ANS MC2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method • Nashville, TN • April 19-23, 2015, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2015) SAND2015-1941C

PARALLEL DETERMINISTIC TRANSPORT SWEEPS OF STRUCTURED AND UNSTRUCTURED MESHES WITH OVERLOADED MESH DECOMPOSITIONS

Shawn D. Pautz*
Sandia National Laboratories
Albuquerque, NM 87185-1179
sdpautz@sandia.gov

Teresa S. Bailey
Lawrence Livermore National Laboratory
P.O. Box 808, L-561
Livermore, CA 94551
bailey42@llnl.gov

ABSTRACT

The efficiency of discrete-ordinates transport sweeps depends on the scheduling algorithm, domain decomposition, the problem to be solved, and the computational platform. Sweep scheduling algorithms may be categorized by their approach to several issues. In this paper we examine the strategy of domain overloading for mesh partitioning as one of the components of such algorithms. In particular, we extend the domain overloading strategy, previously defined and analyzed for structured meshes, to the general case of unstructured meshes. We also present computational results for both the structured and unstructured domain overloading cases. We find that an appropriate amount of domain overloading can greatly improve the efficiency of parallel sweeps for both structured and unstructured partitionings of the test problems examined on up to 10^5 processor cores.

Key Words: transport sweeps, parallel transport, domain overloading

1 INTRODUCTION

Standard solution techniques for discretizations of the first-order form of the Boltzmann transport equation often make use of "sweeps". During a sweep the streaming-plus-collision operator is approximately inverted for each spatial element in the mesh and each direction in the discrete-ordinates set. Since discretization of the streaming-plus-collision operator yields a block-lower-triangular matrix, it can be inverted by Gaussian elimination, provided that operations are carried out in an appropriate order. The partial ordering of tasks imposed by Gaussian elimination indicates that these tasks are not concurrent, which poses a challenge to the scalability of sweep algorithms.

Various parallel scheduling algorithms have been developed to manage the concurrency of the sweep operator while also accounting for other concerns such as communication costs [1-5]. Recent work has demonstrated strategies for developing "optimal" sweep algorithms for the structured mesh case; however, these strategies have yet to be generalized to the unstructured

---

* To whom correspondence should be addressed

mesh case. In the present work we show one way to generalize the method of domain overloading, which has been shown to have useful properties for structured algorithms. We present computational results for structured mesh sweeps with volumetric overloaded partitioning, which previously had only been theoretically analyzed. We also present results for the unstructured algorithm applied to the same meshes.

The rest of the paper is organized as follows. In Section 2 we illustrate the challenge of parallel sweep scaling and provide a taxonomy for existing sweep algorithms. We describe several such algorithms and define our new unstructured overloading algorithm. In Section 3 we present numerical results for both structured and unstructured overloading. In Section 4 we present conclusions and ideas for future work.

2 THEORY

In this section we illustrate the challenge of obtaining good parallel sweep efficiency with a simple example. We provide a taxonomy of existing sweep algorithms. We then focus on mesh partitioning, noting some common approaches for structured meshes. We then show how we may generalize the concept of overloading to unstructured mesh partitioning.

2.1 General Issues of Parallel Sweeps

We begin by illustrating the process of sweeping. In Figure 1 we depict a small structured mesh as well as one direction $\vec{\Omega}$ from a discrete ordinates set. In order to invert the streaming-plus-collision term in any of the spatial elements for direction $\vec{\Omega}$ we must know all fluxes on "incoming" faces of the element, that is, for faces on which $\vec{\Omega} \cdot \vec{n} < 0$, where $\vec{n}$ is the outward normal on the face. At the beginning of the sweep these fluxes are known only on the external boundary of the problem; in this case only element 4 has all incoming fluxes known. Once the streaming-plus-collision term is inverted in element 4, we may use its outgoing fluxes to calculate the incoming fluxes on the faces adjoining elements 3 and 8; thus the solutions to elements 3 and 8 depend on that of element 4. Similar dependencies exist at every face shared by two elements. These dependencies are illustrated in the "sweep graph" of Figure 2. No task in the graph may be started until all of its upstream dependencies have been resolved. The maximum width of the graph (i.e. the number of tasks in one row of the graph) gives the maximum concurrency of the sweep tasks (i.e. the maximum number of tasks that may be worked on simultaneously). In this particular case we can use at most four processors productively, even though there are a total of sixteen tasks. Furthermore, to use these processors efficiently we must assign tasks in a constructive manner. If some of the tasks in one level of the graph are assigned to the same processor we will have degraded the potential efficiency of the sweep process.

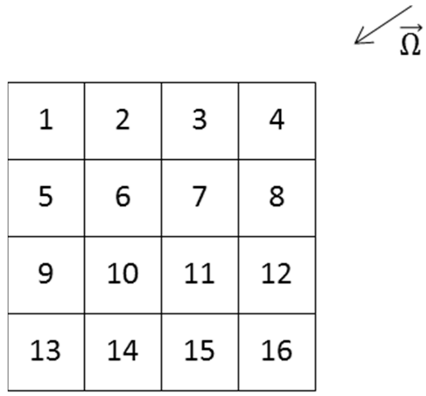

Figure 1. Example sweep problem.

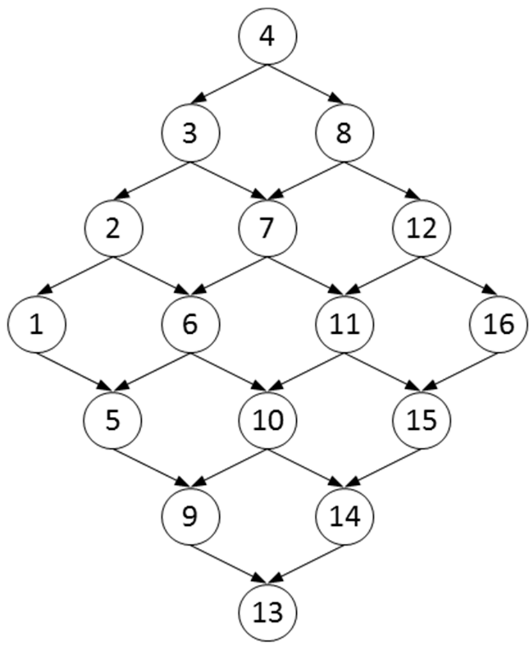

Figure 2. Example sweep graph.

To address the above situation a variety of sweep scheduling algorithms have been constructed. We may define a taxonomy to describe these algorithms based on how they address several issues:

1. Mesh partitioning/decomposition

   Typically sweep scheduling algorithms use only spatial decomposition: they assign all sweep tasks for a given spatial element to the same processor in order to limit the communication costs of other parts of the transport algorithm. Different algorithms may use different strategies for this element-to-processor assignment.

2. Task aggregation

It can be useful to group multiple tasks on the same processor as a larger atomic entity, for example grouping multiple elements and angles allows for a reordering of loops. Other operations are disallowed until the streaming-plus-collision operator is inverted in all elements of the group. Various amounts and types of aggregation are used by different algorithms.

3. Message aggregation

   Completion of tasks sometimes generates data needed by other processors. This data may be sent as soon as possible or aggregated with data generated later to produce a smaller number of messages.

4. Scheduling heuristics

   As the sweep progresses a given processor will likely have more than one task available to be worked on. Different strategies may be employed to give relative priorities to different tasks.

We believe that the above classification usefully describes existing sweep algorithms. Not all algorithms will make explicit distinctions among the above four categories. For example, some algorithms merge the idea of mesh partitioning and task aggregation; task and message aggregation also may be treated as a single concept.

2.2 Mesh Partitioning Algorithms

In the present work we will focus on mesh partitioning algorithms. We will describe different approaches for structured algorithms and note how (or whether) they have been extended to unstructured algorithms. For our purposes we will restrict the term “partitioning” or “decomposition” to the element-to-processor assignment only, even though in other contexts the same terms may be used to indicate aggregation strategies. Also, when we use the term “unstructured algorithm”, we may apply it to the decomposition of either structured or unstructured meshes, provided that structured information (e.g. rows, columns) is not used to define the decomposition; only general concepts such as element connectivity may be used.

Although we will describe “different” mesh partitioning algorithms below, we note that several structured algorithms are just special cases of the same general formula. Consider a structured mesh with a $N_x \times N_y \times N_z$ element layout, with each element identified by the triplet $\{i, j, k\}$. We define an “overloading” layout of $\omega_x \times \omega_y \times \omega_z$ (the overloading concept will be described later). We also define a processor “tile” of $P_x \times P_y \times P_z$; a given processor may be identified by $\{p_x, p_y, p_z\}$. In this case we can define a general structured mesh partitioning with the following element-to-processor assignment:

$$p_x = \left(i\%(N_x/\omega_x)\right)/\left(N_x/(\omega_x P_x)\right) \tag{1a}$$

$$p_y = \left(j\%(N_y/\omega_y)\right)/\left(N_y/(\omega_y P_y)\right) \tag{1b}$$

$$p_z = \left(k\%(N_z/\omega_z)\right)/\left(N_z/(\omega_z P_z)\right) \tag{1c}$$

where all divisions use integer math and % is the modulo operator. A corresponding set of equations for the general unstructured case is not known. For this reason we will examine particular special cases of the above equation.

An important concept for understanding sweep efficiency for structured grids is *pipefill*. The pipefill time is the time it takes for all processors to receive boundary information from an upstream processor or a physical boundary condition. In general, for structured grids, the pipefill penalty is proportional to

$$pipefill \propto P_x + P_y + P_z \tag{2}$$

2.2.1 Contiguous volumetric

In the structured case we may obtain a "contiguous volumetric" decomposition by setting $\omega_x = \omega_y = \omega_z = 1$ and $P_x \approx P_y \approx P_z$ in Equation (1). An example of such a decomposition is given in Figure 3 for a 2D mesh of 32x32 elements and $P_x = P_y = 4$. We use the term "contiguous" since all of the elements assigned to a given processor will form one contiguous block. The term "volumetric" is used to indicate that such a block will minimize the surface-to-volume ratio (i.e. the number of faces on its boundary) and thus the communication volume during sweeps. The efficiency of sweeps on such a decomposition will be problem-dependent. It generally will be good if the number of directions is high and better than other partitionings if communication costs are high. It can have poor efficiency if the number of directions is low relative to the number of processors. The pipefill penalty for this type of decomposition is $O(P^{1/3})$ in 3D and $O(P^{1/2})$ in 2D.

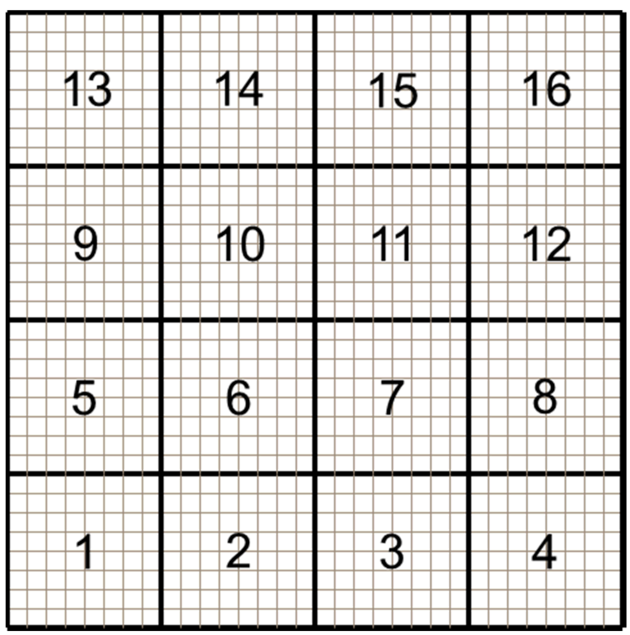

Figure 3. Contiguous volumetric structured decomposition.

In the unstructured case one can use general graph partitioners such as Chaco [6] to obtain an approximation to the structured algorithm [3]. Each vertex in the graph represents an element in the mesh, and undirected edges between vertices represent shared faces between elements. The graph partitioner attempts to divide the graph into P regions (where P is the number of processors) of approximately the same size and with as few "edgecuts" (edges connecting two different regions) as possible. In Figure 4 we depict the application of Chaco to the example mesh from Figure 3. Although it is perfectly load-balanced, the edgecut is slightly higher than for the structured case – the surface to volume ratio is smaller in Figure 3 than in Figure 4. It also lacks symmetry, and the regions are irregular and often concave, which may have implications for other aspects of the overall scheduling algorithm. Although formal analysis may be more difficult than for the structured case, it has been computationally observed to exhibit sweep behavior similar to that with structured decompositions.

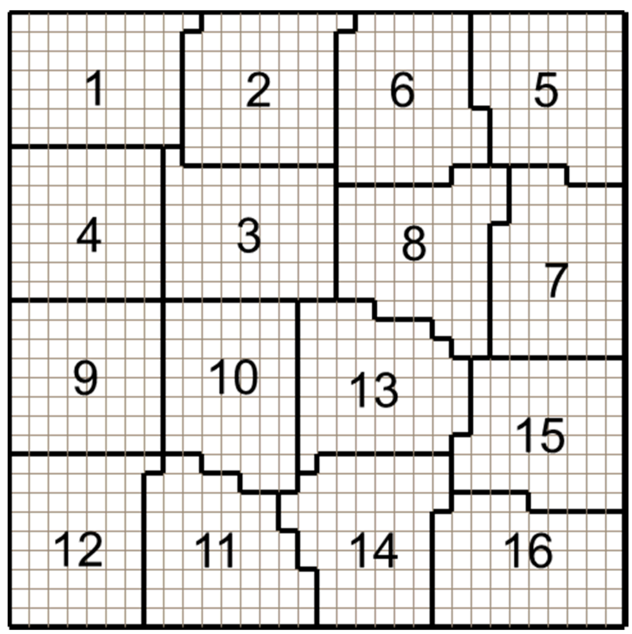

Figure 4. Contiguous volumetric unstructured decomposition.

2.2.2 KBA and hybrid KBA

In the structured case we may obtain a "KBA" or "hybrid KBA" decomposition [1] by setting $\omega_x = \omega_y = \omega_z = 1$, $P_y \approx P_z$, and either $P_x = 1$ (KBA) or $P_x = 2$ (hybrid KBA) in Equation (1). We depict a hybrid KBA decomposition in Figure 5. A useful property of the KBA approach is that it guarantees that any given processor has tasks at multiple levels of the graph, but at the expense of higher communication volume than the contiguous volumetric algorithm. Although the pipefill penalty for this type of decomposition is $O(P^{1/2})$ in 3D and $O(P)$ in 2D (worse than for the contiguous volumetric approach), the proportionality constant is different. KBA may have greater efficiency than contiguous volumetric decompositions if the number of directions is low enough and if communication costs are low enough.

There have been few documented attempts to generalize KBA to unstructured meshes. In [2] the columns of KBA were approximated by projecting the cell centers of a 3D mesh onto a plane and then partitioning those points. Although this produces a geometric approximation to columns, it is not clear if this yields the same graph properties as the structured approach. We are unaware of any other general algorithm to produce an "unstructured KBA" decomposition.

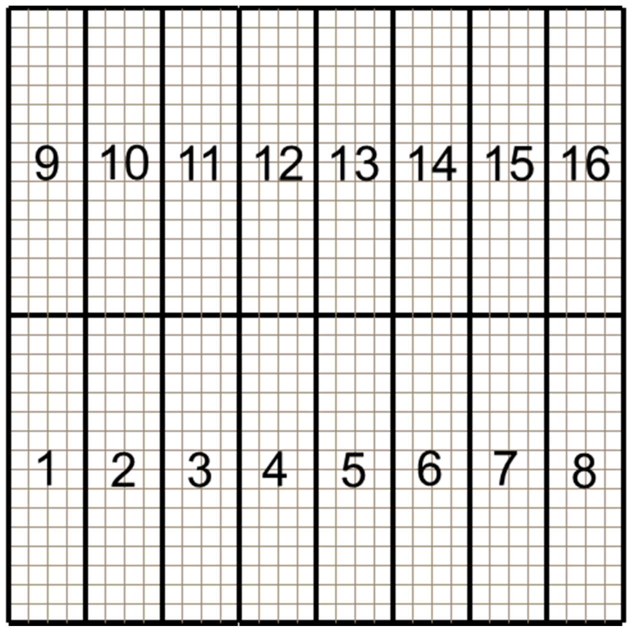

Figure 5. Hybrid KBA partitioning.

2.2.3 Overloading

In the structured case we may obtain an "overloaded" decomposition by setting ω_x, ω_y, and/or ω_z to a value greater than unity [4]. This produces multiple (typically non-contiguous) regions for each processor. We may view this, at least conceptually, as a two-step process. By first defining ω_x, ω_y, and ω_z we obtain an overloaded structure for the mesh, as depicted in Figure 6 for $\omega_x = \omega_y = 2$. In the second step we define P_x, P_y, and P_z to complete the overloaded decomposition, as shown in Figure 7 for $P_x = P_y = 4$, which duplicates the same processor "tile" (layout) in each of the four overloaded regions of Figure 6. The overloaded approach is similar in some ways to KBA in that each processor is assigned tasks at multiple levels of the sweep graph, at the expense of greater communication volume. KBA has been cast as an overloaded volumetric decomposition in [4]. Other parts of the sweep algorithm such as scheduling heuristics may be more complicated for the general overloaded case due to the greater spatial complexity of this decomposition. Although theoretical performance models have been used to make predictions about the efficiency of the overloaded approach, there have been few actual performance results. Only limited results for the Compton-Clouse algorithm [4] have been published for overloaded algorithms. The Compton-Clouse algorithm is specialized and requires power of 2 decompositions. The algorithms in this paper are much more general.

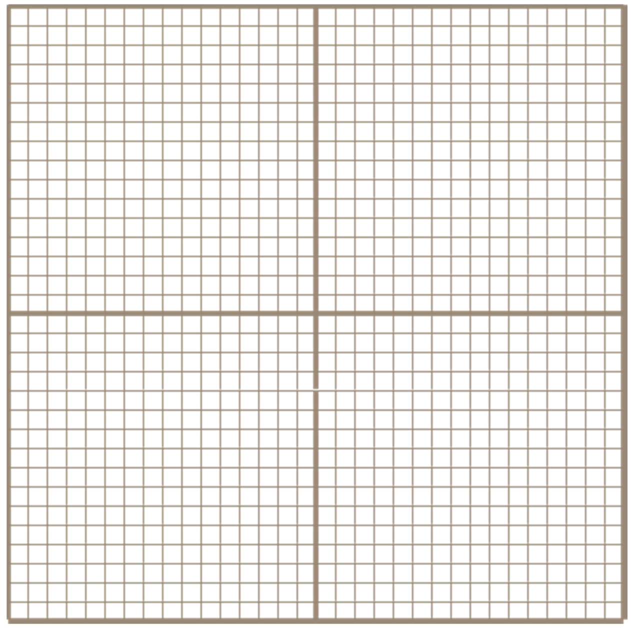

Figure 6. Structured overloading.

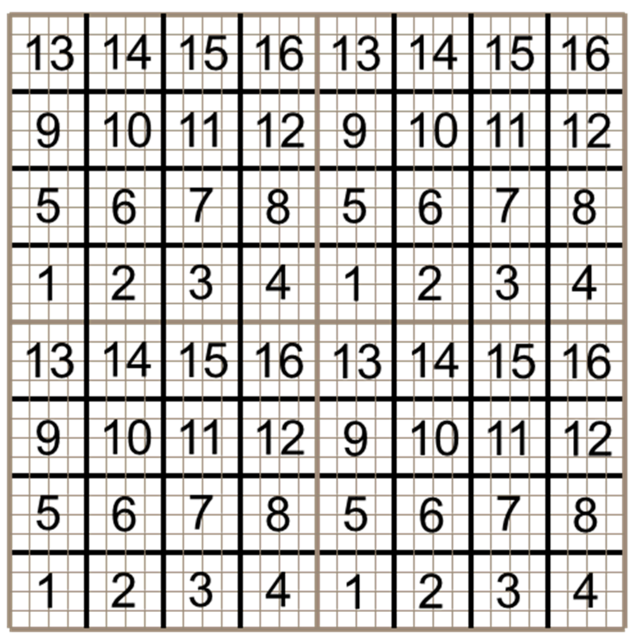

Figure 7. Overloaded structured decomposition.

We now synthesize some of the above concepts to define an unstructured overloaded partitioning algorithm. First we define an overloading parameter ω. Then we use the unstructured volumetric approach to subdivide the mesh into ω (overloaded) regions. This is depicted in Figure 8 for $\omega = 4$. In the second step each overloaded region is separately partitioned into P subregions, again with the unstructured volumetric algorithm. Each processor is then assigned one subregion from each overloaded region; this is depicted in Figure 9 for $P = 16$. This completes the decomposition algorithm.

It is hoped that the above unstructured algorithm captures the essential features of the structured one. It yields multiple non-contiguous regions scattered throughout the mesh for each processor, which should yield tasks at numerous levels of the sweep graph. Within each contiguous region an attempt is made to minimize the communication volume. We also have an overloading parameter ω which may be used to balance the need for concurrency with the need to limit communication costs. Although we do not have an expression comparable to Equation (1), we can view the unstructured contiguous volumetric algorithm as just a special case of the overloaded algorithm for $\omega = 1$. In addition, the same graph partitioner used for the contiguous volumetric approach may be used in the overloaded approach, potentially simplifying its implementation; we merely have leveraged it in a hierarchical way.

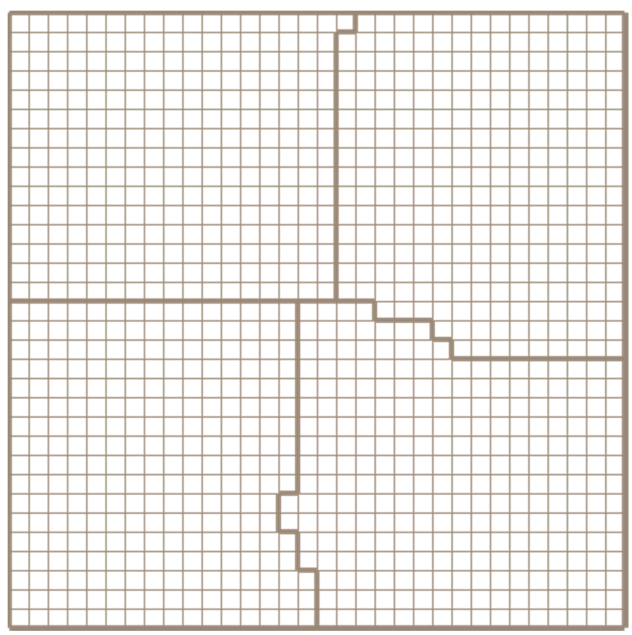

Figure 8. Unstructured overloading.

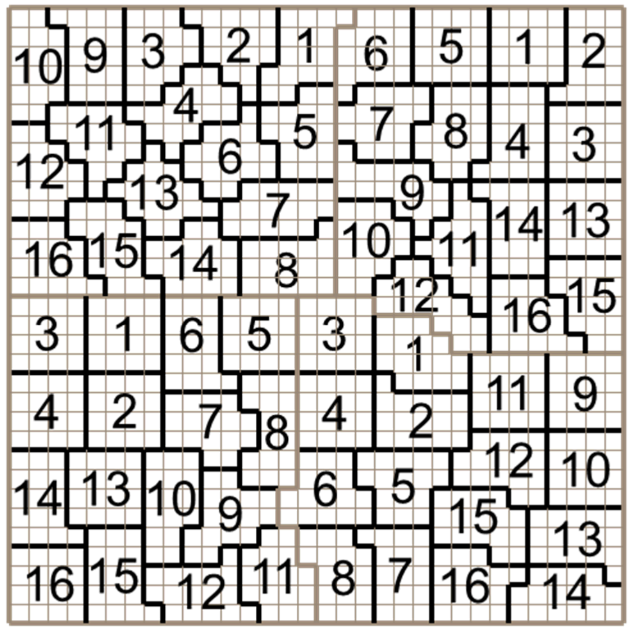

Figure 9. Overloaded unstructured decomposition.

3 NUMERICAL RESULTS

In order to test the overloading technique for both structured and unstructured algorithms we implemented both of these techniques in a mesh partitioning code. The code takes as input a description of the mesh and the desired partitioning algorithm and parameters and produces as output the element-to-processor assignment.

Although the focus of the present work is on mesh partitioning techniques, we do need to make choices regarding other aspects of the scheduling algorithm. For computational studies of the structured algorithm we chose a task aggregation strategy of grouping together all elements within a contiguous overloaded region (i.e. "cellsets") [5]. For the unstructured algorithm no element aggregation was performed (aggregation was observed to degrade the results, which we hope to explore in future work). In both cases we allowed for aggregation of directions by mapping the directions of the angular quadrature set onto a lower-order quadrature and aggregating accordingly ("anglesets") [5]. In all cases the message aggregation was equivalent to the task aggregation: any available messages were sent as soon as each aggregation of tasks completed. For scheduling heuristics we used the DFHDS strategy [3] to prioritize tasks.

Timing results were obtained with the SCEPTRE transport code [7]. SCEPTRE contains a discontinuous finite element discretization of the linear multigroup Boltzmann transport equation for unstructured meshes and uses fully upwind sweeps to solve the streaming-plus-collision term. We defined a simple monoenergetic problem with isotropic scattering and required SCEPTRE to perform five source iterations without any preconditioning in order to obtain timings dominated by the cost of sweeps.

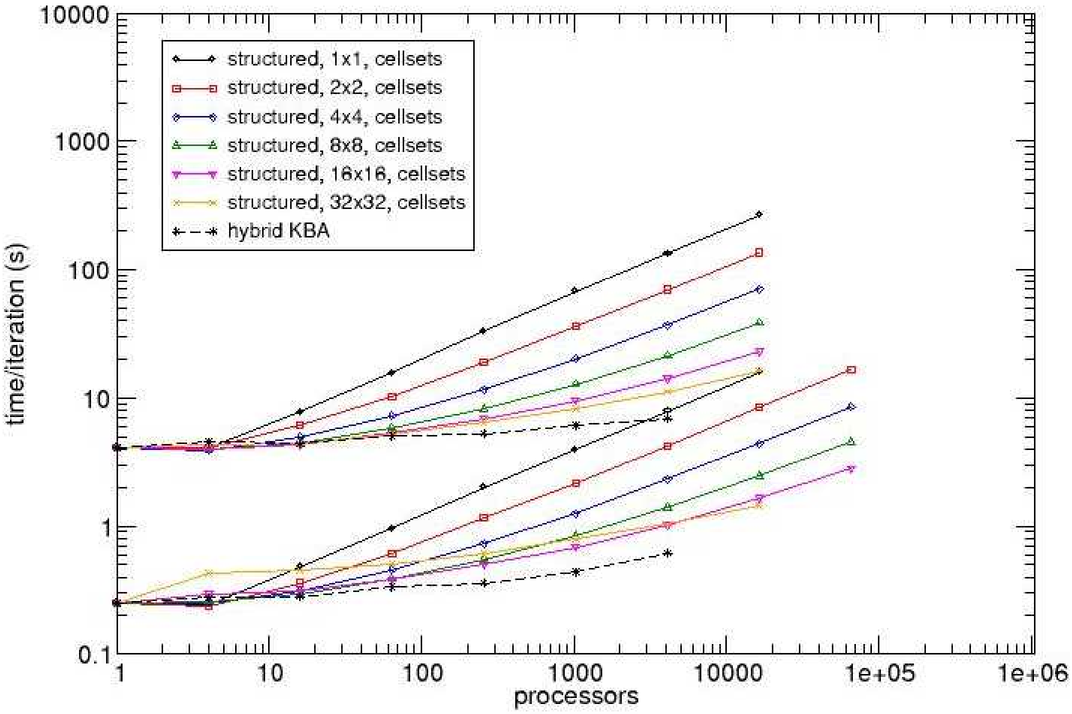

Figure 10. Weak scaling, 2D, S2, structured partitioning.

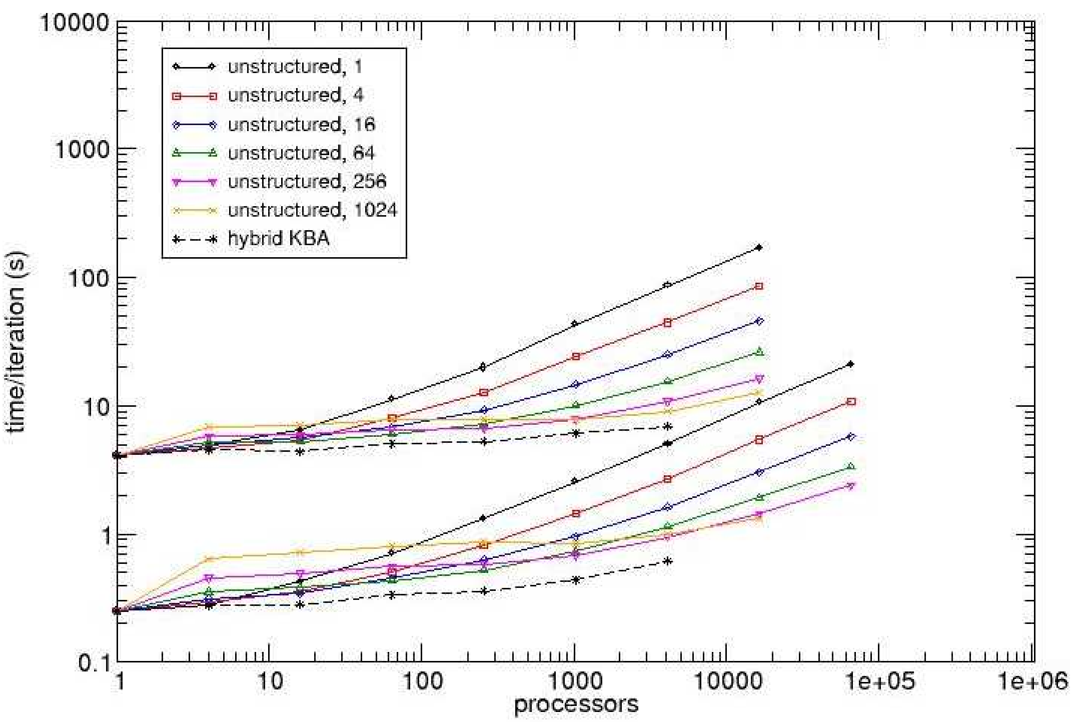

Figure 11. Weak scaling, 2D, S2, unstructured partitioning.

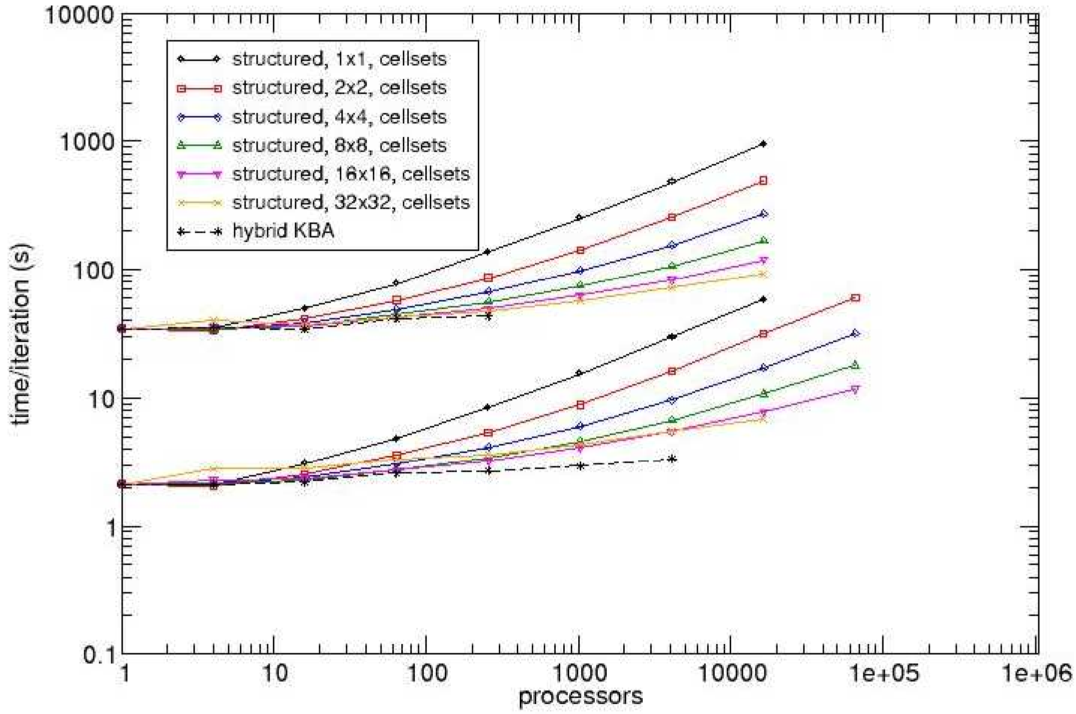

Figure 12. Weak scaling, 2D, S8-S4, structured partitioning.

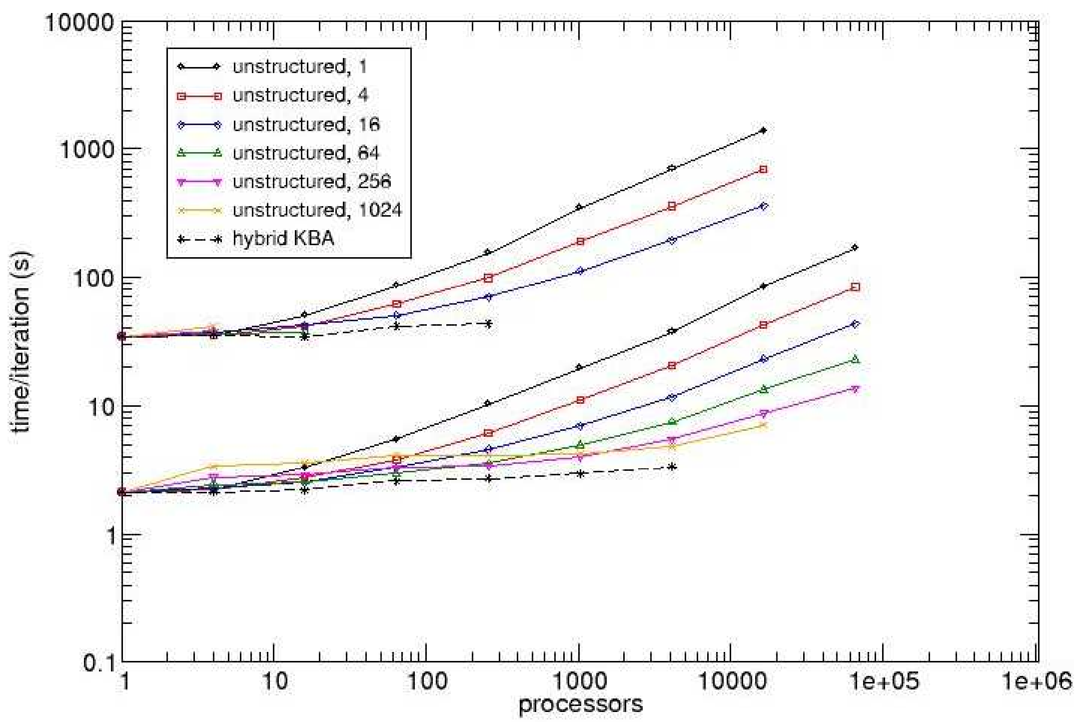

Figure 13. Weak scaling, 2D, S8-S4, unstructured partitioning.

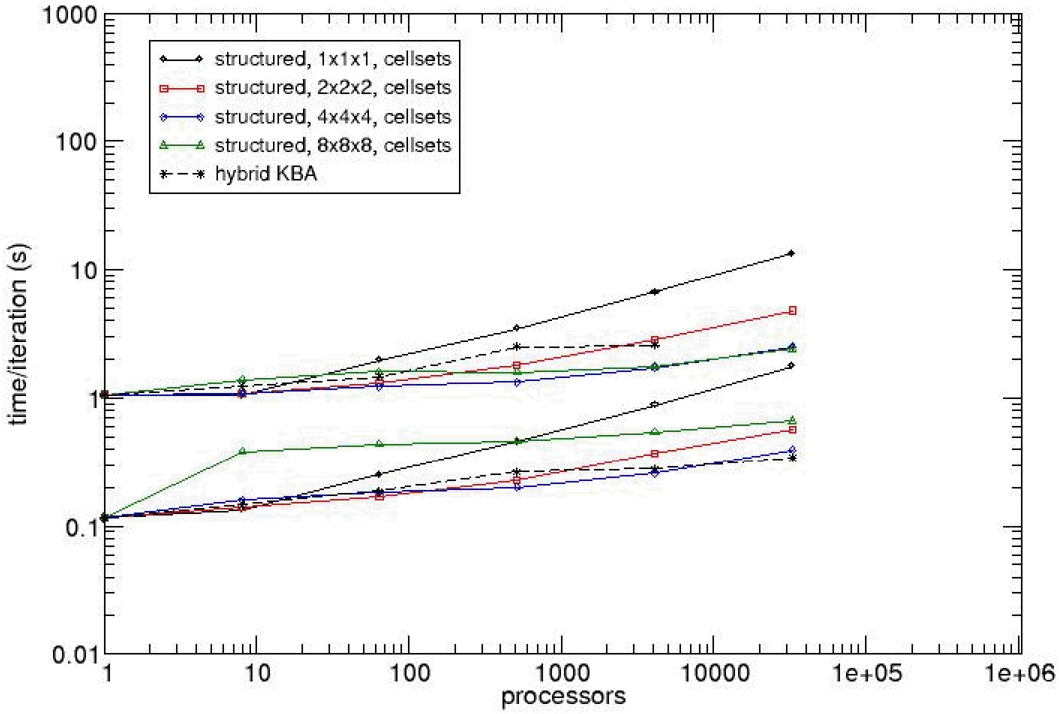

Figure 14. Weak scaling, 3D, S2, structured partitioning.

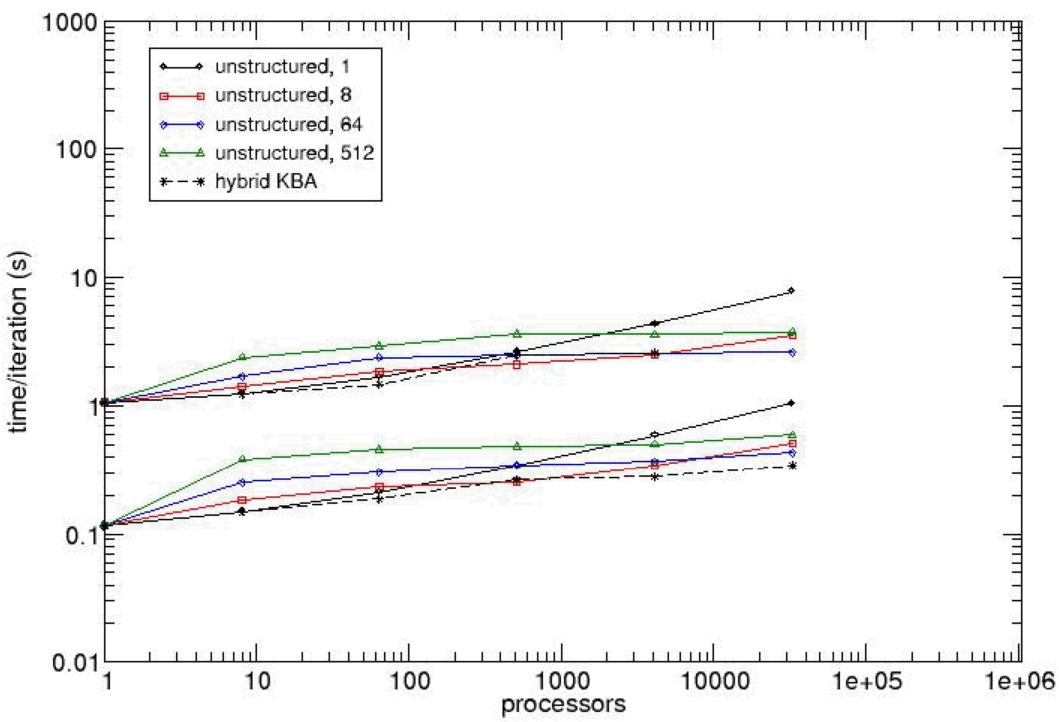

Figure 15. Weak scaling, 3D, S2, unstructured partitioning.

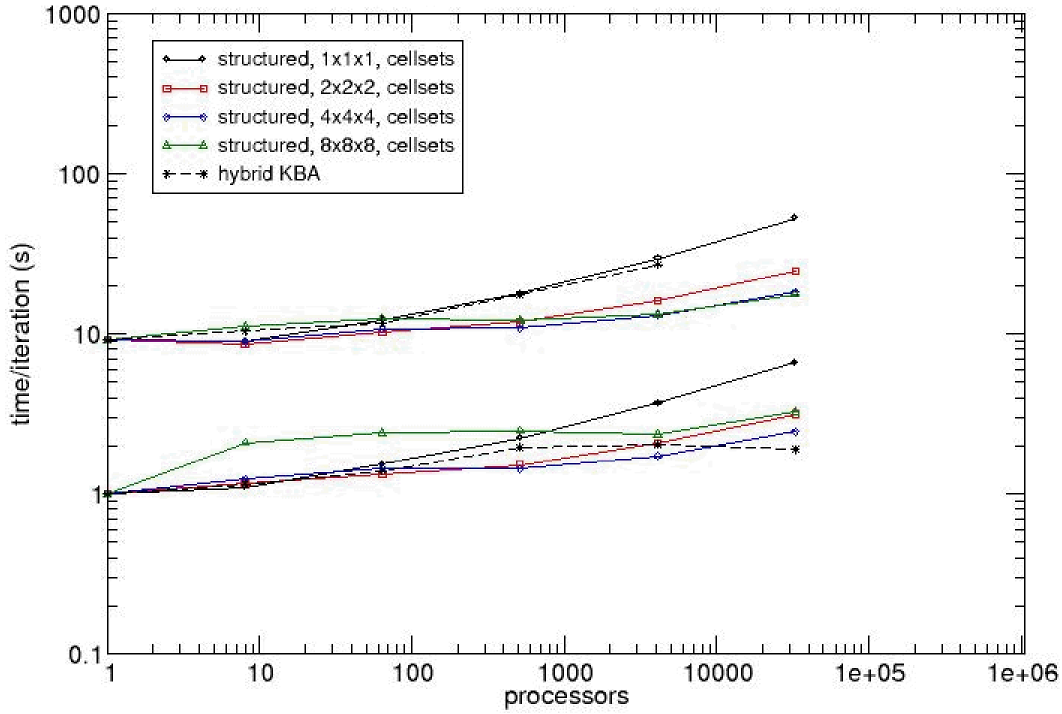

Figure 16. Weak scaling, 3D, S8-S4, structured partitioning.

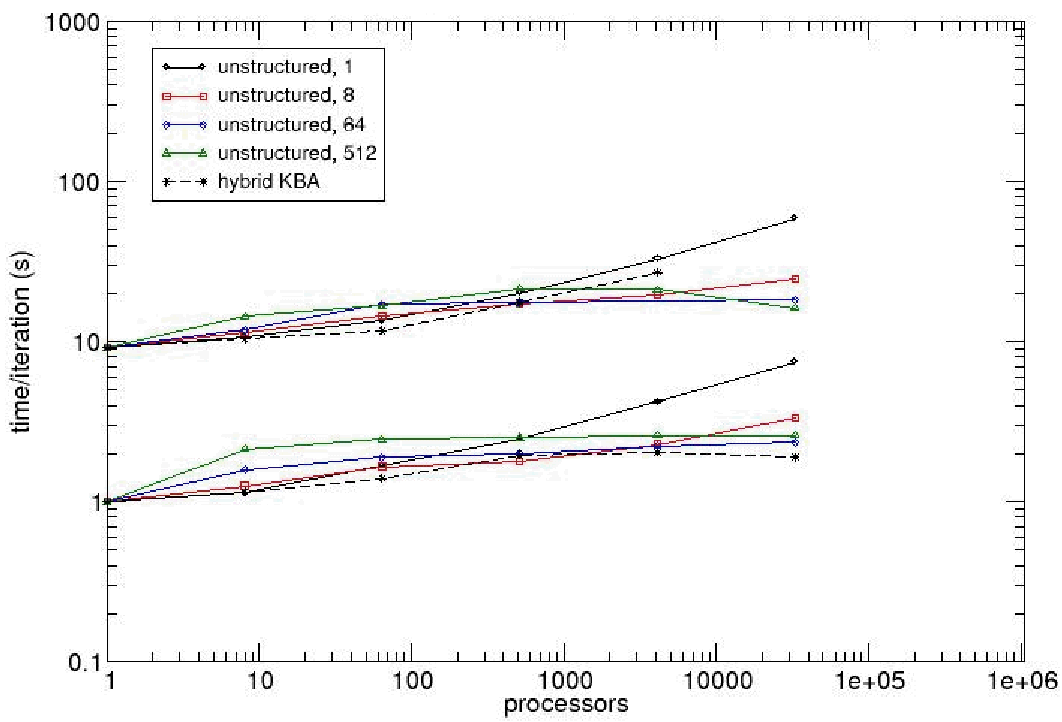

Figure 17. Weak scaling, 3D, S8-S4, unstructured partitioning.

Weak scaling results were generated on simple structured $N \times N$ quadrilateral and $N \times N \times N$ hexahedral meshes. SCEPTRE runs were performed on Cielo, a massively parallel computer located at Los Alamos National Laboratories consisting of 8515 compute nodes, each containing two AMD Opteron 6100 Series 8-way processors. We examined problems with the S_2 level-symmetric quadrature and with the S_8 quadrature aggregated with the S_4 quadrature. We examined the use of both the structured algorithm with $\omega \times \omega$ (2D) or $\omega \times \omega \times \omega$ (3D) overloaded decompositions and the unstructured algorithm with comparable degrees of overloading. We also generated results with hybrid KBA for comparison. All of these results are presented in Figures 10-17. In these figures two sets of results are presented corresponding to 4096 and 65536 (2D) and 512 and 4096 (3D) elements/core. The degree of overloading is indicated in the legend for each scaling curve.

From these results we make the following observations:

- At low levels of parallelism increasing amounts of overloading lead to increased runtimes. This is not surprising, since overloading increases communication costs in order to redistribute concurrent tasks among the processors. At low core counts little or no redistribution of concurrency is necessary in comparison to no overloading in order to achieve high efficiencies. These results are completely consistent with the theoretical understanding of structured mesh sweeps [5].
- At increasingly higher levels of parallelism improvements are seen when the amount of overloading is also increased. This is what is hoped for with overloading; we have purchased much-needed concurrency at the expense of additional communications costs. These results are completely consistent with the theoretical understanding of structured mesh sweeps [5].
- The results for unstructured overloading are comparable to those for structured overloading for equivalent degrees of overloading. This is encouraging and what we desired from the unstructured algorithm. It is also somewhat surprising since we used two different extremes of aggregation for structured versus unstructured scheduling.
- Hybrid KBA often, but not always, outperformed the overloading technique; sufficient overloading produced comparable results. This also is not too surprising, since existing structured performance models do predict that the KBA approach and the overloading approach are competitive with each other.
- Results in 2D are much more pronounced than the ones in 3D. This is to be expected, since the amount of concurrent work scales as $\sqrt{N}$ in 2D versus $N^{2/3}$ in 3D for meshes with N elements.

4 CONCLUSIONS

We have described several different (but related) domain decomposition strategies employed to manage the concurrency issue of parallel deterministic transport sweeps. We have particularly focused on the technique of overloading, which has been somewhat ignored to date in the structured mesh case. We have described and implemented an unstructured generalization of overloading partitioning. We have also presented scaling results for both structured and unstructured sweep algorithms.

The results are very encouraging. Theoretical performance models have predicted improved efficiencies with structured overloading in comparison to contiguous volumetric partitioning in many circumstances; we have computationally observed this. We have also observed improved efficiencies for unstructured overloading. This improved efficiency is particularly important, since many computational transport techniques employ unstructured meshes and some structured scheduling algorithms have not been generalized for unstructured use. It is hoped that further work on structured overloading (e.g. performance models) may prove useful to unstructured algorithms.

There remains further work to be done. We need to obtain results at higher levels of parallelism and on other platforms, especially for 3D. We wish to explore variants of overloading that modify how the tiling is done. We want to explore the use of different aggregation techniques and scheduling heuristics. There is also a need for more sophisticated performance models that account for a fuller range of sweep scheduling algorithmic choices.

5 ACKNOWLEDGMENTS

We thank the various members of the PSAAP project in the Departments of Nuclear Engineering and Computer Science at Texas A&M University for fruitful discussions on this topic, in particular for working towards a common taxonomy and language for various sweep algorithms.

Work by the first author was conducted at Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Work by the second author was conducted at Lawrence Livermore National Laboratory. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

6 REFERENCES

1. R.S. Baker and K.R. Koch, "An Sn Algorithm for the Massively Parallel CM-200 Computer," *Nucl. Sci. Eng.*, **128**, p.312 (1998).
2. S. Plimpton, B. Hendrickson, S. Burns, and W. McLendon III, "Parallel Algorithms for Radiation Transport on Unstructured Grids," *Proc. SuperComputing 2000*, Dallas, Texas, November 4-10, 2000.
3. S.D. Pautz, "An Algorithm for Parallel S_n Sweeps on Unstructured Meshes," *Nucl. Sci. and Eng.*, **140**, pp. 111-136 (2002).
4. T.S. Bailey and R.D. Falgout, "Analysis of Massively Parallel Discrete-Ordinates Transport Sweep Algorithms with Collisions," *Proc. Int. Conf. on Mathematics, Computational Methods and Reactor Physics*, Saratoga Springs, NY, May 3-7, 2009.
5. M.P. Adams, M.L. Adams, W.D. Hawkins, T. Smith, L. Rauchwerger, N.M. Amato, T.S. Bailey, and R.D. Falgout, "Provably Optimal Parallel Transport Sweeps on Regular Grids," *Proc. Int. Conf. on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, Idaho, May 5-9, 2013.

6. B. Hendrickson and R. Leland, *The Chaco User's Guide, Version 2.0*, SAND95-2344, Sandia National Laboratories, Albuquerque, NM, July 1995.
7. S. Pautz, B. Bohnhoff, C. Drumm and W. Fan, "Parallel Discrete Ordinates Methods in the SCEPTRE Project," *Proc. Int. Conf. on Mathematics, Computational Methods and Reactor Physics*, Saratoga Springs, NY, May 3-7, 2009.