Finding Strongly Connected Components in Parallel in Particle Transport Sweeps *

Technical Report TR01-005 Department of Computer Science Texas A&M University

William C. McLendon III † Bruce Hendrickson ‡ Steve Plimpton §

Lawrence Rauchwerger ¶

3 April 2001

Abstract

Discrete ordinates methods are commonly used to simulate radiation transport for fire or weapons modeling. The computation proceeds by sweeping the flux across a grid. A particular cell can't be computed until all the cells immediately upwind of it are finished. If the directed dependence graph for the grid cells contains a cycle then sweeping methods will deadlock. This can happen in unstructured grids and time stepped problems where the grid is allowed to deform. In this paper we present a parallel algorithm to detect cycles in the dependence graphs present in these grids as well as an implementation and experimental results on shared and distributed memory machines.

*A preliminary version of this work will be presented at the 10th SIAM Conference on Parallel Processing for Scientific Computing in March 2001. This work was funded by the U.S. Department of Energy and was performed in part at Sandia National Labs, a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the U.S. DOE under contract number DE-AC-94AL85000 and at Texas A&M University. Research at Texas A&M University was also supported in part by NSF CAREER Award CCR-9734471, NSF Grant ACI-9872126, NSF Grant EIA-9975018, DOE ASCI ASAP Level 2 Grant B347886 and a Hewlett-Packard Equipment Grant.

†Department of Computer Science, Texas A&M University `mclendon@cs.tamu.edu`

‡Sandia National Laboratories, Albuquerque, NM. `bahendr@sandia.gov`

§Sandia National Laboratories, Albuquerque, NM. `sjplimp@sandia.gov`

¶Department of Computer Science, Texas A&M University `rwerger@cs.tamu.edu`

1 Introduction

Detailed multi-physics simulations are computationally expensive problems and thus require enormous computational resources, if they are to be executed in practical time. Such large computational platforms usually consist of distributed parallel systems which have to execute the codes in fully parallel mode to insure scalable performance. In this paper we will consider a prototypical radiation transport solver used in a ASCI multi-physics code. In this module the transport equations are solved using a sweep method. Sweeping methods used in radiation transport discretize the radiation field by angle, and flux propagation is computed for a set of discrete directions or ordinates. The computation for each angle is performed by sweeping the flux across a grid, i.e., a finite element mesh commonly used for fluids or shock hydrodynamics modeling. Radiation enters a mesh cell via faces whose outward normals point upwind, and exits through downwind faces. This implies an order of computation on the grid cells which, for a single ordinate direction, is represented as a directed dependence graph. Two example meshes and their associated dependence graphs for a particular angle are shown in Fig. 1.

Each of the (typically several hundred) ordinate directions induces an associated dependence graph. Sweeping methods will deadlock if any of the dependence graphs contains a cycle, such as the one in the dependence graph for the twisted grid shown in Fig. 1(b). Such situations occur frequently in 3-D unstructured grids and in multi-physics problems where the underlying discretized space (the mesh) deforms over time. To avoid deadlock, cycles in the set of ordinate dependence graphs must be detected and broken before the sweep can be performed. Typically, the application will use 'old' information for the edges removed from the dependence graphs, which will affect the convergence rate of the computation. Since the mesh elements (vertices of the dependence graph) are distributed across processors, we require a scalable parallel algorithm for cycle detection.

More formally, we are interested in finding all *strongly connected components* of a directed graph. A strongly connected component (SCC) is a maximal subset of vertices in which there is a directed path from any vertex to any other. Although the number of cycles in a graph can be exponential in the number of vertices, the number of SCCs is at most linear in the number of vertices, and typically quite small, since no vertex can be in more than one SCC.

Tarjan's classic serial algorithm for detection of SCCs runs linearly with respect to the number of edges and uses depth-first search [15]. However, depth-first search is known to be difficult to parallelize – the special case of lexicographical depth first search is P-Complete [11, 14], which in practical terms means it is unlikely that a scalable parallel algorithm exists.

There are some parallel algorithms for detecting SCCs that do not rely on depth first search.

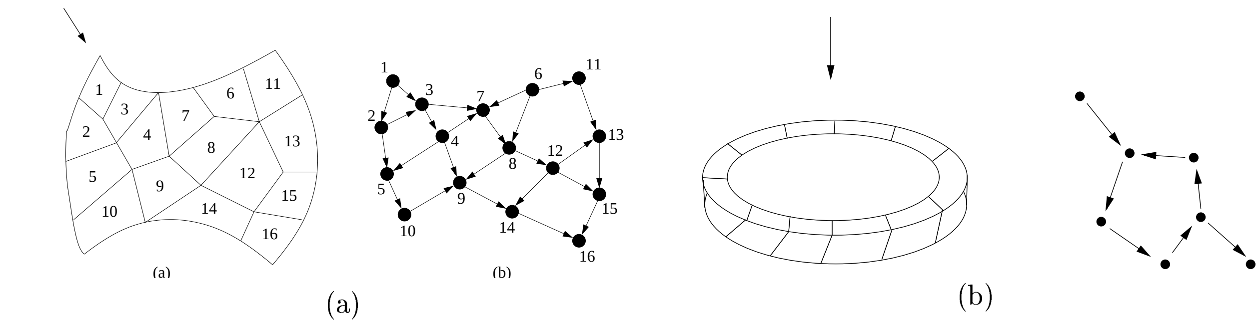


Figure 1: (a) An unstructured finite-element mesh (left) and its associated acyclic dependence graph for the angle shown (right). (b) A twisted ring of mesh elements that induces a cycle for the shown angle (left), and its dependence graph for the angle shown (right). A sweeping method will deadlock when it encounters a cycle such as this.

Gazit and Miller have an NC algorithm which can be used for locating SCCs that uses matrix multiplication [8]. Vishkin and Cole [5] and Amato [3] have proposed optimizations or extensions of this algorithm, but they still require $O(n^{2.376})$ processors and $O(\log^2 n)$ time where n is the number of vertices in the graph. An NC algorithm developed by Kao for planar graphs was developed requiring $O(\log^3 n)$ time and $n/\log n$ processors [10]. Another parallel algorithm for planar graphs is due to David Bader [4], but our applications are non–planar. There are also some parallel algorithms for related problems in directed graphs [9, 12], but they are not well suited for our application either due to their complexity or because they do not directly compute SCCs. In current work, we are studying whether some of these techniques might be adapted to compute SCCs.

In this paper we will describe our modification of a recently proposed parallel algorithm due to Fleischer *et al.* [7] and describe our implementation of it in MPI. We will also quantify its performance by presenting experimental results obtained on two different machines: A 16 processor HP V-class server and ASCI Red, which is a 9280 processor supercomputer maintained at Sandia National Laboratory.

2 The Modified DCSC Algorithm

The DCSC algorithm of Fleischer *et al.* [7] is a recursive, divide–and–conquer approach for finding the SCCs that does not rely on depth-first search. The main idea of DCSC is to recursively partition the dependence graph $G = (V, E)$ (a directed graph which may contain cycles) so that all SCCs will be entirely contained within a partition. The recursion stops when partitions contain either single vertices or SCCs. The partitioning is based on the following Lemma [7]:

Lemma 1 *Let $G = (V, E)$ be a directed graph, with $v \in V$ a vertex in G, and let $Pred(G, v)$ and $Succ(G, v)$ denote the set of predecessors and successors of v in G, respectively. Then, the unique SCC containing v in G, denoted $SCC(G, v)$, if one exists, is $Pred(G, v) \cap Succ(G, v)$. Moreover, any SCC of G is a subset of $Pred(G, v)$, $Succ(G, v)$, or $Rem(G, v) = V - \{Pred(G, v) \cup Succ(G, v)\}$.*

The DCSC algorithm described in [7] initiates the partitioning with a randomly chosen vertex $v \in V$. The expected serial complexity of DCSC is shown to be $O(|V| \log |V|)$ when all vertices in the graph have constant degree, which is true for the meshes arising in the transport computations.

Our ModifiedDCSC algorithm, outlined in Fig. 2, improves on the basic algorithm by performing a filtering or *trimming* step at the beginning of each iteration which reduces the size of the graph that must be processed. In particular, the *forward trim* performs a topological traversal of G, and all vertices visited in the process are removed from G. Recall that a topological traversal begins from a vertex with in-degree zero and visits vertices after all their ancestors have been visited, and produces a linear ordering (a topological sort) of the vertices of G such that all edges are directed left to right. Thus, no vertices on a cycle, or vertices reachable from a cycle, will be visited by a topological traversal. We say that vertices that are reachable from a cycle c are contained in the *shadow* of c. The forward trim of G removes all vertices from V that are not contained in cycles or in the shadow of some cycle. We also perform a *reverse trim* of G by a topological traversal of the graph $G = (V, \overline{E})$, in which the directions of all edges have been reversed (the edge reversals are conceptual only). The reverse trim also produces a shadow.

Fig. 4(a) illustrates how the forward and reverse trim steps remove nodes which are neither a part of nor are dependent on a cycle. A more abstract view is shown in Fig. 3(a). If the reverse trim encounters another cycle within the shadow of the forward trim, a second shadow will be cast by this cycle into the previous shadow. The overlapping shadow of the forward and reverse trims is called a *dark shadow*, and contains the vertices of G that must be further processed. The effectiveness of

Algorithm: ModifiedDCSC(G)

```
IF G is empty THEN return
TRIM G in forward direction
IF G is not empty THEN
   TRIM G in backward direction
   Select pivot v from the dark shadow of G
   MARK Pred(G,v) and Succ(G,v) in G
   SCC(G,v) = Pred(G,v) ∩ Succ(G,v)
   DO in parallel:
      ModifiedDCSC( Pred(G,v) − SCC(G,v) )
      ModifiedDCSC( Succ(G,v) − SCC(G,v) )
      ModifiedDCSC( Rem(G,v) )
ENDIF
```

Figure 2: ModifiedDCSC Algorithm.

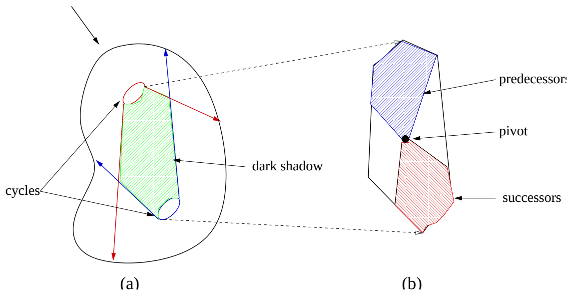


Figure 3: (a) An abstract 2D mesh containing two cycles (the circular rings). The TRIM steps remove the white regions, leaving the dark shadow (shaded). (b) The dark shadow is partitioned around the pivot into *Pred*, *Succ*, and *Rem* sets by the MARK step.

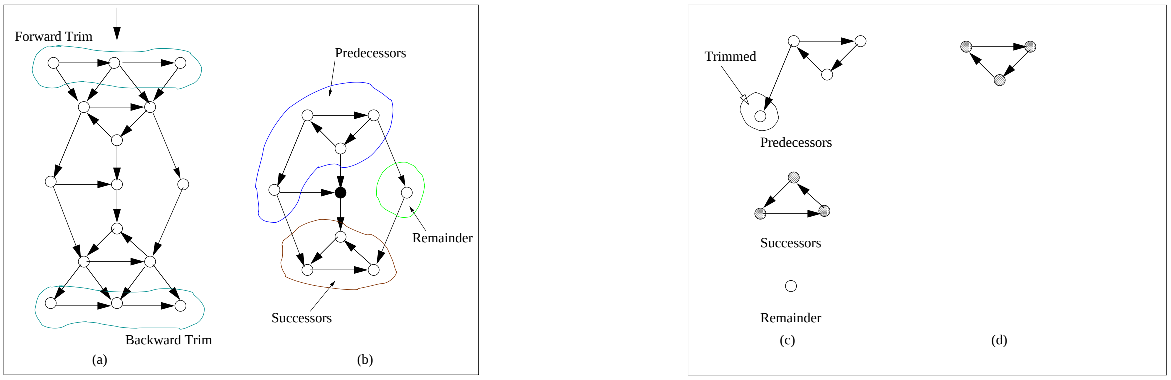


Figure 4: Example graph being trimmed in (a) by forward and backward trims. Part (b) shows the predecessor, successor, and remainder markings of the resulting dark shadow for the selected pivot (the solid vertex). Parts (c) and (d) show the subgraphs as they are reduced to strongly connected components.

the trim is dependent on how close the cycles are to the starting points of the forward and reverse topological traversals (vertices with in-degree or out-degree zero, respectively). If we encounter a cycle early in the traversal, the shadow will be large, thus reducing the effectiveness of the trim. For acyclic dependence graphs the forward trim step will eliminate all vertices in one sweep.

Following the trims, ModifiedDCSC applies DCSC to the remaining portion of the graph. A node, called *pivot* v, is selected from the *dark shadow*. Based on Lemma 1, the dark shadow is partitioned into four regions:

Pred(G,v) - Nodes from which the pivot v can be reached along some path.

Succ(G,v) - Nodes that can be reached along a path from the pivot v.

Rem(G,v) - Nodes that are neither predecessors nor successors of v (the remainder).

SCC(G,v) - Nodes that are both predecessors and successors, the (unique) SCC containing v.

The partitioning of the *dark shadow* into disjoint regions is illustrated in Fig. 3(b). An example of this partitioning on an actual dependence graph is shown in Fig. 4(b), where the graph is partitioned into three subgraphs that do not share any SCCs. Fig. 4(c-d) show two more iterations of ModifiedDCSC where the graphs are reduced to SCCs.

The *dark shadow* is partitioned by a *marking* step which traverses the dark-shadow region of the dependence graph forwards and backwards in BFS order starting from the pivot v. After marking is complete, if there are any nodes that are both predecessors and successors, they comprise a SCC containing the pivot v. The pivot v, and any SCC containing it, are extracted from the graph, partitioning it into three disjoint subgraphs containing $Pred(G,v)$, $Succ(G,v)$, and $Rem(G,v)$. The key observation of [7] is that any cycles remaining in the graph will be wholly contained within one of these regions. Thus we can call ModifiedDCSC recursively on each of the 3 new graphs. The recursion stops when all subgraphs contain either singleton vertices or SCCs.

3 Implementation

In our radiation transport solver, the numerical solution of the transport equations is obtained on a parallel machine by sweeping across many different ordinate angles simultaneously [13]. The cycle detection algorithm just described can exploit the same idea to achieve parallelism. In other words, ModifiedDCSC is executed simultaneously on a set of dependence graphs whose vertices are distributed across processors. This increases the complexity of the code and its resource requirements, but is critical for good parallel performance; otherwise BFS trims and marks on a single dependence graph would have limited inherent parallelism.

We have implemented the ModifiedDCSC algorithm in C with MPI. Our code takes as input the finite element grid and a list of ordinates (angles). ¿From this, we generate a dependence graph for each ordinate and store them as a list of distributed graphs. Note that since vertices correspond to finite elements, each dependence graph has the same set of vertices. In fact the set of edges is the same as well since these correspond to neighboring elements. But each dependence graph directs the edges differently. As elements are partitioned among processors for large problems, so are the vertices for all the dependence graphs. Processors store ghost node information for edges in a graph that link to a neighboring processor's node. Such information includes the processor ID of the owning processor, location of the ghost node in that processor's data structure, as well as marking and trimming status of the ghost node.

The pivot vertex selection is currently made randomly from the dark shadow (the vertices remaining after the trim operations). This is in accord with the algorithm presentation in [7] where the expected serial running time of DCSC is shown to be $O(n \log n)$ for graphs with constant degree. In current work, we are extending our implementation with different heuristics for selecting the pivot vertex.

3.1 Recomputing vs. storage of edges

ModifiedDCSC can traverse multiple times through a given node in the graph as it recursively breaks the graph into smaller pieces which means edges are traversed multiple times during execution. The memory required for the storage of all edge information can be significant because there are 6 edges per node (faces per element) in a hexahedral mesh. However, if edge directions are not precomputed each time an edge is encountered a non-trivial geometric computation must be performed. In our implementation, we chose to incur the memory cost of precomputing to reduce overall execution time.

3.2 Taking advantage of ordinate pairings

The work and memory requirements for the SCC search can be further reduced by taking advantage of the structural similarity between the dependence graphs for paired ordinates. We define an ordinate, o_1, to be paired with another ordinate, o_2, if $o_1 = -o_2$. *All* directed edges in the graph

Table 1: Mesh characteristics used in experiments (60 angles searched).

Mesh	Description	Size	Angles Checked	Total SCCs	Avg. Size of SCC
b42000	Brick with many, evenly distributed cycles	42875	60	4420	4.7
b64000	Larger version of b42000	64000	60	17437	5.4
s20	Area around a submarine hull Few, localized cycles	43984	60	4	20.0
sphere2	Solid spherical mesh. Few cycles near center	36712	60	1	8.0
warpcyl	Warped cylinder with concentric, stacked rings, many large cycles. Elements twisted by 18 degrees.	28000	60	280	400

for o_1 are flipped in relation to the corresponding edges in the graph for o_2. The elements in an SCC for o_1 will also be an SCC for o_2 because the cycle is preserved by reversing the edges connecting that cycle. It follows that we only need to search one angle from each pair to find the SCCs in both, thus reducing the work in half. This optimization also allows us to select one angle from each ordinate pair such that their distribution around the 3-d mesh is roughly uniform. This enables more processors to begin working quickly on the upwind end of one or more of the dependence graphs.

4 Experimental Results

The first system we have used for our experiments is a 16-Processor ccUMA HP-V2200 Server running HPUX 11. It has 4 GB total main memory with a 4 MB cache (2 MB Data / 2 MB Instruction) on each processor. Each processor is PA-RISC 8200 clocked at 200 MHz. We have used the HP proprietary MPI libraries.

The second system used is the Intel TeraFLOPS (ASCI Red) supercomputer at Sandia National Laboratories. It is a massively parallel distributed memory computer consisting of 4640 Nodes with 2 Intel Pentium Pro 333 MHz processors per node, or 9280 processors in total. Each processor has 32 KB L1 and a 512 KB L2 cache and 256 MB per node. ASCI Red uses proprietary message passing hardware with 310 Mbytes/sec bandwidth.

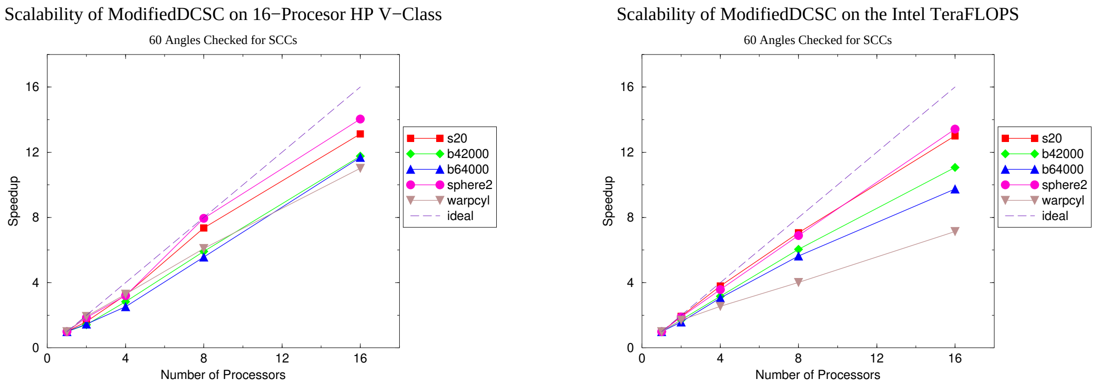
We conducted a series of experiments to illustrate the performance characteristics of Modified-DCSC. Specifically we investigated the scalability of our algorithm, mesh deformation dynamics, and message aggregation effects.

4.1 Effect of mesh structure

The meshes listed in Table 1 represent different geometries and shapes generated as finite element grids to represent a range physical models.

Fig. 5 shows the scalability of ModifiedDCSC on these meshes for (a) the HP computer and (b) ASCI Red. Fig. 6 shows the scalability on ASCI Red up to 64 processors. For these and subsequent speedup curves, we normalized against the single processor run time of ModifiedDCSC. In our experiments, the single processor ModifiedDCSC was usually at least as fast as Tarjan's algorithm for these problems.

Fig. 6 on ASCI Red shows that an increased number of SCCs (b42000, b64000, and warpcyl) reduces scalability. Because meshes with few SCCs benefit much more from trimming, the depth of the recursion of the algorithm is reduced, i.e., the number global steps is smaller allowing for a



(a) (b)

Figure 5: Scalability on **(a)** 16 processor HP V-Class Server and **(b)** ASCI Red.

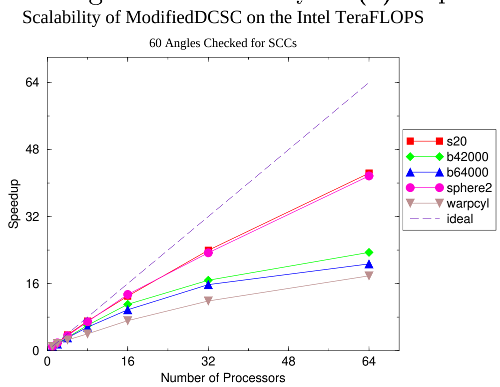


Figure 6: Scalability of ModifiedDCSC on ASCI Red up to 64 processors

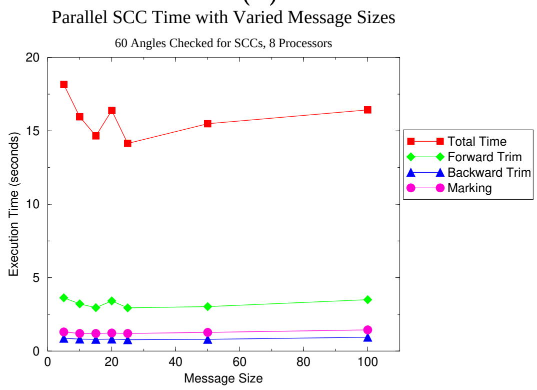


Figure 7: Effect of message aggregation on HP V-class for mesh b42000, 8 processors, 60 Angles.

larger speedup.

4.2 Varying degrees of mesh deformation

Multi-physics codes operate on meshes which can be slowly deformed at every time step. The result can be more cycles and thus reduced performance. We have simulated these changes by progressively increasing the magnitude of deformation of node position in the mesh. For this purpose we generated a $30x30x30$ brick mesh and moved the corner nodes of the cells in some random direction. The magnitude of deformation was increased in increments of 10% of the distance to the nearest corner node in a cell. Table 2 shows the information about the mesh used for this test and the increasing number of SCC's resulting from the deformation. We define the *Degree of Strong Connectivity* to be the ratio of the number of nodes involved in SCCs divided by the number of nodes in the graph. Summing these contributions over all ordinates gives a normalized measure of the amount of the graph involved in cycles.

Fig. 8 shows scalability of our algorithm on the deformed meshes both on (a) the HP and on (b) the ASCI Red machine. Table 3 shows the actual execution times for this experiment. It is important to note that the execution time of the ModifiedDCSC algorithm represents a fraction of the time needed to perform the actual sweep (the 'real' computation).

An increased displacement of corner nodes in the mesh causes an increased absolute number of SCCs as well as an increase in the average number of nodes contained in the SCCs. The higher

Table 2: Mesh characteristics as a function of deformation.

Mesh	Geometry	Deform % Magnitude	# SCCs Found in 60 Angles	Avg. SCC Size	Degree of strong connectivity
D_00	30^3	0	0	0	0
D_01	30^3	10	0	0	0
D_02	30^3	20	0	0	0
D_03	30^3	30	70	4.03	0.0002
D_04	30^3	40	899	4.31	0.0024
D_05	30^3	50	2701	4.62	0.0077
D_06	30^3	60	4825	5.02	.0150
D_07	30^3	70	7120	5.31	.0234

Table 3: Execution times for deformed meshes on the HP V-class and ASCI Red

	HP V class					ASCI Red				
Mesh	1	2	4	8	16	1	2	4	8	16
D_00	40.73	24.49	12.01	5.55	2.78	16.55	8.66	4.49	2.29	1.21
D_01	41.45	24.49	11.08	4.63	2.89	20.90	10.83	5.57	2.83	1.48
D_02	43.82	27.18	12.31	6.01	3.09	20.91	10.83	5.58	2.83	1.48
D_03	45.10	19.13	10.63	5.45	3.40	23.39	12.30	6.46	3.37	1.87
D_04	61.64	32.97	17.71	6.85	4.52	28.02	15.41	8.46	4.51	2.60
D_05	67.01	38.06	17.77	8.13	5.03	31.48	18.51	10.21	5.61	3.07
D_06	77.15	39.94	20.63	9.91	5.47	34.31	20.31	10.97	5.87	3.46
D_07	72.57	43.85	21.08	10.95	6.32	38.88	22.78	12.44	6.82	4.03

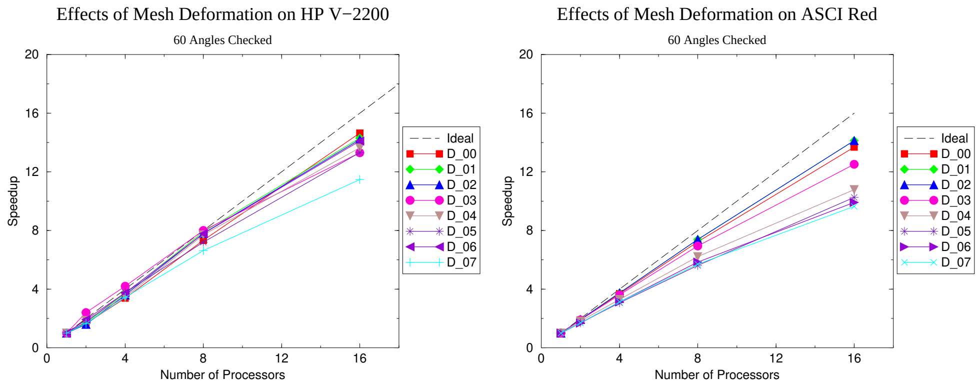
complexity of these meshes increases the amount of time ModifiedDCSC required for computing the full SCC search. The scalability of our heuristic improves as the trimming step can traverse deeper into the graph and involve more processors. When the trims are blocked before they can reach over several hops, fewer processors get involved and thus performance is poorer. In this case our technique relies much more on the more expensive marking step to isolate SCCs.

In Fig. 9(a) we show the effect of mesh deformation on the number of nodes contained in SCCs and their (negative) effect on the obtained speedup. Fig. 9(b) presents the absolute numbers obtained. Due to physical considerations in this experiment we have assumed a relatively small deformation and thus a small number of SCCs. We believe that our algorithm would not perform as well for stronger connected graphs.

4.3 Message Aggregation

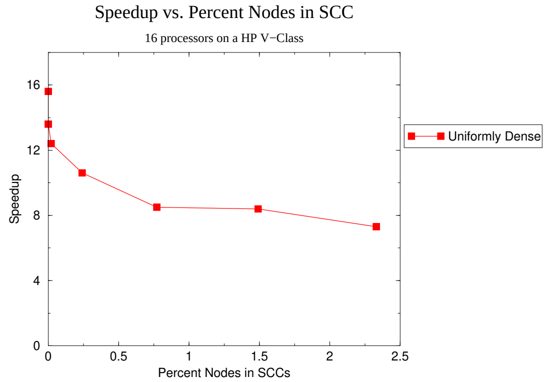
Communication inside the marking and trim steps is parameterized so that we can aggregate individual messages into larger messages for transmission. We tested the effects of sending smaller, more frequent messages versus less frequent, larger messages.

As the trim or mark routine encounters an edge crossing a processor boundary a message is inserted into a message buffer. This message buffer contains the destination processor and element which is on the other end of that edge. Changing the size of this message buffer affects the frequency of communications and their overhead. After experimentally varying the message size parameter on the b42000 mesh on 8 processors with 60 angles being searched we found that (Fig. 7) that the minimum execution times on 8 processors for the HP machine is obtained for a message size of 25.



(a) (b)

Figure 8: Scalability of ModifiedDCSC for increasing mesh deformations on (a) HP V-Class and (b) ASCI Red.



(a)

Mesh	# SCCs Found in 60 Angles	# Elements in SCCs	Speedup on 16 Processors
d_00	0	0	15.6
d_01	0	0	13.6
d_02	0	0	13.6
d_03	70	282	12.4
d_04	899	3879	10.6
d_05	2701	12487	8.5
d_06	4825	24218	8.4
d_07	7120	37809	7.3

(b)

Figure 9: Effect of uniform deformations of the mesh on the fraction of the total number of nodes that belong to SCCs and the resulting speedup.

5 Conclusions

We have presented the ModifiedDCSC algorithm, and its implementation, that offers a scalable method for detecting the strongly-connected components which arise in sweep calculations for radiation transport. We have studied the sensitivity of this algorithm to various characteristics of the input meshes. Not surprisingly, we found that scalability is negatively influenced by the number and density of SCCs of the graph. We have also explored various optimization parameters, e.g., message size at the application level.

In the future we intend to study more in depth the results of our mesh deformation experiment and possibly be able to predict the performance of ModifiedDCSC. Moreover, adaptive techniques could be employed to reduce the impact of small deformations occuring at successive timesteps of the multi-physics code.

References

[1] A. Aggarwal and R. J. Anderson, *A random NC algorithm for depth first search*, Combinatorica, 8 (1988), pp. 1–12.

[2] A. Aggarwal, R. J. Anderson, M.-Y. Kao, *Parallel depth-first search in general directed graphs*, SIAM J. Comput., 19 (1990), pp. 397–409.

[3] N.M. Amato, *Improved Processor Bounds for Parallel Algorithms for Weighted Digraphs*, Inform. Proces. Lett., 45 (1993), pp. 147–152.

[4] D. A. Bader, *A practical parallel algorithm for cycle detection in partitioned digraphs*, Tech. Rep. Technical Report AHPCC-TR-99-013, Electrical & Computer Eng. Dept., Univ. of New Mexico, Albuquerque, NM, 1999.

[5] R. Cole and U. Vishkin, *Faster optimal prefix sums and list ranking*, Information and Computation, 81 (1989), pp. 334–352.

[6] T. H. Cormen, C. E. Leiserson, and R. L Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Cambridge, MA, 1990.

[7] L. Fleischer, B. Hendrickson, A. Pinar, *On identifying strongly connected components in parallel*, Proc. Irregular'2000, Lecture Notes in Computer Science, Springer Verlag, 2000.

[8] H. Gazit and G. L. Miller, *An improved parallel algorithm that computes the BFS numbering of a directed graph*, Inform. Proces. Lett., 28 (1988), pp. 61–65.

[9] P. Gibbons, R. Karp, V. Ramachandran, D. Soroker, R. Tarjan, *Transitive Reduction in Parallel via Branchings*, Journal of Algorithms, **12** (1991), pp. 110-125.

[10] M. Y-Kao, *Linear-processor NC algorithms for planar directed graphs I: Strongly-connected components*, SIAM J. Comput., 22 (1993), pp. 431–459.

[11] R. Karp and V. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, Handbook of Theoretical Computer Science - Volume A, ed. J. Van Leeuwen, Elsevier Science Publishers/The MIT Press, Amsterdam, 1990, pp. 869–941.

[12] L. Lovász, *Computing Ears and Branchings in Parallel*, Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science (FOCS), (1985), pp. 464-467.

[13] S. Plimpton, B. Hendrickson, S. Burns, W. McLendon III, *Parallel algorithms for radiation transport on unstructured grids*, Proc. SuperComputing 2000 (SC2000), IEEE, 2000.

[14] J. H. Reif, *Depth-first search is inherently sequential*, Inform. Process. Lett., 20 (1985), pp. 229–234.

[15] R. E. Tarjan, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.