# An Approach for Load Balancing Massively Parallel Transport Sweeps on Unstructured Grids

Tarek H. Ghaddar, Jean C. Ragusa

*Dept. of Nuclear Engineering, Texas A&M University, College Station, TX, 77843-3133*
*tghaddar@tamu.edu, jean.ragusa@tamu.edu*

## INTRODUCTION

When running any massively parallel code, attaining load balancing is a priority in order to achieve the best possible parallel efficiency. A load balanced problem has an equal number of degrees of freedom per processor. Load balancing is an important factor to minimize idle time for all processors and is attained by equally distributing (as much as possible) the work load among all processors.

To the best of our knowledge, transport sweeps are the only method to scale on current and next-generation supercomputers. PDT, Texas A&M University's massively parallel deterministic transport code, has been shown to scale on logically Cartesian grids out to 750,000 cores [1]. Logically Cartesian grids are constructed with mesh cells that are identified using integer triplets $ijk$ (i.e., cubic cells), but allow for vertex motion in order to conform to curved shapes. The concepts and results presented in this paper are implemented in PDT. PDT is a solver for neutron, thermal, gamma, coupled neutron-gamma, electron, and coupled electron-photon radiation transport phenomena. It uses discrete ordinates for angular discretization, multi-group data, and discontinuous finite elements in space.

A new unstructured meshing capability was implemented in PDT in order to realistically represent certain geometries. Cut lines (planes for 3D cases) are used to partition such geometries into logically-Cartesian subdomains, which are then individually meshed in parallel using the Triangle Mesh Generator [2]. These subdomains are then "stitched" together in order to create a continuous geometry. 2D meshes can be extruded in the $z$ dimension for 3D problems.

However, unstructured meshes often create unbalanced problems due to the way localized features are meshed, so a load balancing algorithm was added into PDT.

## APPLICATION OF 2D AND 3D UNSTRUCTURED MESHES

The capability for PDT to generate and run using unstructured meshes is important because it enables more realistic problems to be tackled. However, we wish to preserve a logically-Cartesian grid at the "subset" level, with an unstructured mesh inside each subset. These logically Cartesian subdomains (subsets) are obtained using cut planes in 3D and cut lines in 2D. Figure 1 demonstrates this functionality. The geometry is decomposed into 3 subsets in $x$ and 3 in $y$, with the first two subsets meshed using Triangle.

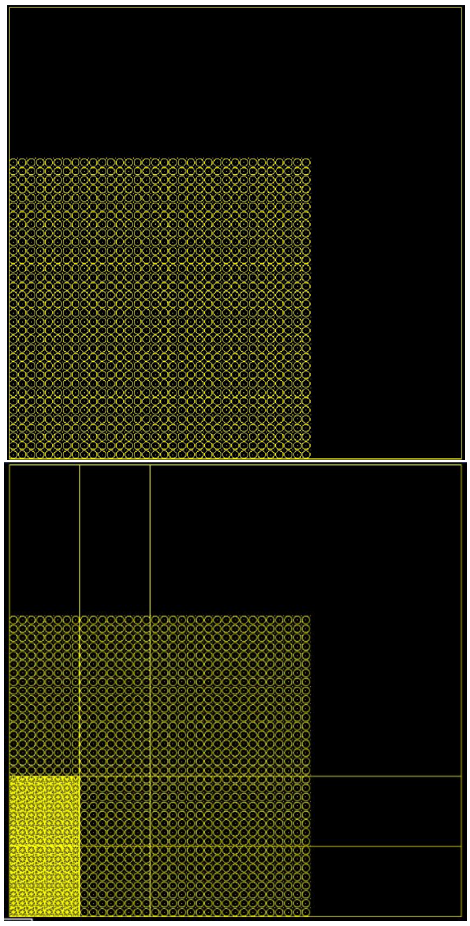

Fig. 1. A PSLG describing a fuel lattice (top image), and the "subset" grid imposed on the PSLG (bottom image, with only 2 subsets meshed for illustration purposes).

The orthogonal grid composed of the cut planes is superimposed on the geometry; each resulting subset is meshed in parallel. Subsets are now the base structured unit when performing transport sweeps. Discontinuities along the subset boundary are fixed by "stitching" hanging nodes, creating degenerate polygons along these boundaries. Because PDT's spatial discretization employs Piece-Wise Linear Discontinuous (PWLD) finite element basis functions, degenerate polygons do not lead to numerical difficulties [**?**, **?**].

Currently, only 2D and 2D extruded unstructured capability is available in PDT. The 2D input geometry is described by a Planar Straight Line Graph (PSLG). After superimposing the orthogonal grid, a PSLG is created for each subset, and meshed. The mesh can be extruded in the $z$ dimension for 3D problems. Obviously, this is not as general as a 3D unstructured tetrahedral mesh, but for many problems of interest,

it is a useful capability to have. Furthermore, this simpler setting allows for the testing of our load balancing approach for unstructured grids.

To demonstrate the newly implemented unstructured meshing capability in PDT, Texas A&M Nuclear Engineering's Impurity Model 1 (IM1) problem is used. Figure 2 showcases the 2D mesh of the IM1 problem,

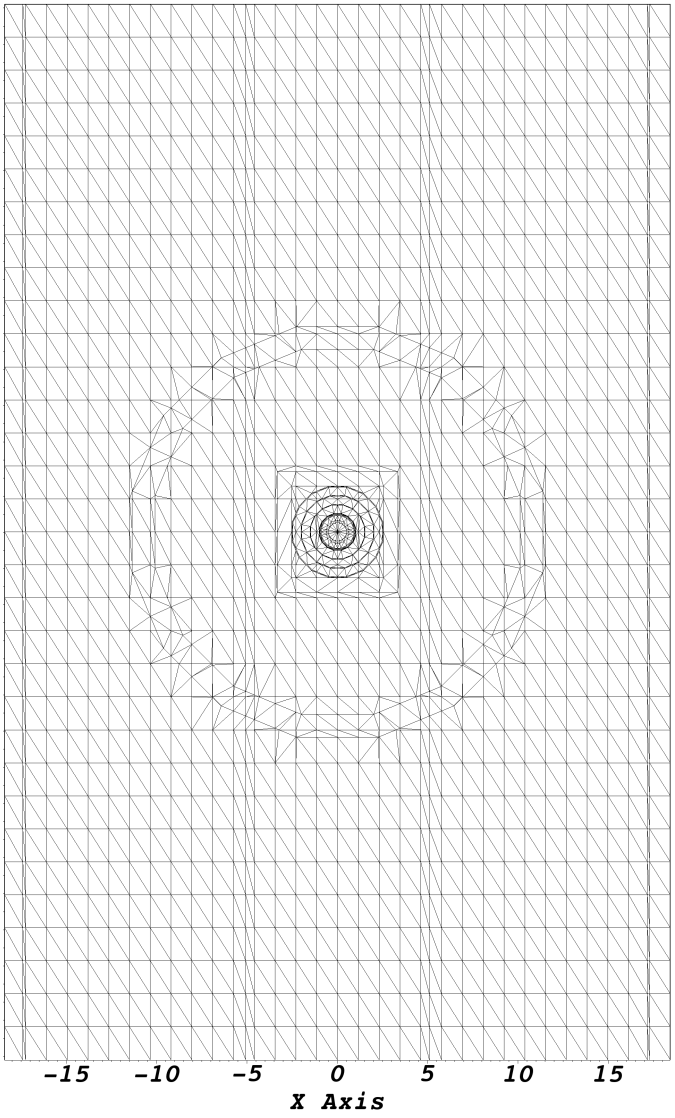


Fig. 2. The 2D mesh of the IM1 problem.

The combination of the 2D mesh and extrusion parameters yield the full 3D problem, shown in Fig. 3.

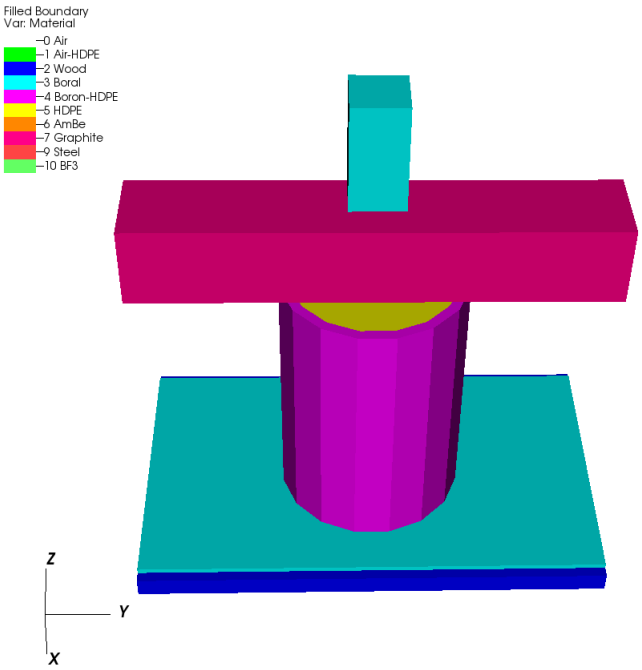


Fig. 3. The 2D extruded view of the IM1 problem.

## THE LOAD BALANCING ALGORITHM

If the number of cells in each subset can be reasonably balanced, then the problem is effectively load balanced. This has already been done using the provably-optimal transport sweep algorithm on logically Cartesian grids [1]. We extend this concept to unstructured meshes. The Load Balance algorithm described below 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. The load balancing metric, shown in Equation (1), dictates how balanced or unbalanced a problem is:

$$f = \frac{\max_{ij}(N_{ij})}{\frac{N_{tot}}{I \cdot J}}, \tag{1}$$

where $f$ is the load balance metric, $N_{ij}$ is the number of cells in subset $i$, $j$, $N_{tot}$ is the global number of cells in the problem, and $I$ and $J$ are the total number of subsets in the $x$ and $y$ direction, respectively. The metric is a measure of the maximum number of cells per subset divided by the average number of cells per subset.

The load balancing algorithm moves cut planes based on two sub-metrics, $f_I$ and $f_J$. Equation (2) defines these two parameters:

$$f_I = \frac{\max_{i}[\sum_j N_{ij}]}{\frac{N_{tot}}{I}} \tag{2a}$$

$$f_J = \frac{\max_j[\sum_i N_{ij}]}{\frac{N_{tot}}{J}}. \quad (2b)$$

$f_I$ is calculated by taking the maximum number of cells per column and dividing it by the average number of cells per column. $f_J$ is calculated by taking the maximum number of cells per row and dividing it by the average number of cells per row. If these two numbers are greater than predefined tolerances, the cut lines in the respective directions are redistributed. Once redistribution and remeshing occur, a new metric is calculated. This iterative process occurs until a maximum number of iterations is reached, or until $f$ converges within the user defined tolerance. The Load Balance algorithm behaves as follows:

```
//I,J subsets specified by user
//Check if all subsets meet the tolerance
while (f > tol_subset)
{
  //Mesh all subsets
  if (f_I > tol_column)
  {
    Redistribute(X);
  }
  if (f_J > tol_row)
  {
    Redistribute(Y);
  }

}
```

remove

**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**

## RESULTS

remove 1 or 2 and add a sentence to the effect that more examples will be provided in the full paper

The following sections will showcase the metric behavior and convergence for three test cases and the new unstructured meshing capability both in 2D and 3D.

### Test Cases for Metric Behavior and Convergence

In order to illustrate the behavior of the load balancing metric, calculated by Eq.1, three test cases are presented. Figure 4 shows the first test case, a 20 cm by 20 cm domain with two pins in opposite corners of the domain. Figure 5 shows the same size domain but with the pins on the same side.These are two theoretically very unbalanced cases, as geometrically there are two features located distantly from each other with an empty geometry throughout the rest of the domain. Figure 6 shows a lattice and reflector, which due to its denser and repeated geometry, theoretically is a more balanced problem.

A series of 162 inputs was constructed for each case. These inputs are constructed by varying the maximum triangle area from the coarsest possible to 0.01 cm$^2$. In addition, the number of subsets, $N$, varies from 2×2 to 10×10.

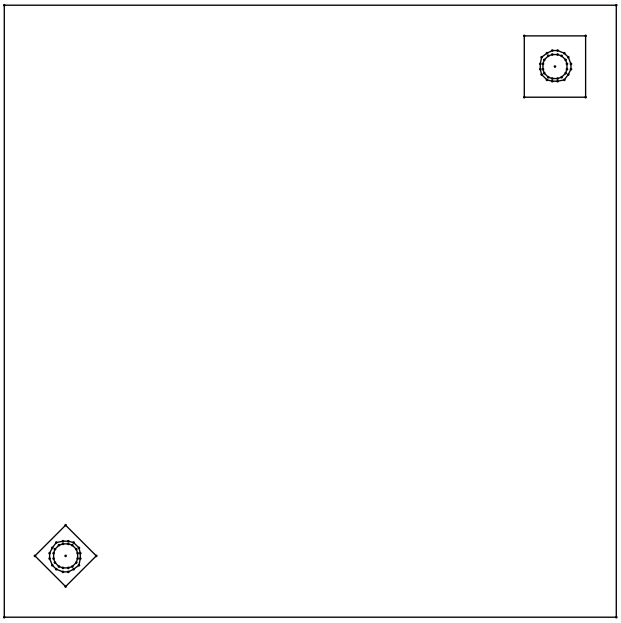

Fig. 4. The first test case used in order to test effectiveness and convergence of the load balancing metric.

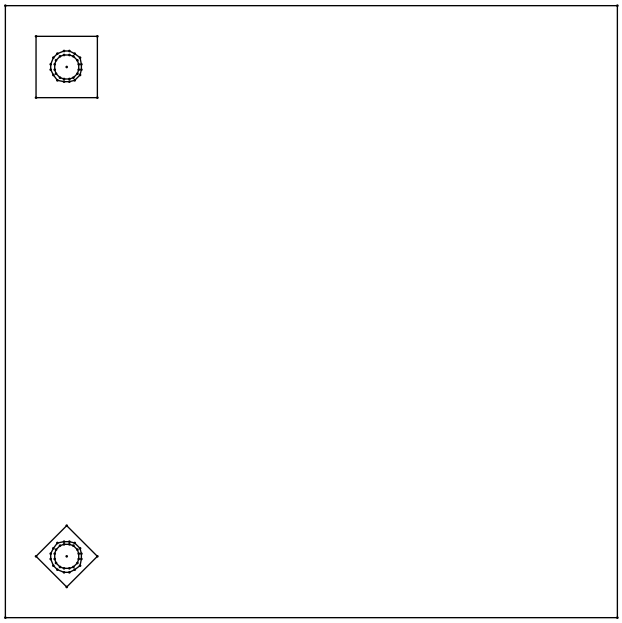

Fig. 5. The second test case used in order to test effectiveness and convergence of the load balancing metric.

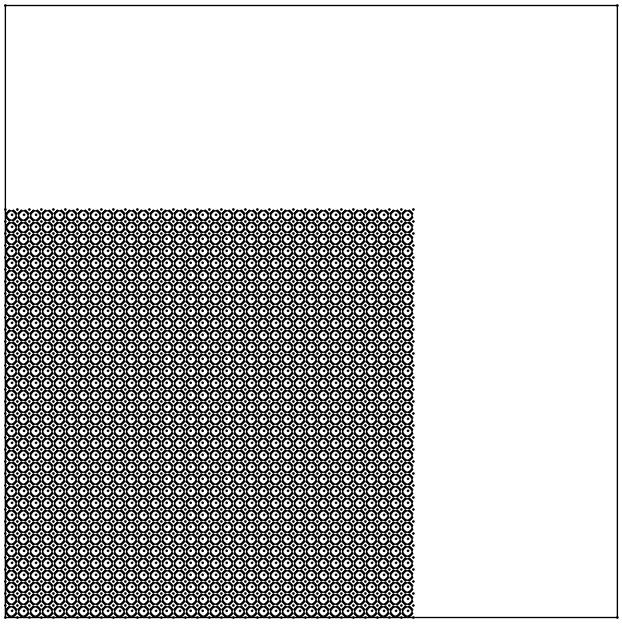

Fig. 6. The third test case used in order to test effectiveness and convergence of the load balancing metric.

### Metric Behavior and Convergence

For each test case, the 162 input inputs are run twice, once with no load balancing iterations, and once with ten load balancing iterations. The best metric is reported and recorded. Two figures for each test cases are presented below: the first figure will show the metric behavior for no iterations and the second figure will show the metric behavior for each input run with ten load balancing iterations.

Figure 7 shows the metric behavior for Fig. 4. The maximum metric value is 24.7650, and occurs when Fig. 4 is run with 8x8 subsets and a maximum triangle area of 1.6 $cm^2$. The minimum metric value is 1.0016 and occurs when Fig. 4 is run with 4x4 subsets and a maximum triangle area of 0.04 $cm^2$.

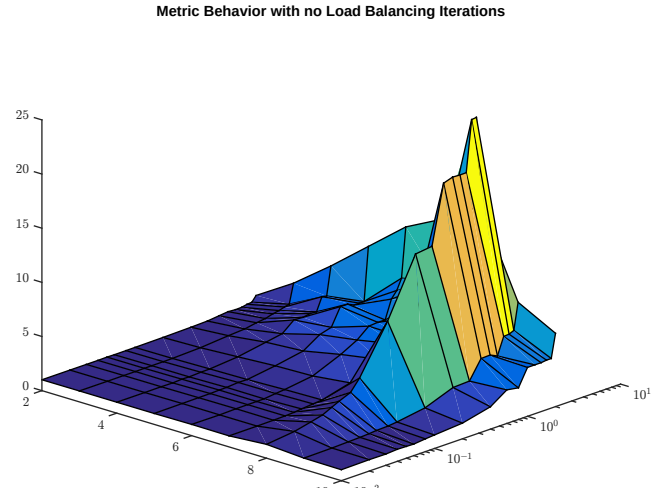


Fig. 7. The metric behavior of the first test case run with no load balancing iterations.

Figure 8 shows the metric behavior for Fig. 4 after 10 load balancing iterations. The maximum metric value is 5.0538 and occurs when Fig. 4 is run with 10x10 subsets and a maximum triangle area of 1.2 $cm^2$. The minimum metric value is 1.0017 and occurs when Fig. 4 is run with 4x4 subsets and a maximum triangle area of 0.04 $cm^2$.

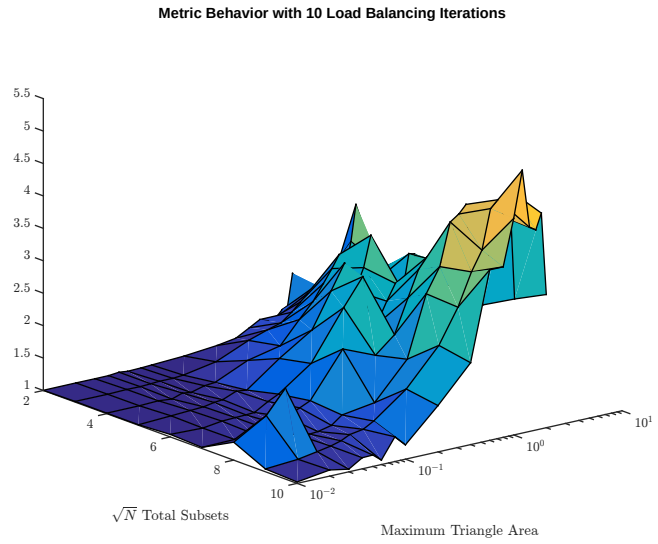


Fig. 8. The metric behavior of the first test case run with 10 load balancing iterations.

Figure 9 shows the metric behavior for Fig. 5. The maximum metric is 22.6654 and occurs when Fig. 5 is run with 8x8 subsets with a maximum triangle area of 1.8 $cm^2$. The

minimum metric is 1.0024 and occurs when Fig. 5 is run with 2x2 subsets with a maximum triangle are of 0.01 cm$^2$.

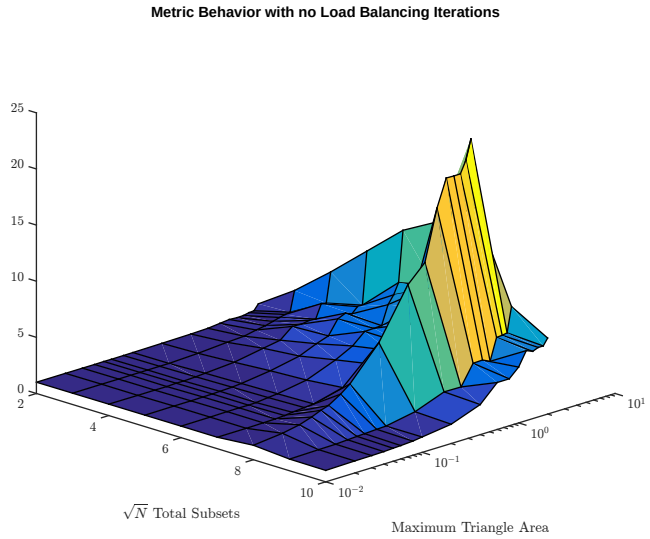


Fig. 9. The metric behavior of the second test case run with no load balancing iterations.

Figure 10 shows the metric behavior for Fig. 5 after ten load balancing iterations. The maximum metric is 3.9929 and occurs when Fig. 5 is run with 10x10 subsets with a maximum triangle area of 1.8 cm$^2$. The minimum metric is 1.0024 and occurs when Fig. 5 is run with 2x2 subsets with a maximum triangle are of 0.01 cm$^2$.

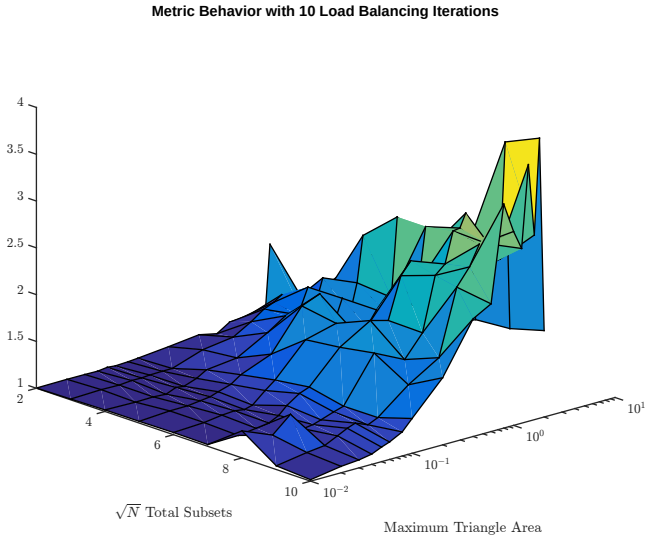


Fig. 10. The metric behavior of the second test case run with 10 load balancing iterations.

Figure 11 shows the metric behavior for Fig. 6. The maximum metric is 2.6489 and occurs when Fig. 6 is run with 10x10 subsets with a maximum triangle area of 1.8 cm$^2$. The minimum metric is 1.0179 and occurs when Fig. 6 is run with 2x2 subsets with a maximum triangle are of 0.08 cm$^2$.

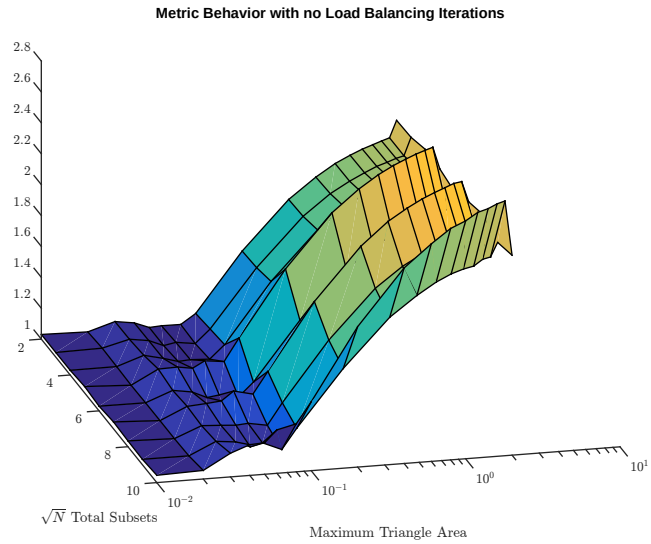


Fig. 11. The difference in metric behavior of the third test case with no load balancing iterations.

Figure 12 shows the metric behavior for Fig. 6 after ten load balancing iterations. The maximum metric is 2.2660 and occurs when Fig. 6 is run with 10x10 subsets with a maximum triangle area of 0.4 cm$^2$. The minimum metric is 1.0021 and occurs when Fig. 6 is run with 2x2 subsets with the Triangle's coarsest possible mesh.

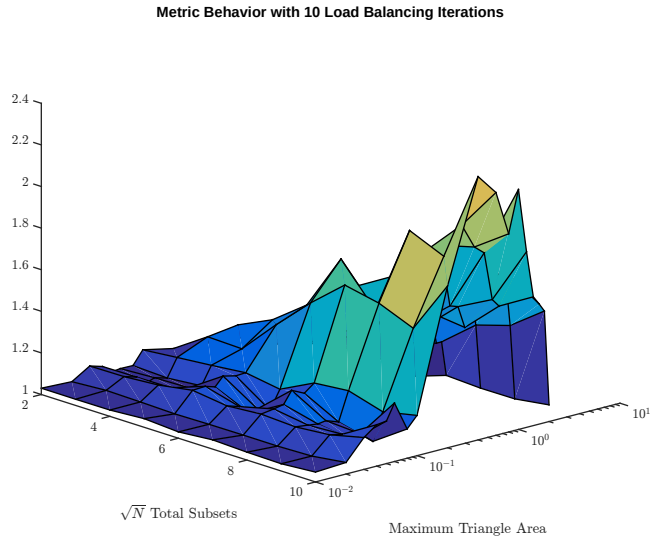


Fig. 12. The difference in metric behavior of the third test case after ten load balancing iterations.

Because Fig. 6 has more features and is more symmetric of a problem, the initial load balancing metric will not be as large as the load balancing metric of Figs. 5 and 4. As a result, the improvement in the load balancing metric after 10 iterations will not be as great in problems similar to Fig. 6.

Good improvement is seen throughout all three test cases for all three inputs, particularly the first two test cases, which were initially very unbalanced. However, there were many inputs run that had problems with $f > 1.1$, which means many problems were unbalanced by more than 10%. The user will not always have the luxury of choosing the number of subsets they want the problem run with, as this directly affects the number of processors the problem will be run with. Certain problems will require more processors and will require minimizing the total number of cells in the domain for the problem to complete running in a reasonable amount of time. As a result, improvements to the algorithm must be made.

This can be done by changing how the cut lines are redistributed. Instead of changing entire row and column widths, the cut lines can be moved on the subset level. However, this can sacrifice the strict orthogonality that PDT currently utilizes to scale so well on a massively parallel scale. Changes to the performance model and the scheduler will possibly have to be made.

Another option is to implement domain overloading, which is the logical extension of the work presented in this paper. This would involve processors owning different numbers of subsets, with no restriction on these subsets being contiguous. This would be the most effective method at perfecting this algorithm, and would lead to less problems being unbalanced by more than 10%.

## CONCLUSIONS

We have proposed and tested a load-balance algorithm for radiation transport problems on unstructured grids solved using sweeps. The load balancing algorithm outlined in this paper performs satisfactorily for feature-heavy problems and even for some particularly sparse problems. Its effectiveness depends on the maximum triangle area used and the number of subsets chosen to decompose the problem domain.

Load imbalance reduction is observed throughout all three test cases for all three inputs, particularly the first two test cases, which were initially very unbalanced. However, there were many inputs run that had problems with $f > 1.1$, which means many problems were unbalanced by more than 10% after application of the load balancing algorithm. Further improvement to the method will include: investigation of domain-overloading [1] and load-balancing by dimension.

## ACKNOWLEDGMENTS

## REFERENCES


1. M. A. ET AL, "Provably Optimal Parallel Transport Sweeps with Non-Contiguous Partitions," in "ANS MC2015-Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method," (2015).
2. J. SHEWCHUK, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in M. C. LIN and D. MANOCHA, editors, "Applied Computational Geometry: Towards Geometric Engineering," Springer-Verlag, *Lecture Notes in Computer Science*, vol. 1148, pp. 203–222 (May 1996), from the First ACM Workshop on Applied Computational Geometry.