0^\infty dE'\int_{4\pi}d\Omega'\Sigma_s(\vec{r},E'\to E,\Omega'\to\Omega)\psi(\vec{r},E',\vec{\Omega}')+S_{ext}(\vec{r},E,\vec{\Omega}), \tag{1}$$

where $\vec{\Omega}_m\cdot\vec{\nabla}\psi$ is the leakage term and $\Sigma_t\psi$ is the total collision term (absorption and outscatter). These are the loss terms of the neutron transport equation. The right hand side of Eq. 1 represent the gain terms, where S_{ext} is the external source of neutrons and $\int_0^\infty dE'\int_{4\pi}d\Omega'\Sigma_s(E'\to E,\Omega'\to\Omega)\psi(\vec{r},E',\vec{\Omega}')$ is the inscatter term, which represents all neutrons scattering from energy E' and direction $\vec{\Omega}'$ into energy dE about E and $d\Omega$ about direction $\vec{\Omega}$.

In order to solve this equation, a couple of assumptions and approximations are used. First, we assume isotropic scattering, which turns Eq. 1 into:

$$\vec{\Omega}\cdot\vec{\nabla}\psi(\vec{r},E,\vec{\Omega})+\Sigma_t(\vec{r},E)\psi(\vec{r},E,\vec{\Omega})=\frac{1}{4\pi}\int_0^\infty dE'\int_{4\pi}d\Omega'\Sigma_s(\vec{r},E'\to E)\psi(\vec{r},E',\vec{\Omega}')+S_{ext}(\vec{r},E,\vec{\Omega}) \tag{2}$$

where the inscatter term is simplified. The double differential scattering cross section, $\Sigma_s(E'\to E,\Omega'\to\Omega)$, no longer depends on direction, and is divided by 4π to reflect isotropic behavior. The next step to solving the transport equation is to discretize in energy, yielding Eq. 3 the multigroup transport equation:

$$\vec{\Omega}\cdot\vec{\nabla}\psi_g(\vec{r},\vec{\Omega})+\Sigma_{t,g}(\vec{r})\psi_g(\vec{r},\vec{\Omega})=\frac{1}{4\pi}\sum_{g'}\Sigma_{s,g'\to g}(\vec{r})\phi_{g'}(\vec{r})+S_{ext,g}(\vec{r},\vec{\Omega}), \tag{3}$$

where the transport equation becomes a coupled system of equations, with each energy group having an equation. We also note that because the cross section no longer has angular dependence, we are left with the scalar flux because $\int_{4\pi}d\Omega'\psi(\vec{r},\vec{\Omega}')\equiv\phi(\vec{r})$.

Next, we discretize in angle, introducing another coupled system of equations, with each equation represented by Eq. 4:

$$\vec{\Omega}\cdot\vec{\nabla}\psi_{g,m}(\vec{r})+\Sigma_{t,g}(\vec{r})\psi_{g,m}(\vec{r})=\frac{1}{4\pi}\sum_{g'}\Sigma_{s,g'\to g}(\vec{r})\phi_{g'}(\vec{r})+S_{ext,g,m}(\vec{r}), \tag{4}$$

where the subscript m is introduced to describe the angular flux in direction m. We notice that the subscript is not added to our inscatter term because of the isotropic scattering assumption, and because the scalar flux does not depend on angle.

In order for this discrete form of the transport equation to yield an accurate solution, a technique called

source iteration is introduced. This is shown by a simplified transport equation Eq. 5:

$$\vec{\Omega}_m \cdot \vec{\nabla} \psi_m^{(l+1)}(\vec{r}) + \Sigma_t \psi_m^{(l+1)}(\vec{r}) = q_m^{(l)}(\vec{r}), \tag{5}$$

where the right hand side terms of Eq. 4 have been combined into one general source term, q_m. In addition, only the angular subscript is shown, and the group discretization is understood. The angular flux of iteration $(l+1)$ are calculated using the (l^{th}) value of the scalar flux and external source in that direction.

We notice that after the angular and energy dependence have been accounted for, the problem must be discretized in space as well. This is done by meshing the domain, and solving the spatial problem one cell at a time for a given direction. The solution across a cell interface is connected based on an upwind approach, where face outflow radiation becomes face inflow radiation for the downwind cells. Sweeping the mesh and solving one cell at a time is possible utilizing one of three popular discretization techniques: finite difference,[?] finite volume,[?] or discontinuous finite element.[?] Figure 1 demonstrates solving one cell at a time in a transport sweep.

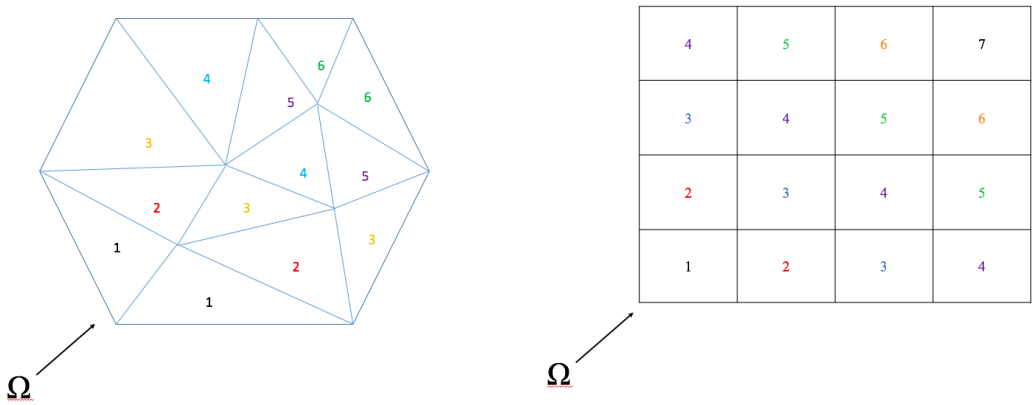

Figure 1. A demonstration of a sweep on a structured and unstructured mesh.

The number in each cell represents the ordering with which the cells are solved. All cells must receive the solution downwind from them before solving for their own solution. This dependency can be represented and stored as a task dependence graph, shown by Fig. 2.

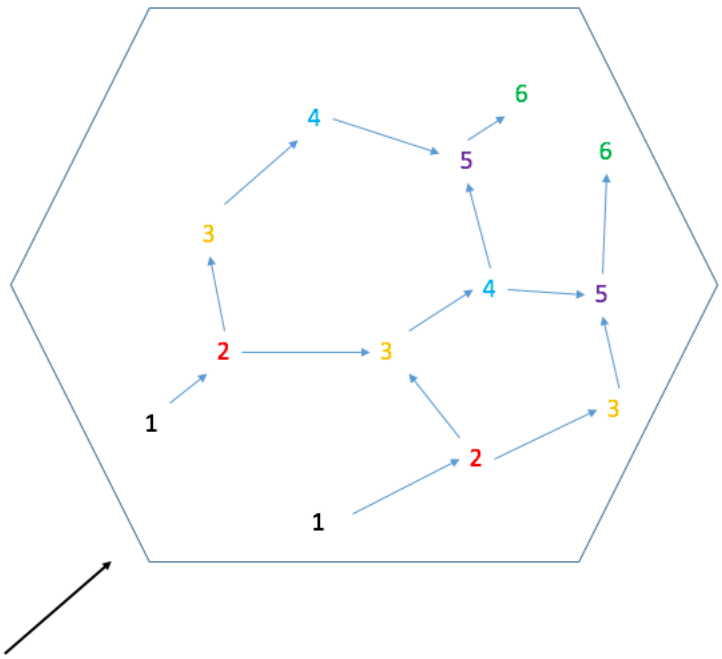

Figure 2. A task dependence graph of the unstructured mesh example in Fig. 1.

The concepts presented in this introduction are used to varying degrees by PDT, Texas A&M University's massively parallel deterministic transport code. It is capable of multi-group simulations, and employs discrete ordinates for angular discretization. Types of simulations include steady-state, time-dependent, criticality, and depletion simulations. It solves the transport equation for neutron, thermal, gamma, coupled neutron-gamma, electron, and coupled electron-photon radiation. PDT has been shown to scale on logically Cartesian grids out to 750,000 cores. All work proposed in this document has been and will be implemented in PDT.

II. Parallelization of Transport Sweeps

As mentioned in the previous section, a transport sweep is set up by overlaying a domain with a finite element mesh.The sweep then solves the transport equation cell by cell using a discontinuous finite element approach. The order of which cell to solve first is given by a task dependence graph, as shown in Fig. 2.

A parallel sweep algorithm is defined by three properties:

- partitioning: dividing the domain among available processors
- aggregation: grouping cells, directions, and energy groups into tasks
- scheduling: choosing which task to execute if more than one is available

The basic concepts of parallel transport sweeps, partitioning, aggregation, and scheduling, are most easily described in the context of a structured transport sweep. A structured transport sweep takes place on a quadrilateral cartesian mesh. Furthermore, the work proposed utilizes aspects of the structured transport sweep.

If M is the number of angular directions per octant, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

In a regular grid, we have the number of cells in each Cartesian direction: N_x, N_y, N_z. These cells are aggregated into "cellsets". However, in an unstructured mesh, the number of cells cannot be described as such. In PDT specifically we initially subdivide the domain into subsets, which are just rectangular subdomains. Within each subset, an unstructured mesh is created. This creates a pseudo-regular grid. These subsets become the N_x, N_y, N_z equivalent for an unstructured mesh. The spatial aggregation in a PDT unstructured mesh is done by aggregating subsets into cellsets.

If M is the total number of angular directions, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

Fine grain work units are aggregated into coarser-grained units called *tasks*. A few terms are defined that describe how each variable is aggregated.

- $A_x = \frac{N_x}{P_x}$, where N_x is the number of cells in x and P_x is the number of processors in x
- $A_y = \frac{N_y}{P_y}$, where N_y is the number of cells in y and P_y is the number of processors in y

- $N_g = \frac{G}{A_g}$
- $N_m = \frac{M}{A_m}$
- $N_k = \frac{N_z}{P_z A_z}$

It follows that each process owns N_k cell-sets (each of which is A_z planes of $A_x A_y$ cells), $8N_m$ direction-sets, and N_g group-sets for a total of $8N_m N_g N_k$ tasks.

One task contains $A_x A_y A_z$ cells, A_m directions, and A_g groups. Equivalently, a task is the computation of one cellset, one groupset, and one angleset. One task takes a stage to complete. This is particularly important when comparing sweeps to the performance models.

Equation 6 approximately defines parallel sweep efficiency. This can be calculated for specific machinery and partitioning parameters by substituting in values calculated using Eqs. 10,11, and 12.

$$\begin{aligned} \varepsilon &= \frac{T_{\text{task}} N_{\text{tasks}}}{[N_{\text{stages}}][T_{\text{task}} + T_{\text{comm}}]} \\ &= \frac{1}{[1 + \frac{N_{\text{idle}}}{N_{\text{tasks}}}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \end{aligned} \tag{6}$$

Equations 7 and 8 show how T_{comm} and T_{task} are calculated:

$$T_{\text{comm}} = M_L T_{\text{latency}} + T_{\text{byte}} N_{\text{bytes}} \tag{7}$$

$$T_{\text{task}} = A_x A_y A_z A_m A_g T_{\text{grind}} \tag{8}$$

where T_{latency} is the message latency time, T_{byte} is the additional time to send one byte of message, N_{bytes} is the total number of bytes of information that a processor must communicate to its downstream neighbors at each stage, and T_{grind} is the time it takes to compute a single cell, direction, and energy group. M_L is a latency parameter that is used to explore performance as a function of increased or decreased latency. If a high value of M_L is necessary for the model to match computational results, improvements should be made in code implementation.

II.A. KBA Partitioning

Several parallel transport sweep codes use KBA partitioning in their sweeping, such as Denovo[?] and PARTISAN.[?] The KBA partitioning scheme and algorithm was developed by Koch, Baker, and Alcouffe.[?]

The KBA algorithm traditionally chooses $P_z = 1, A_m = 1, G = A_g = 1, A_x = N_x/P_x, A_y = N_y/P_y$, with A_z being the selectable number of z-planes to be aggregated into each task. With $N_k = N_z/A_z$, each processor performs $N_{\text{tasks}} = 8MN_k$ tasks. With the KBA algorithm, $2MN_k$ are pipelined from a given corner of the 2D

processor layout. The far corner processor remains idle for the first $P_x + P_y - 2$ stages, which means that a 2 octant sweep completes in $2MN_k + P_x + P_y - 2$ stages. If an octant-pair sweep does not begin until the previous pair's finishes, the full sweep requires $8MN_k + 4(P_x + P_y - 2)$ stages, which means the KBA parallel efficiency is:

$$\varepsilon_{KBA} = \frac{1}{[1 + \frac{4(P_x+P_y-2)}{8MN_k}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{9}$$

II.B. The Structured Transport Sweep in PDT

The minimum possible number of stages for given partitioning parameters P_i and A_j is $2N_{\text{fill}} + N_{\text{tasks}}$. N_{fill} is both the minimum number of stages before a sweepfront can reach the center-most processors and the number needed to finish a direction's sweep after the center-most processors have finished. Equations 10, 11, and 12 define N_{fill}, N_{idle}, and N_{tasks}:

$$N_{\text{fill}} = \frac{P_x + \delta_x}{2} - 1 + \frac{P_y + \delta_y}{2} - 1 + N_k(\frac{P_z + \delta_z}{2} - 1) \tag{10}$$

$$N_{\text{idle}} = 2N_{\text{fill}} \tag{11}$$

$$N_{\text{tasks}} = 8N_m N_g N_k \tag{12}$$

where δ_u is 1 for P_u odd, and 0 for P_u even.

Figure 3 shows three different partitioning schemes used in transport sweeps.

Figure 3. Three different partitioning schemes in 2D, from left to right: hybrid KBA, volumetric non-overloaded, and volumetric overloaded.

The overloaded volumetric partitioning proceeds as follows:

1. In a 2D (3D) domain, cellsets are divided into 4 (8) spatial quadrants (octants), with an equal number of cellsets in each SQO (SQO is defined as a spatial quadrant or octant).
2. Assign 1/4 of the processors (1/8) in 3D to each SQO.
3. Choose the individual overload factors $\omega_x, \omega_y, and \omega_z$ and individual processor counts $P_x, P_y, and P_z$,

such that $\omega_x\omega_y\omega_z = \omega_r$ and $P_xP_yP_z = P$, with all P_u even. ω_u is defined as the number of cellsets assigned to each P_u.

4. An array of $\omega_x \cdot \omega_y \cdot \omega_z$ "tiles" in each SQO. Each tile is an array of $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ cellsets. These cellsets are mapped one-to-one to the $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ procesors assigned to the SQO, using the same mapping in each tile.

Each tile has a logically identical layout of cellsets, and each processor owns exactly one cellset in each tile in its SQO, making each processor responsible for ω_r cellsets.

The optimal scheduling algorithm rules are as follows:

1. If $i \leq X$, then tasks with $\Omega_x > 0$ have priority, while for $i > X$, tasks with $\Omega_x < 0$ have priority.
2. If multiple ready tasks have the same sign on Ω_x, apply rule 1 to j, Y, Ω_y.
3. If multiple ready tasks have the same sign on Ω_x and Ω_y, apply rule 1 to k, Z, Ω_z.
4. If multiple tasks are ready in the same octant, then priority goes to the cellset for which the priority octant has greatest downstream depth.
5. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x, then priority goes to the cellset for which the priority octant has greatest downstream depth of graph in y.
6. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x and y, then priority goes to the cellset for which priority octant has greatest depth of graph in z.

This ensures that each SQO orders the octants: the one it can start right away (A), three that have one sign difference from$A$(B,C, and D), three that have two sign differences ($\bar{D}, \bar{C}, \bar{B}$), and one in opposition to its primary ($\bar{A}$).

There are three constraints in order to achieve the optimal stage count. In these constraints, $M = \omega_g\omega_m/8$, which is the number of tasks per octant per cellset.

1. $M \geq 2(Z-1)$
2. $\omega_z M \geq 2(Y-1)$
3. If $\omega_x > 1$, then $\omega_y\omega_z M \geq X$

These conditions ensure there are no idle time in a variety of situations. At large processor counts, the product $\omega_m\omega_g$ must be large. This means that a weak scaling series refined only in space, but only coarsely refined in angle and energy, will eventually fail the constraints.

The optimal efficiency formula changes slightly from the KBA and hybrid KBA partitioning method in order to account for the overload factors. The only change is in the $\frac{N_{idle}}{N_{tasks}}$ term, as shown in Eq. 13.

$$\varepsilon_{opt} = \frac{1}{[1 + \frac{P_x+P_y+P_z-6}{\omega_g\omega_m\omega_r}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{13}$$

II.C. The Unstructured Transport Sweep

While PDT has scaled well out to 750,000 cores, similar levels of parallel scaling have not been achieved using unstructured sweeps yet. Pautz proposed a new list scheduling algorithm has been constructed for modest levels of parallelism (up to 126 processors)[1] .

There are three requirements for a sweep scheduling algorithm to have. First, the algorithm should have low complexity, since millions of individual tasks are swept over in a typical problem. Second, the algorithm should schedule on a set of processors that is small in comparison to the number of tasks in the sweep graph. Last, the algorithm should distribute work in the spatial dimension only, so that there is no need to communicate during the calculation of the scattering source.

Here is the pseudocode for the algorithm:

```
Assign priorities to every cell-angle pair
Place all initially ready tasks in priority queue
While (uncompleted tasks)
    For i=1,maxCellsPerStep
       Perform task at top of priority queue
       Place new on-processor tasks in queue
    Send new partition boundary data
    Receive new partition boundary data
    Place new tasks in queue
```

An important part of the algorithm above is the assigning priorities to tasks. Specialized prioritization heuristics generate partition boundary data as rapidly as possible in order to minimize the processor idle time.

Nearly linear speedups were obtained on up to 126 processors. Further work is being done for scaling to thousands of processors.

II.C.1 Cycle Detection

A cycle is a loop in a directed graph and they can occur commonly in unstructured meshes. However, they don't exist in 2D triangular extruded problems, and because our domain partitioning is convex arbitrary degenerate polygons appearing on subdomain boundaries will not produce cycles. Even though they are not applicable to this application of unstructured transport sweeps, they are discussed here for completeness.

Cycles can cause hang time in the problem, as a processor will wait for a message that might will never come. This means that the computation for one or more elements will never be completed. The solution to this is to "break" any cycles that exist by removing an edge of the task dependence graph (TDG). Old flux information is used on a particular element face in the domain. Most of the time, the edge removed is oriented obliquely with respect to the radiation direction.

Algorithms for for finding cycles are called *cycle detection* algorithms. This must be done efficiently in parallel, both because the task dependence graph is distributed, and because the finite element grid may be deforming every timestep and changing the associated TDG.

Cycle detection utilizes two operations: trim and mark. Trimming identifies and discards elements which are not in cycles. At the beginning of cycle detection, graphs are trimmed in the downwind direction, then the remaining graphs are trimmed in the upwind direction. A pivot vertex is then selected in each graph. Graph vertices are then marked as upwind, downwind, or unmarked. Then, if any vertices are both upwind and downwind, the cycle is these vertices plus the pivot vertex. An edge is removed between 2 cycle vertices, and 4 new graphs are created: a new cycle, the upwind vertices without the cycle, the downwind vertices without the cycle, and a set of unmarked vertices. This recursively continues until all cycles are eliminated.

III. Motivation and Proposed Method for Load Balancing Unstructured Meshes in PDT

The capability for PDT to generate and run on an unstructured mesh is important because it allows us to run problems without having to conform our mesh to the problem as much. When using the unstructured meshing capability in PDT, the input geometry is described by a Planar Straight Line Graph (PSLG). This is a list of vertices and the straight line segments connecting them. Before the mesh is built, the user determines how many "subsets", he/she would like to subdivide the PSLG into. The unstructured mesh is built using the Triangle Mesh Generator,[?] a 2D mesh generator. A series of cut planes in x and y are laid over the PSLG, creating an overlaid regular Cartesian grid. Each convex, orthogonal unit created by the grid is termed a "subset". Figure 4 demonstrates this functionality.

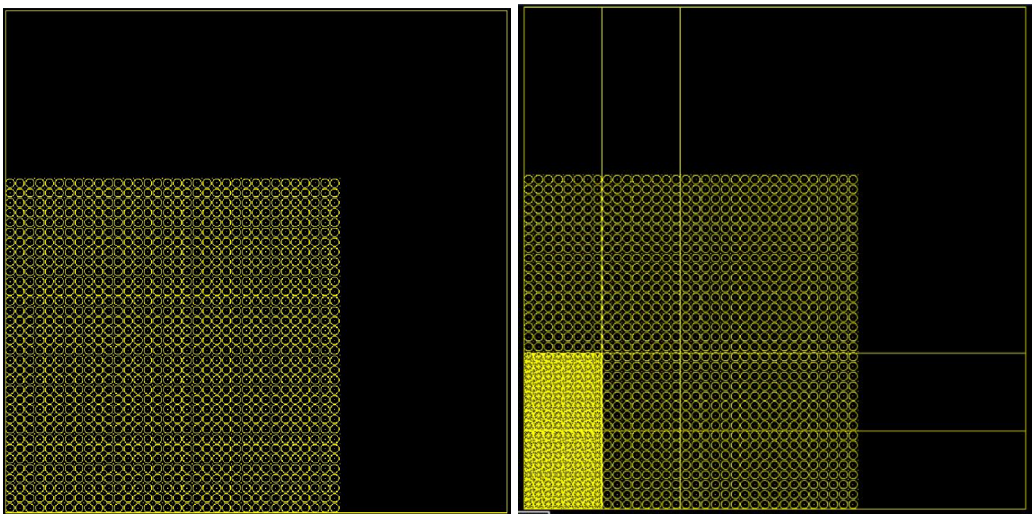

Figure 4. A PSLG describing a fuel lattice, and with a orthogonal "subset" grid imposed on on the PSLG.

This orthogonal grid is superimposed in parallel, and each "subset" is meshed using Triangle in parallel. Subsets are now the base structured unit when calculating our parallel efficiency. Discontinuities along the boundary are fixed by "stitching" hanging nodes, creating degenerate polygons along subset boundaries. Because PDT's spatial discretization employs Piece-Wise Linear Discontinuous (PWLD) finite element basis functions, there is no problem solving on degenerate polygons. A 2D mesh can extruded in the z dimension in order to give us the capability to run on 3D problems. Obviously, this is not as good as a unstructured tetrahedral mesh, but for many problems, it is a great capability to have.

When discussing the parallel scaling of transport sweeps, a load balanced problem is of great importance. A load balanced problem has an equal number of degrees of freedom per processor. Load balancing is important in order to minimize idle time for all processors by equally distributing (as much as possible) the work each processor has to do. For the purposes of unstructured meshes in PDT, we are looking to "balance" the number of cells. Ideally, each processor will be responsible for an equal number of cells.

If the number of cells in each subset can be reasonably balanced, then the problem is effectively load balanced. The Load Balance algorithm described on the next page details how the subsets will be load balanced. In summary, the procedure of the algorithm involves moving the initially user specified x and y cut planes, re-meshing, and iterating until a reasonably load balanced problem is obtained.

Load Balance: A load balancing algorithm that equalizes the number of triangles per subset.

I, J subsets specified by user
Mesh all subsets
N_{tot} = total number of triangles
N_{ij} = number of triangles in subset ij
$f = \max_{ij}(N_{ij}) / \frac{N_{tot}}{I \cdot J}$
{//Check if all subsets meet the tolerance}
if $f < \text{tol}_{\text{subset}}$ **then**
  DONE with load balancing
else
  $f_I = \max_{i}[\sum_j N_{ij}] / \frac{N_{tot}}{I}$
  $f_J = \max_{j}[\sum_i N_{ij}] / \frac{N_{tot}}{J}$
  if $f_I > \text{tol}_{\text{row}}$ **then**
    Redistribute(X_i)
  end if
  if $f_J > \text{tol}_{\text{col}}$ **then**
    Redistribute(Y_j)
  end if
  if redistribution occured **then**
    REMESH and repeat algorithm
  end if
end if
if There is still a discrepancy amongst subsets **then**
  Move cutplane segments on the subset level and remesh (may require changes to scheduling algorithm)
end if

Redistribute: A function that moves cut lines in either X or Y.

Input: CutLines (X or Y vector that stores cut lines).
Input: num_tri_row or num_tri_col, a pArray containing number of triangles in each row or column
Input: The total number of triangles in the domain, N_{tot}
stapl::array_view num_tri_view, over num_tri_row/column
stapl::array_vew offset_view
stapl::partial_sum(num_tri_view) {Perform prefix sum}
{We now have a cumulative distribution stored in offset_view}
for $i = 1$:CutLines.size()-1 **do**
  vector ¡double¿ pt1 = [CutLines(i-1), offset_view(i-1)]
  vector ¡double¿ pt2 = [CutLines(i), offset_view(i)]
  ideal_value = $i \cdot \frac{N_{tot}}{\text{CutLines.size()}-1}$
  X-intersect(pt1,pt2,ideal_value) {Calculates the X-intersect of the line formed by pt1 and pt2 and the line y = ideal_value.}
  CutLines(i) = X-intersect
end for

IV. Preliminary Results

Unstructured meshing capability has been placed in PDT, Figure coming to showcase it (probably C5G7 reactor mesh).

V. Goals and Proposed Plan

- Implment unstructured meshing capability in PDT for 2D and 2D extruded problems.
- Generate unstructured mesh in parallel using the same partitioning scheme and number of processors as the sweep.
- Perform stitching between meshed subdomains to preserve interface continuity.
- Implement load balancing algorithm for unstructured meshes in PDT. A load balanced problem would be defined by f in the pseudocode, or the subset with the maximum number of cells, divided by the average number of cells per subset.
- Verify and test code to prove load balancing algorithm effectiveness.
- Show results of the parallel transport sweeps for benchmark problems.

V.A. Possible Outlook for Research

- Propose a scaling study for unstructured meshes in PDT, similar to PDT's Zerr problem.

References

[1] Shawn D. Pautz. An algorithm for parallel sn sweeps on unstructured meshes. *Los Alamos National Laboratory Publications*, LA-UR-01-1420.