M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, April 16-20, 2017, on USB (2017)

Transport Sweeps Using an Improved Slice Balance Approach with LDFE and GPU Acceleration

Richard M. Vega, Marvin L. Adams

Texas A&M University, Department of Nuclear Engineering
richard_vega@tamu.edu, mladams@tamu.edu

Abstract - *This paper presents an improvement to the traditional Slice Balance Approach (SBA) that is more accurate and allows for a new method of parallelization of discrete ordinates transport sweeps. The accuracy of the new approach is compared to the traditional cell balance and slice balance methods for both the diamond difference (DD) and linear discontinuous finite element (LDFE) spatial discretization schemes. The results show that the alterations made to the traditional SBA reduce numerical diffusion of the solution in the vicinity of discontinuities for both DD and LDFE. A manufactured solution is used to confirm that the second order convergence rate of the LDFE spatial discretization scheme is retained for a smooth solution when the improved SBA is applied. The new method of parallelization exhibits no processor idle time and allows for an arbitrary domain decomposition scheme, with no restriction on the smoothness of the inter-node domain boundaries. These advantages are gained at the cost of higher communication and memory demand. The added communication is coalesced into a small number of messages, and the added memory requirement is that the entire mesh (not the entire solution) be duplicated on each node. This added memory requirement is roughly 320 bytes per spatial cell in the mesh. A weak scaling study is performed to characterize the parallel efficiency of the proposed method. Furthermore, through the use of graphics processing units (GPUs) the time required to compute geometric quantities on a per-slice basis, which cannot be pre-computed and stored due to excessive memory requirements, can be significantly reduced, and perhaps even hidden altogether by pipe-lining the geometric setup and transport sweep routines on the GPU and CPU respectively.*

I. INTRODUCTION

The SBA was introduced by Grove [1] as a multiple balance method, as defined by Morel and Larsen [2], for solving the discrete ordinates (S_N) equations on arbitrary unstructured meshes. It is an inherently face-based method whereby each cell of the spatial mesh is decomposed into slices, each of which intersects a single inlet and single outlet face (or edge in 2-D) for a given ordinate. The streaming-plus-collision operator is inverted independently on each slice for each direction, so the slices can be viewed as a mesh that is finer than the cell-wise mesh.

Besides the accuracy gained by the increased spatial resolution, SBA gains accuracy by a more consistent representation of particle streaming in a way reminiscent of the Method of Characteristics (MOC). Consider the 2-D example of a sliced quadrilateral cell shown in Figure 1. In a cell balance method, the flux in the cell interior and the fluxes exiting on the right and top edges would all be influenced by the fluxes entering on the left and bottom edges. However, the flux entering on the left edge should not influence the flux exiting on the right edge. With the SBA, this non-physical causality is avoided.

The SBA solves for the interior and exiting fluxes of each slice using a prescribed balance method such as a finite element method, and uses these slice-wise fluxes to construct the interior and exiting cell-wise fluxes. In the example shown in Figure 1, the cell interior flux would be constructed from the interior fluxes of the three slices, the flux exiting the cell on the top face would be constructed from the exiting fluxes of Slices 1 and 2, and the flux exiting the cell on the right face would be the exiting flux of Slice 3.

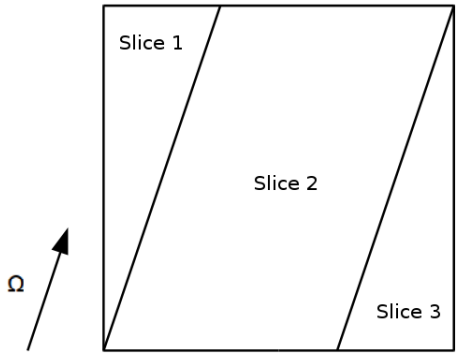

Fig. 1. Example of a sliced quadrilateral cell.

On a per-cell basis, the non-physical causality noted above is avoided by the SBA. However, on a per-face basis it is not. To see why, consider an identical cell to that in Figure 1 placed on top of the depicted cell, composed of Slices 1', 2', and 3' as shown in Figure 2, ignoring the dotted line for the moment. In this new cell, the flux entering on the bottom face is given by the flux exiting the top face of the original cell, which was constructed from Slices 1 and 2. In the SBA, this flux would be used as the incoming fluxes for Slices 2' and 3'. Physically however, the flux leaving Slice 1 should not influence the flux entering Slice 3'.

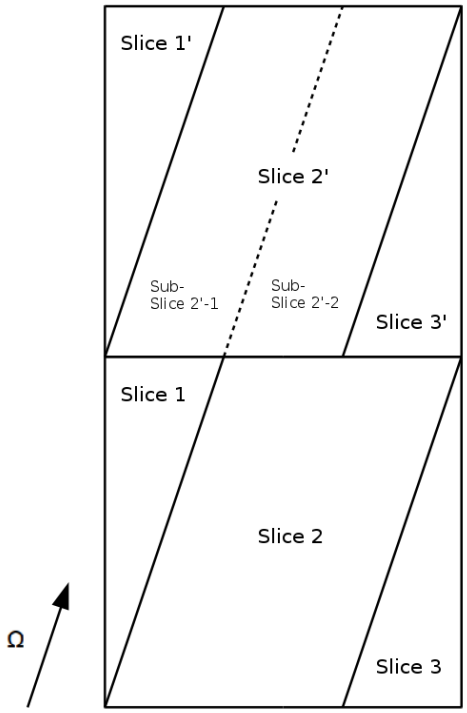

Fig. 2. Example of two adjacent cells illustrating the concept of a sub-slice.

It is this smearing of the facial fluxes that we attempt to reduce in the proposed method by exploiting the concept of a sub-slice. Where a slice is defined as the cell region bounded by a single inlet-outlet face pair, a sub-slice is defined as the portion of a slice that is downstream of a single slice in the upstream cell, as shown in Figure 2. While the balance method is still applied to each slice, and the cell interior flux is still constructed from the slice values, the cell facial fluxes are not. The incoming fluxes are stored on the sub-slices, and the slice can be treated only after all of its contained sub-slices have received their incoming flux information from their upstream slices. At this point the sub-slice incoming fluxes are appropriately averaged for use in the parent slice.

The careful observer will note that this alteration to the SBA only serves to propagate discontinuities into the cell immediately downstream, and further non-physical causalities are indeed still present. For instance, placing a third cell on top of the cells in Figure 2 composed of Slices 1", 2", and 3", we can see that the flux entering Sub-Slice 2"-2 would be determined by the flux exiting Slice 2', which was influenced by the exiting fluxes of Slices 1 and 2. However, the flux exiting Slice 2 should not influence the incoming flux in Sub-Slice 2"-2. It should also be noted that propagating discontinuities throughout the entire mesh in this fashion would result in a more computationally cumbersome beast than either the MOC or S_N methods. One goal of this research is to determine what effect this single cell downstream discontinuity propagation has on the numerical solution with relatively little added cost to the traditional SBA. This will be examined by the propagation of a single ray in 2-D in section III.

The alterations made above to the SBA also lead to the possibility of a new strategy for the parallelization of the S_N transport sweep. Current S_N transport sweep parallelization strategies involve spatial domain decomposition across shared memory nodes, where the inter-node domain boundaries are usually kept planar to avoid re-entry of a ray in the discrete ordinate direction, and each node owns the mesh and solution on its portion of the spatial domain. A node on the boundary then performs a sweep within its domain, and communicates the outgoing fluxes to the domains downstream in the direction of the given ordinate. At this point the original node is free to start the sweep for the next ordinate, while the downstream nodes start the sweep on the previous ordinate. In this way, communication is minimized, and memory demand is reduced as much as possible. This comes at the cost of frequent communication, even if the total amount of data communicated is minimized, and more importantly idle time as nodes in the interior of the domain wait for data at the start of the problem, and are idle again at the end of the problem as the sweep reaches the domain boundary for the final outward-directed angles.

To illustrate a parallelization strategy that exhibits zero idle time, and is made possible by the improved SBA discussed above, consider the cubic domain meshed by arbitrary polyhedra shown in Figure 3. If we were to draw "cut planes" through the mesh as shown in Figure 4, and further refine the definition of a slice such that no slice straddles a cut plane, the transport sweep within each region bounded by consecutive cut planes is now independent of the sweeps being performed in all other such regions. These cut planes divide the domain into we will refer to as pipes. In the example shown in Figure 4, the sweep in each pipe would be performed by a single node, without any communication occurring during the sweep. Furthermore, since the sweep in each angle is independent of the sweep in any other angle, several angles can be swept simultaneously, each with its own pipe decomposition. For instance, if there were 16 nodes, 2 angles could be swept simultaneously, each with a decomposition consisting of 8 pipes.

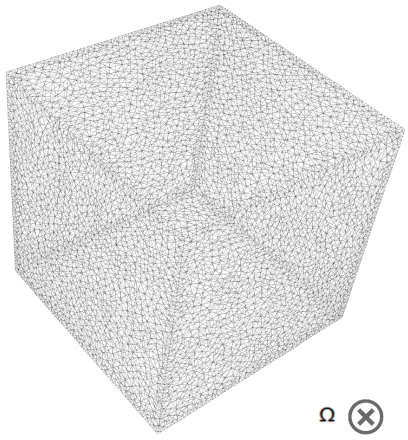

Fig. 3. A cubic domain meshed by arbitrary polyhedral cells viewed from upstream in the discrete ordinate direction.

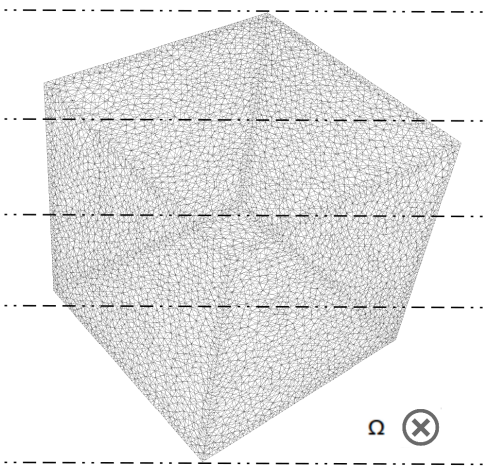

Fig. 4. Pipe decomposition of the domain depicted in Figure 3, defined by five bounding planes parallel to $\mathbf{\Omega}$ and perpendicular to the page.

While no communication is necessary during each sweep, communication before and after the sweep is necessary. With an arbitrary domain decomposition where each node is assigned a region of the mesh for which to store the solution, each node must communicate the source moments as well as the total cross section in each of its local cells to the nodes whose pipes contain these cells. The nodes containing these cells in their pipes will have to communicate the contribution to the solution for these cells back to the node that owns these cells when the sweep is completed.

In this way, volumetric information is communicated between nodes instead of boundary information, resulting in larger messages, but fewer of them. This minimizes the cost of message passing latency which is typically orders of magnitude higher than the cost of per byte communication. The benefit of using an arbitrary domain decomposition scheme to assign cells to nodes for storage of the solution is easy to see for anyone who has tried decomposing an unstructured mesh into regions with flat boundaries. Such regions may not exist naturally in most problems of interest, and such a restriction can be even more difficult to implement given that load balancing is important.

One notable drawback to this parallelization strategy is that the entire mesh geometry must be known by every node in order to construct the slices and sub-slices within its pipe. The mesh can be described by the point coordinates, point indices on each face, face indices on each cell, cell centroids and volumes, and face normal vectors. In the limit where this becomes prohibitive, a hybrid method can be used whereby large portions of a mesh are stored on a subset of nodes, with interface fluxes communicated as needed. This will be addressed in a future communication.

Another difficulty is presented when applying a finite element method to each slice, although this difficulty is present in the traditional SBA as well. Finite element methods require various volume and area moments of the spatial cell as matrix coefficients. In a cell balance method, this information is typically computed before the simulation begins, and stored in memory for re-use during each sweep. The same information would be required for each slice in a slice balance method, however there is now a different, higher resolution mesh for every angle in the S_N quadrature set. Storage of geometric quantities for every slice resulting from every angle becomes prohibitive, and hence these quantities must be re-computed each time the domain is swept. While it seems counterintuitive to compute the same quantities hundreds or even thousands of times, if the time to compute these quantities can be reduced to a small fraction of the overall sweep time, re-computing them may be preferable.

II. THEORY

While the improved SBA presented here has been applied to both the diamond difference and LDFE spatial discretization schemes in 2-D, the origin of this research was the application of the LDFE scheme with the traditional SBA in 3-D. The mathematics involved in such an endeavor were presented by Kennedy, Watson, and Grove[3], however no results were presented, and to the authors' knowledge, no such implementation exists. The alterations made to the SBA mentioned above were devised during such an implementation. For these reasons, and due to the fact that diamond difference implementations are relatively straight-forward, we will focus primarily on the LDFE scheme with the improved SBA in 3-D. We begin by writing the energy-independent, steady-state, fixed source transport equation for angle m and slice s as

$$\mathbf{\Omega}_m \cdot \boldsymbol{\nabla}\psi_{m,s}(r) + \sigma_{t,s}\,\psi_{m,s}(r) = q_{m,s}(r)\,. \tag{1}$$

We then expand the angular flux $\psi_{m,s}(r)$ in the linear discontinuous basis functions $\{b_i^s(r)\}_{i=c,x,y,z}$

$$\psi_{m,s}(r) = \sum_{i=c,x,y,z} b_i^s(r)\,\psi_{m,s}^i\,, \tag{2}$$

where

$$b_c^s(r) = 1; \quad b_x^s(r) = \frac{2\,(x-\bar{x}_s)}{\Delta x_s};$$

$$b_y^s(r) = \frac{2\,(y-\bar{y}_s)}{\Delta y_s}; \quad b_z^s(r) = \frac{2\,(z-\bar{z}_s)}{\Delta z_s}\,.$$

We then multiply equation 1 by a set of weight functions $\{\omega_j^s(r)\}_{j=c,x,y,z}$, spanning the same linear space as the basis functions above

$$\omega_c^s(r) = 1; \quad \omega_x^s(r) = 3\,\frac{2\,(x-\bar{x}_s)}{\Delta x_s};$$

$$\omega_y^s(r) = 3\,\frac{2\,(y-\bar{y}_s)}{\Delta y_s}; \quad \omega_z^s(r) = 3\,\frac{2\,(z-\bar{z}_s)}{\Delta z_s}\,.$$

We then integrate over the volume of the slice to arrive at the following system of equations for each slice

$$\oiint_{\partial V_s} \omega_j^s(r)\,\psi_{m,s}(r)\,(\boldsymbol{\Omega}_m\cdot\boldsymbol{n})\;d^2r + \sum_{i=c,x,y,z}\iiint_{V_s}\Big(-\boldsymbol{\Omega}_m\cdot\boldsymbol{\nabla}\omega_j^s(r)\,b_i^s(r)\,\psi_{m,s}^i + \sigma_{t,s}\,\omega_j^s(r)\,b_i^s(r)\,\psi_{m,s}^i\Big)\,d^3r = \iiint_{V_s}\omega_j^s(r)\,q_{m,s}(r)\,d^3r\,. \quad (3)$$

Note that $\boldsymbol{\Omega}_m\cdot\boldsymbol{n}=0$ on all slice faces except for the slice inlet and outlet, and hence the first term reduces to the sum of integrals over these two faces. The integrals over the inlet face can be performed while sweeping since we are using upwind values for the incoming flux variables. At this point, there are four equations, and eight unknowns; four from the expansion coefficients $\psi_{m,s}^i$, and four from the outlet integrals of the angular flux. We close the system by multiplying equation 2 by each weight function ω_j^s, and integrating over the outlet face of the slice

$$\iint_{\partial V_{s,out}}\omega_j^s(r)\,\psi_{m,s}(r)\,d^2r = \sum_{i=c,x,y,z}\psi_{m,s}^i\iint_{\partial V_{s,out}}\omega_j^s(r)\,b_i^s(r)\,d^2r\,. \quad (4)$$

The next step is to represent the angular flux in the cell $\psi_{m,c}(r)$, given the angular flux in the contained slices. To do this, the angular flux in the cell is expanded in a basis function set for the cell $\{b_i^c(r)\}_{i=c,x,y,z}$

$$\psi_{m,c}(r) = \sum_{i=c,x,y,z} b_i^c(r)\,\psi_{m,c}^i\,, \quad (5)$$

where the basis functions are the same as before with the centroid coordinates $(\bar{x},\bar{y},\bar{z})$ and the extents $(\Delta x,\Delta y,\Delta z)$ evaluated for the cell instead of for the slice. Weight functions for the cell $\{\omega_j^c(r)\}_{j=c,x,y,z}$, are defined similarly. To determine the expansion coefficients for the angular flux in the cell $\psi_{m,c}^i$, we multiply equation 5 by the cell weight functions and integrate over the cell volume

$$\iiint_{V_c}\omega_j^c(r)\,\psi_{m,c}(r)\,d^3r = \sum_{i=c,x,y,z}\psi_{m,c}^i\iiint_{V_c}\omega_j^c(r)\,b_i^c(r)\,d^3r\,. \quad (6)$$

This results in a system of four equations and four unknowns for each cell, and the left hand side of each equation must be accumulated during the sweep. For the constant weight function, this is a relatively simple procedure given by

$$\iiint_{V_c}\psi_{m,c}(r)\,d^3r = \sum_s\iiint_{V_s}\psi_{m,s}(r)\,d^3r = \sum_s V_s\,\psi_{m,s}^c\,, \quad (7)$$

since volume integrals of the non-constant basis functions are zero. For the other weight functions, things are only slightly more complicated. Consider for example the ω_x^c case

$$\iiint_{V_c}\omega_x^c(r)\,\psi_{m,c}(r)\,d^3r = \sum_s\iiint_{V_s}\omega_x^c(r)\,\psi_{m,s}(r)\,d^3r\,. \quad (8)$$

The cell weight function can be expressed in terms of the slice weight functions as

$$\omega_x^c(r) = 3\,\frac{2\,(x-\bar{x}_c)}{\Delta x_c} = 3\,\frac{2\,(x-\bar{x}_s+\bar{x}_s-\bar{x}_c)}{\Delta x_c\,\frac{\Delta x_s}{\Delta x_s}} = \frac{\Delta x_s}{\Delta x_c}\left(\omega_x^s(r) + 3\,\frac{2\,(\bar{x}_s-\bar{x}_c)}{\Delta x_s}\right), \quad (9)$$

and thus we can represent the integral as

$$\iiint_V\omega_x^c(r)\,\psi_{m,c}(r)\,d^3r = \sum_s\frac{\Delta x_s}{\Delta x_c}\left(\iiint_{V_s}\omega_x^s(r)\,\psi_{m,s}(r)\,d^3r + 3\,\frac{2\,(\bar{x}_s-\bar{x}_c)}{\Delta x_s}\,V_s\,\psi_{m,s}^c\right), \quad (10)$$

which after expansion of $\psi_{m,s}(r)$, replaces the last remaining integral with a sum whose coefficients have already been calculated as coefficients in equation 3.

For steady state solutions, we do not need to store the angular flux variables, and instead we can use quadrature integration to write the scalar flux as

$$\phi_c(r) = \sum_m w_m\psi_{m,c}(r) \quad (11)$$

which allows us to use

$$\iiint_{V_c}\omega_j^c(r)\,\phi_c(r)\,d^3r = \sum_{i=c,x,y,z}\phi_c^i\iiint_{V_c}\omega_j^c(r)\,b_i^c(r)\,d^3r\,. \quad (12)$$

instead of equation 6 to evaluate the scalar flux moments, where the left hand sides are given by

$$\iiint_{V_c}\phi_c(r)\,d^3r = \sum_m\sum_s V_s\,w_m\psi_{m,s}^c\,, \quad (13)$$

and

$$\iiint_V\omega_x^c(r)\,\phi_c(r)\,d^3r = \sum_m w_m\sum_s\frac{\Delta x_s}{\Delta x_c}\left(\iiint_{V_s}\omega_x^s(r)\,\psi_{m,s}(r)\,d^3r + 3\,\frac{2\,(\bar{x}_s-\bar{x}_c)}{\Delta x_s}\,V_s\,\psi_{m,s}^c\right), \quad (14)$$

for the ω_c^c and ω_x^c cases respectively. These summations are accumulated throughout the sweep, and the system of four equations is inverted for each cell after each inner iteration. If anisotropic scattering is present, equations for moments can be accumulated much the same way.

To determine the weighted integral of the incoming flux on the downstream sub-slice, labeled $s*$, the coefficients of the linear expansion of the angular flux in the slice are passed to the sub-slice, at which point we can compute the integrals

$$\iint\limits_{\partial V_{s*,in}} \omega_j^{s*}(r)\,\psi_{m,s}(r)\,d^2r\,, \tag{15}$$

where ω_j^{s*} is similar to ω_j^s with the centroid coordinates and extents evaluated for the sub-slice instead of for the slice. These sub-slice inlet integrals are then combined to give the slice inlet integrals in a similar manner as slice volume integrals were combined to give cell volume integrals.

Implementation of the parallelization method described in the previous section proceeds as a loop over sets of angles. A "set" is the collection of angles chosen to sweep simultaneously, which must be evenly divisible into the number of nodes available to the simulation for perfect load-balancing. Several data structures are also needed, and are assembled and stored prior to sweeping. These include the following:

- A data structure that returns a unique node index for each angle-pipe index pair, copied on each node.
- A data structure that returns a list of pipe indices that a given cell contributes to, given the local cell-angle index pair, unique to each node.
- A data structure that returns the angle-pipe index pair that this node is responsible for, for a given angle set index, unique to each node.
- A data structure that returns a list of nodes that each node must send data to for a given angle set index, unique to each node.
- A data structure that returns a list of nodes that each node must receive data from for a given angle set index, unique to each node.
- A data structure that returns the node index and local cell index given a global cell index, copied to each node.

A single inner iteration composed of a sweep for each angle in the quadrature set is described by Algorithm 1. The outer loop over angle sets begins with each node computing the source moments due to both scattering and inhomogeneous sources for each angle in the angle set and for each of its local cells on which it stores the solution. These values are then packaged into a struct along with the global cell index and the cell set index (used here to assign material properties to all cells in a given cell set), and then sends these structs for each cell-angle pair to the node that owns the pipe that this cell belongs to for the given angle.

Algorithm 1 Inner iteration.

e = energy group index
for $a = 0$ to $N_{\text{angle sets}} - 1$ **do**
  for $i = 0$ to $N_{\text{local cells}} - 1$ **do**
    for $j = 0$ to $N_{\text{angles per angle set}} - 1$ **do**
      $m = a \times N_{\text{angles per angle set}} + j$
      q^a, q^x, q^y, q^z = ComputeSource(i, m)
      g = GlobalCellIndex(i)
      s = CellSetIndex(i)
      Source$[i][j]$ = struct$(q^a, q^x, q^y, q^z, g, s)$
    end for
  end for
  for $i = 0$ to $N_{\text{local cells}} - 1$ **do**
    for $j = 0$ to $N_{\text{angles per angle set}} - 1$ **do**
      $m = a \times N_{\text{angles per angle set}} + j$
      for $k = 0$ to $N_{\text{pipes per cell}}[i][m] - 1$ **do**
        p = PipeIndices$[i][m][k]$
        n = NodeIndices$[m][p]$
        SendSource$[n]$.push_back(Source$[i][j]$)
      end for
    end for
  end for
  Send arrays of Source structs.
  Receive structs into array mySources[].
  for $i = 0$ to $N_{\text{received sources}} - 1$ **do**
    for $j = 0$ to $(P+1)^2 - 1$ **do**
      g = mySources.g
      $Y^m_{l,\text{index}} = j$
      $\phi^a_{\text{sum}} = \phi^x_{\text{sum}} = \phi^y_{\text{sum}} = \phi^z_{\text{sum}} = 0$
      Solutions.push_back(
      struct$(g, Y^m_{l,\text{index}}, \phi^a_{\text{sum}}, \phi^x_{\text{sum}}, \phi^y_{\text{sum}}, \phi^z_{\text{sum}})$)
    end for
  end for
  Build the slices and sub-slices and perform the sweep, accumulating the contributions to the sums in equations 13 and 14 into the $\phi^a_{\text{sum}}, \phi^x_{\text{sum}}, \phi^y_{\text{sum}}$, and ϕ^z_{sum} components of the Solutions structs.
  for $i = 0$ to Solutions.size() $- 1$ **do**
    g = Solutions$[i]$.g
    n = GlobToLocNodeInd$[g]$
    SendSolutions$[n]$.push_back(Solutions$[i]$)
  end for
  Send arrays of Solutions structs.
  Receive structs into an array mySolutions[].
  for $i = 0$ to mySolutions.size() $- 1$ **do**
    g = mySolutions$[i]$.g
    l = GlobToLocLocalInd$[g]$
    $Y^m_{l,\text{index}}$ = mySolutions$[i]$.$Y^m_{l,\text{index}}$
    $\phi^a_{\text{sum}}[l][Y^m_{l,\text{index}}][e]$ += mySolutions.ϕ^a_{sum}
    $\phi^x_{\text{sum}}[l][Y^m_{l,\text{index}}][e]$ += mySolutions.ϕ^x_{sum}
    $\phi^y_{\text{sum}}[l][Y^m_{l,\text{index}}][e]$ += mySolutions.ϕ^y_{sum}
    $\phi^z_{\text{sum}}[l][Y^m_{l,\text{index}}][e]$ += mySolutions.ϕ^z_{sum}
  end for
end for

Once each node has received an array of these structs, it knows which cells from the global mesh are in its pipe, as well as the volumetric source strength and total cross section (using the cell set index) within each one. The node then builds another array of structs on which to store the contribution to the sums in equations 13 and 14, which it will send back to the node that it received the cell from after it has performed a sweep through its pipe. Before performing this sweep however, it must construct the slices and sub-slices that are contained in its pipe. This is performed in the following seven stages.

1. Count slices: counts the number of slices in the pipe and builds a list of inlet and outlet face pairs that define each slice.
2. Build slice bases: assigns to each slice its global cell index, inlet and outlet face indices, source moments, and total cross section.
3. Slice integration: calculates the centroid, extents, volume integrals, and outlet face integrals which appear in equations 3 and 4 for each slice.
4. Count sub-slices: counts the number of sub-slices in the pipe, and builds a list of outlet faces for the downstream cell that form a sub-slice with the slice's outlet face.
5. Build sub-slice bases: assigns to each sub-slice its cell, inlet face, outlet face, and upstream slice indices.
6. Sub-slice integration: calculates the centroid, extents, and inlet face integrals for each sub-slice.
7. Assign downstream and contained: assigns to each slice its downstream sub-slice indices as well as the indices of all sub-slices contained within each slice.

The sweep algorithm proceeds in an outer loop over stages, with a maximum stage count limit set sufficiently high to complete the boundary-to-boundary sweep. On the first stage, a loop over the slices in the pipe picks out the slices whose inlet faces are on the boundary with $\mathbf{\Omega}_m \cdot \boldsymbol{n} < 0$. These slices are assigned inlet flux values from the boundary conditions, their indices are pushed in a queue containing all the slices in the current stage that are ready to be solved, and their boolean member to indicate that they have already been solved is set to true. Next, a loop over the ready queue solves equations 3 for each slice in the queue and updates the summations in equations 13 and 14. A second loop over the ready queue then picks out each slice, and loops over the sub-slices downstream of it. For each sub-slice, the number of sub-slices pending solution in the slice containing the sub-slice is incremented, and the expansion coefficients for the upstream slice are used to calculate the incoming flux moments on the sub-slice. The ready queue is then zeroed out. A final loop over the slices checks to see if the number of sub-slices contained within each slice pending solution is equal to the number of sub-slices contained within the slice and that the slice has not yet been solved. If these conditions are satisfied, the slice is placed in the ready queue, its boolean member indicating that it has been solved is set to true, and its incoming flux moments are calculated by appropriately transforming the incoming flux moments of its contained sub-slices. Finally, if the number of slices in the ready queue is zero, this indicates that the sweep has completed, and the outer stage loop is exited.

Algorithm 2 S_N transport sweep.

```
m = angle index
for k = 1 to MAX_STAGE_COUNT do
    if k = 1 then
        for s = 0 to N_slices − 1 do
            i = slices[s].inletFace
            if isOnIncomingBoundary(i) then
                ψ^c_{m,s,in}, ψ^x_{m,s,in}, ψ^y_{m,s,in}, ψ^z_{m,s,in} =
                        BoundaryConditions(i)
                Ready.push_back(s)
                slices[s].done = true
            end if
        end for
    end if
    for l = 0 to N_Ready − 1 do
        s = Ready[l]
        ψ^c_{m,s}, ψ^x_{m,s}, ψ^y_{m,s}, ψ^z_{m,s} = Solve(s)
        UpdateScalarFluxSums(ψ^c_{m,s}, ψ^x_{m,s}, ψ^y_{m,s}, ψ^z_{m,s})
    end for
    for l = 0 to N_Ready − 1 do
        s = Ready[l]
        for i = 0 to N_sub-slices downstream of slice s − 1 do
            s* = slices[s].downstream[i]
            p = slice index that s* is contained in
            NumPendingContained[p] += 1
            ψ^c_{m,s*,in}, ψ^x_{m,s*,in}, ψ^y_{m,s*,in}, ψ^z_{m,s*,in} =
                CalculateInletFluxes(ψ^c_{m,s}, ψ^x_{m,s}, ψ^y_{m,s}, ψ^z_{m,s})
        end for
    end for
    Ready.resize(0)
    for s = 0 to N_slices − 1 do
        if (NumPendingContained[s] = NumContained[s])
            and slices[s].done = false) then
            ψ^c_{m,s,in}, ψ^x_{m,s,in}, ψ^y_{m,s,in}, ψ^z_{m,s,in} = 0
            Ready.push_back(s)
            slices[s].done = true
            for j = 0 to NumContained[s] − 1 do
                ψ^c_{m,s,in}, ψ^x_{m,s,in}, ψ^y_{m,s,in}, ψ^z_{m,s,in} +=
                    Transform(ψ^c_{m,s*,in}, ψ^x_{m,s*,in}, ψ^y_{m,s*,in}, ψ^z_{m,s*,in})
            end for
        end if
    end for
    if N_Ready = 0 then
        Break from the stage loop.
    end if
end for
```

III. RESULTS AND ANALYSIS

The research presented here has led to the development of a discrete ordinates transport code named SliceT. As is the case with the traditional SBA, an attractive feature of SliceT is that it is able to handle arbitrary polyhedral spatial meshes. As the code is still in the early stages of development, it currently uses source iteration for its inner iteration method without acceleration, although the authors are currently in the process of implementing the Modified Interior Penalty Diffusion Synthetic Acceleration (MIP DSA)[4] method. The code currently uses the standard Gauss-Seidel method for the multi-group outer iterations, and it currently handles only Gauss-Chebyshev product quadrature sets in angle. Cross sections are supplied in the matxs format provided by NJOY[5]. All other input is in OpenFOAM format[6], although the code is not written using the OpenFOAM libraries. This format is used primarily to take advantage of the meshing utilities and post-processing abilities that OpenFOAM provides.

1. Convergence Order

Before presenting results on the accuracy of the improved SBA compared to the traditional slice balance and cell balance approaches, it is useful to confirm that the second order convergence rate of the LDFE spatial discretization scheme is indeed retained for a smooth solution. This is done via the use of the manufactured solution

$$\psi_m(r) = \sin\left(\frac{\pi x}{H}\right)\sin\left(\frac{\pi y}{H}\right)\sin\left(\frac{\pi z}{H}\right) \tag{16}$$

on a cubic spatial domain of side length H, with homogeneous material properties, isotropic scattering, a single energy group, and a Gauss-Chebyshev quadrature set consisting of 100 total angles (10 polar and 10 azimuthal). This solution is shown at the mid-plane of the domain in Figure 5. For this convergence study, we define the L^2 relative error norm as

$$e = \sqrt{\frac{\sum_{i=1}^{N} V_i \left(\langle\phi\rangle_{c,i} - \langle\phi\rangle_{e,i}\right)^2}{\sum_{i=1}^{N} V_i \langle\phi\rangle_{e,i}^2}}, \tag{17}$$

where N is the number of cells, V_i is the volume of cell i, $\langle\phi\rangle_{c,i}$ is the calculated cell-averaged scalar flux in cell i, and $\langle\phi\rangle_{e,i}$ is the exact cell-averaged scalar flux in cell i. This error norm as a function of cell edge length is shown in Figure 6.

These results show that the second order convergence rate is indeed attained with the use of the improved SBA, although admittedly it does not show a comparison to the convergence rate of the standard LDFE cell balance method. This comparison is currently being performed, and it is anticipated that the proportionality constant will be smaller for the improved SBA due to the increased spatial resolution on which the streaming plus collision operator is inverted. As this is a smooth solution, the more accurate representation of particle streaming is not anticipated to produce significant gains in accuracy over the standard LDFE cell balance method.

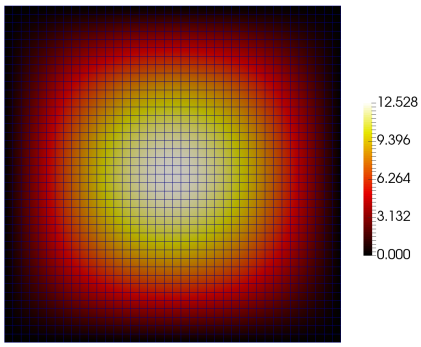

Fig. 5. Manufactured scalar flux solution at the mid-plane.

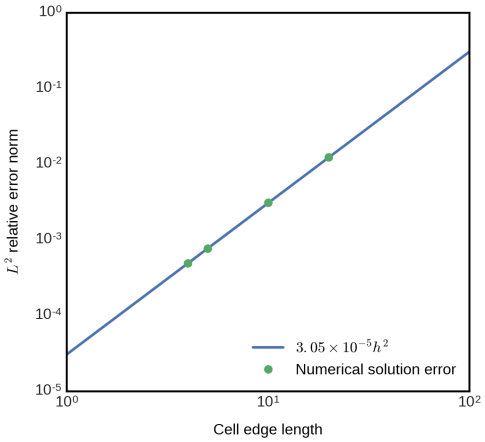

Fig. 6. L^2 relative error norm as a function of cell edge length.

2. Ray Propagation

To compare the accuracy of the cell balance, slice balance, and improved slice balance approaches for the diamond difference and LDFE spatial discretization schemes, consider the propagation of a single ray in 2-D. This problem was motivated by Matthews[7] where several finite element, finite volume, nodal, and characteristic methods were compared for their ability to reduce numerical diffusion, lateral oscillations, and correct propagation direction. While this analysis will not go into nearly as much depth as the cited study, we use the same geometric parameters given below and compare the un-normalized solution for each of the six combinations of balance approach and spatial discretization scheme. The problem examines the propagation of a single ray initiated from a single face on the x boundary nearest the origin. The analytic solution is a decaying exponential ray of width Δx, and zero everywhere else, exhibiting a sharp angular discontinuity. Figure 7 shows numerical solutions for the six combinations of balance method and discretization scheme.

$$\mu = 0.3500212 \qquad \eta = 0.8688903$$
$$\Delta x = 0.5 \qquad \Delta y = 0.5$$
$$\sigma = 0.02 \qquad \sigma_s = 0$$

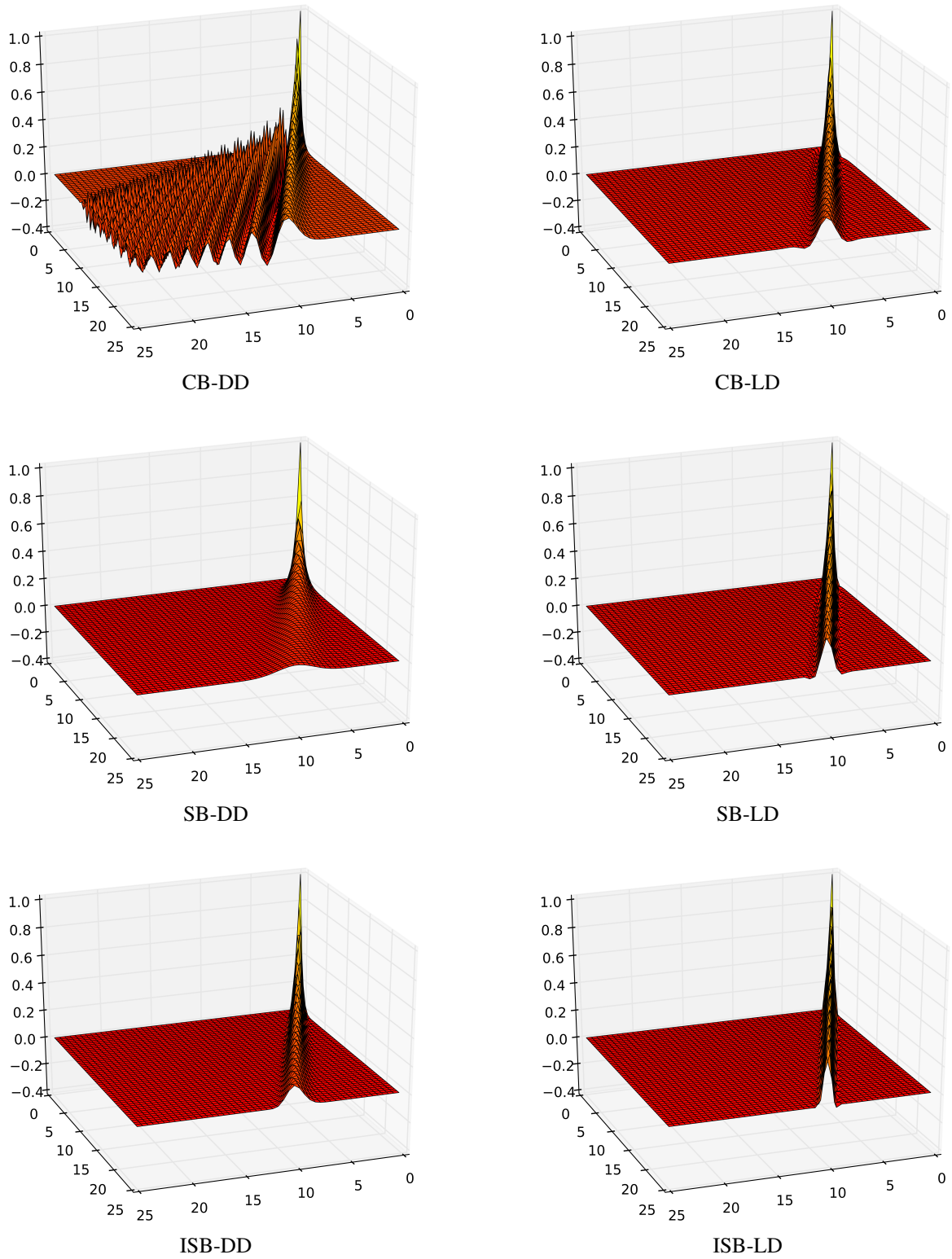

Fig. 7. Cell balance (CB), slice balance (SB), and improved slice balance (ISB) solutions using diamond difference (DD) and linear discontinuous (LD) finite elements for propagating a single ray in 2-D.

Figure 7 reveals several important results. It shows that the traditional SBA reduces the lateral oscillations present in the cell balance approach for both discretization schemes. This is incredibly effective for the diamond difference scheme due to that method's known propensity for producing large lateral oscillations. These oscillations, and the fact that the cell balance diamond difference scheme produces a ray that veers slightly off the propagation direction, led Matthews to recommend that while diamond difference is inexpensive, it is extremely unphysical, and we should abandon it and all of its offspring[7]. This analysis shows that the unphysical oscillations can indeed be fixed with very little extra computation via SBA.

If we further apply the improved SBA to the diamond difference scheme, the results improve even more, producing a sharper ray with less numerical diffusion. Both the SBA and improved SBA also steer the diamond difference ray closer to the true propagation direction. The same results are true for linear discontinuous scheme, however in this case the lateral oscillations in the cell balance approach are much less severe, so there is less of a problem to correct. Applying the improved SBA to the linear discontinuous scheme produces the sharpest ray of all six, traveling in a direction indistinguishable from the true propagation direction with the given mesh resolution. If the extra computation cost can be afforded, it is clear that the improved SBA with the linear discontinuous scheme is the best choice for problems with shadow-type discontinuities.

3. Weak Scaling

The results of a weak scaling study in which an S_4 quadrature set consisting of 32 total angles, one energy group, and isotropic scattering, are shown in Figure 8. Each node of the host machine contains 16 cores, and the functions to build the slices and sub-slices, as well as the sweep within each pipe are threaded on these cores using OpenMP. The problem being solved is a spherical volumetric source of unit strength in the center of a cubic domain. The domain is decomposed into cubic sub-domains, each containing 125,000 spatial cells (50 cells in each dimension). To keep the overall domain cubic, and the total number of nodes even, the data points were obtained at 1, 2^3, 4^3, and 6^3 nodes (core counts from 16 to 3,456). The results show clearly that as the number of angles per angle set increases, so does the parallel efficiency. This is advantageous for problems with large angular quadrature sets. The reason for this is that as the number of angles per angle set decreases, the number of pipes per angle increases, further refining the mesh and leading to more work and making it more difficult to perfectly load-balance the work among pipes by strategically placing the pipe boundaries, which is done prior to the simulation using a method proposed by Ghaddar[8].

4. GPU Acceleration

A serial implementation of the improved SBA shows that the slice and sub-slice setup functions (all functions aside from the sweep) take up roughly 90% of the computation time. Implementing the counting and integration functions for both the slices and sub-slices on an NVIDIA GTX-870m GPU shows that this fraction can be significantly reduced, and if pipe-lined to where the GPU is computing the slices for its pipe in the next angle set, while the CPU is performing the sweep in the previously sliced pipe, the slice and sub-slice setup time could potentially be hidden altogether. The run times of these functions can be seen in Figure 9.

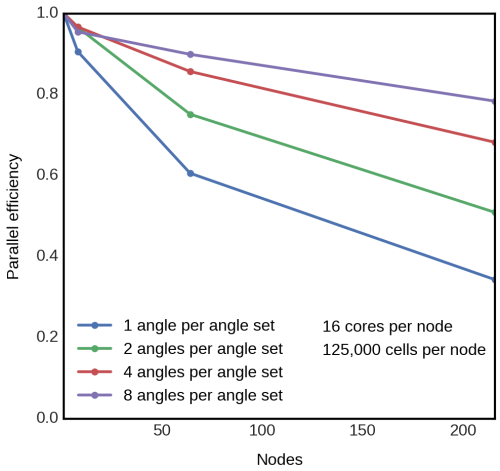

Fig. 8. Weak scaling results.

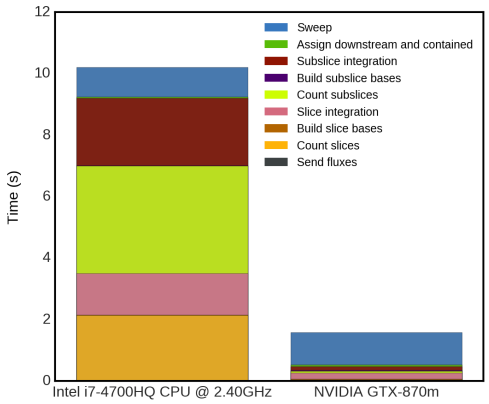

Fig. 9. Run time comparison between CPU and GPU.

5. Arbitrary Polyhedral Unstructured Meshes

Finally, it is often the case that methods that are able to handle arbitrary polyhedral unstructured meshes are demonstrated on structured or unstructured Cartesian meshes, as has been done here up to this point. For this reason, and admittedly for fun, we now consider a volumetric, isotropic, unit strength radiation source in the shape of the author's dog, Jethro. The simulation uses an S_4 quadrature set, consisting of 32 total angles, and isotropic scattering. Jethro is depicted in Figure 10, the geometric model used in the simulation is shown in Figure 11, and the scalar flux solution at one mid-plane of the cubic bounding domain is shown in Figure 12. There is no analytic solution to verify solution accuracy in this case, but the results appear reasonable.

Fig. 10. Jethro.

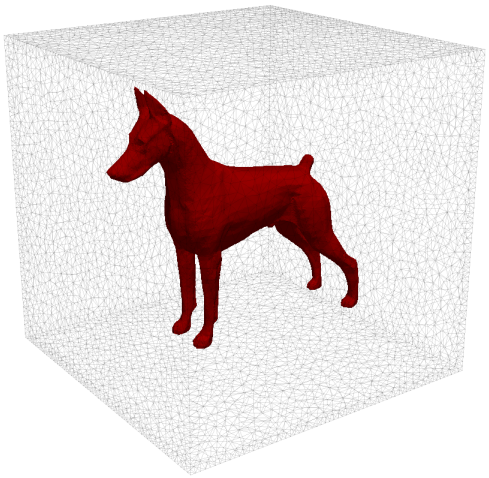

Fig. 11. Jethro-shaped source in cubic bounding domain.

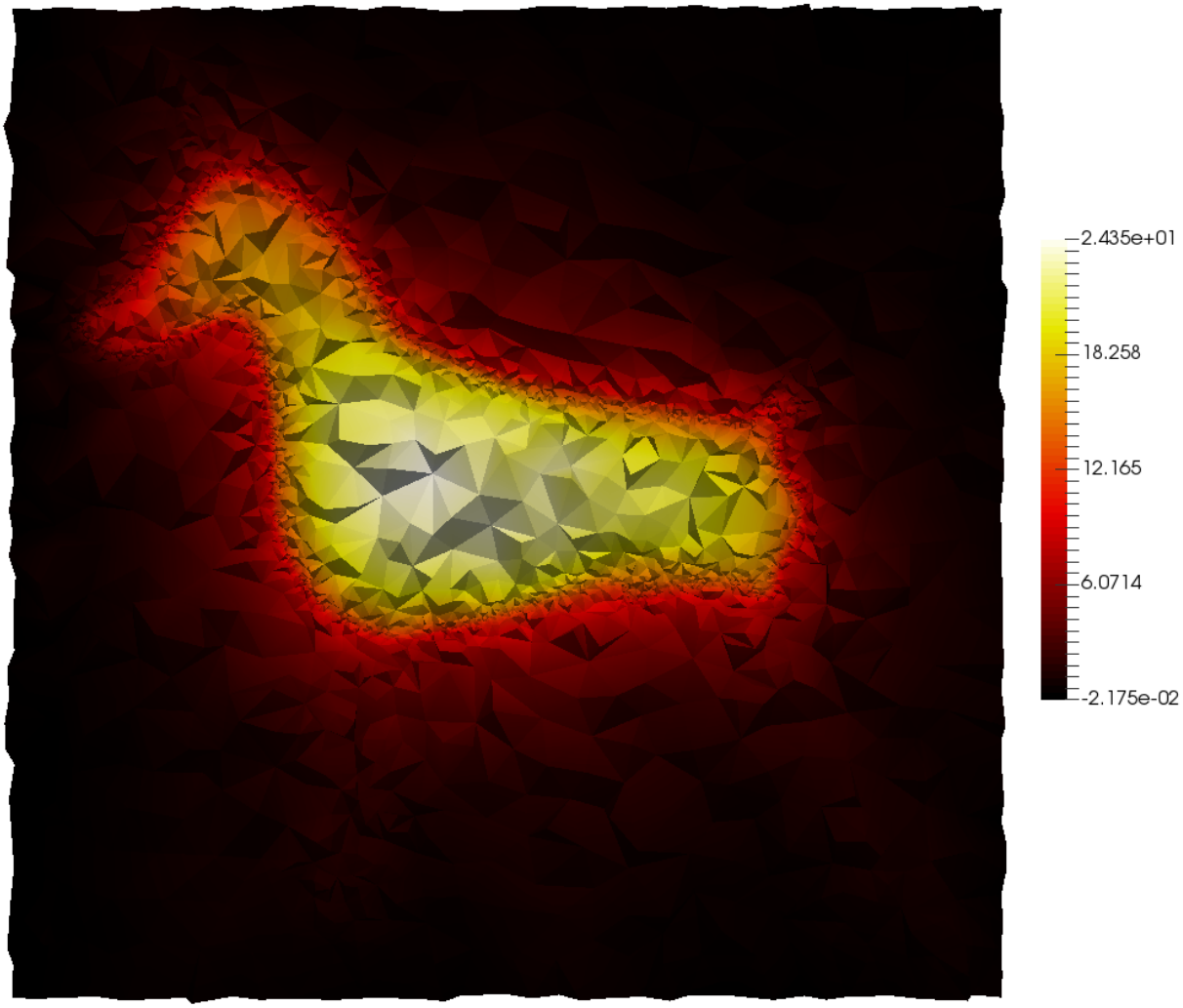

Fig. 12. Scalar flux solution at the mid-plane for the Jethro-shaped source.

IV. CONCLUSION

The improvement to the SBA presented here retains all of the benefits of the traditional SBA, including the ability to easily handle arbitrary unstructured meshes, while improving accuracy in the vicinity of solution discontinuities, with relatively little added cost. This is done via a more consistent representation of particle streaming through the addition of the "sub-slice." This improvement in accuracy has been demonstrated by propagating a single ray in 2-D. For smooth problems, the second order spatial convergence for the LDFE spatial discretization scheme is retained. This was demonstrated using a manufactured solution.

Through the addition of the sub-slice, a new parallelization strategy is made possible. This strategy exhibits zero idle time at the expense of added communication and memory requirements. The added memory requirement is simply that the entire mesh description must be stored on each shared memory node. This added memory requirement amounts to roughly 320 bytes per spatial cell, which is thought to be acceptable due to the trend in high performance computing, where supercomputer architectures are becoming more heterogeneous with larger amounts of memory per node. For instance, the Summit and Sierra super-computers currently under procurement by the United States Department of Energy will have 512 GB of memory per node, shared between an IBM CPU and an NVIDIA GPU[9].

A weak scaling study shows that the parallel efficiency of the new strategy improves upon inclusion of more angles per angle set, where all angles in an "angle set" are swept simultaneously. While investigation into the new parallelization strategy is not complete, a compelling benefit of the strategy is that it allows for an arbitrary domain decomposition on which to store the solution. It is the storage of the solution, and not the mesh itself, that leads to most transport problems becoming memory bound. This arbitrary domain decomposition removes a common restriction that inter-node domain boundaries be planar in order to avoid ray re-entry. The benefit is realized in the mesh generation process, where domain decomposition methods common in the field of computational fluid dynamics such as the scotch algorithm[10], can be used freely. The benefit of using such an arbitrary domain decomposition strategy was demonstrated by the Jethro source problem, containing over 1.5 million cells with no natural planar divisions in the problem geometry.

Finally, it is the case that finite element implementations into both the traditional and improved SBA would suffer from the need to repeatedly calculate matrices for each slice occurring in the slice-refined mesh for each angle in the quadrature set. As the number of angles necessary for 3-D problems can be large, the memory demand to store these volume and area integrals for each slice cannot be accommodated, even on the next-generation of super-computers mentioned above. For the diamond difference spatial discretization scheme, this is not an issue, but for finite element methods it is, and this may explain why reports of such implementations could not be found prior to beginning this research. As shown by the timing results of Figure 9, the repeated calculation of finite element matrices may be practical through the use of the GPU. This is because the GPU is designed to handle just this type of task, which is computationally expensive but embarrassingly parallel. In addition, the results shown in Figure 9 were obtained using a gaming laptop GPU that is over two years old, has fewer double precision compute cores per each single precision compute core, and has fewer compute cores overall by a factor more than 2, than the Tesla series GPUs found in Summit and Sierra. It is anticipated that these GPUs will reduce the percentage of time spent computing matrices even further. A strategy in which a single core of the CPU on each node can be devoted to communicating and directing the GPU in order to pipeline the setup and sweep functions may be able to hide the slice and sub-slice setup time altogether.

V. ACKNOWLEDGMENTS

This material is based upon work supported by the United States Department of Energy National Nuclear Security Administration Stewardship Science Graduate Fellowship. We thank Jethro for the use of his likeness.

REFERENCES

1. R. E. GROVE, *A Characteristic-based Multiple Balance Approach for Solving the S_N Equations on Arbitrary Polygonal Meshes*, Ph.D. thesis, University of Michigan (1996).
2. J. E. MOREL and E. W. LARSEN, "A Multiple Balance Approach for Differencing the S_N Equations," *Nuclear Science and Engineering*, **105**, *1*, 1–15 (1990).
3. R. A. KENNEDY, A. M. WATSON, and R. E. GROVE, "Linear Discontinuous (LD) Coefficients In The Slice Balance Approach (SBA) Mathematical Framework For The Discrete Ordinates Code Jaguar," in "Proceedings of PHYSOR 2010," American Nuclear Society, LaGrange Park, IL (2010).
4. Y. WANG and J. C. RAGUSA, "Diffusion Synthetic Acceleration for High-Order Discontinuous Finite Element S_N Transport Schemes and Application to Locally Refined Unstructured Meshes," *Nuclear Science and Engineering*, **166**, *2*, 145–166 (2010).
5. A. KAHLER, R. MACFARLANE, D. MUIR, and R. BOICOURT, "The NJOY Nuclear Data Processing System, Version 2012," *Los Alamos National Laboratory (Dec. 2012)* (2012).
6. OPENCFD, "OpenFOAM user guide," *OpenFOAM Foundation*, **2**, *1* (2011).
7. K. A. MATHEWS, "On the propagation of rays in discrete ordinates," *Nuclear science and engineering*, **132**, *2*, 155–180 (1999).
8. T. GHADDAR, *Load Balancing Unstructured Meshes for Massively Parallel Transport Sweeps*, Master's thesis, Texas A&M University (2016).
9. NVIDIA CORPORATION, "Summit and Sierra Supercomputers: An Inside Look at the U.S. Department of Energy's New Pre-Exascale Systems," Whitepaper (2014).
10. F. PELLEGRINI and J. ROMAN, *Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 493–498 (1996).