Identifying Strongly Connected Components in Parallel *

William C. McLendon III [†], ***Bruce Hendrickson*** [‡], ***Steve Plimpton*** [‡], ***Lawrence Rauchwerger*** [†]

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.

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

*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.

†Texas A&M University, Dept. of Computer Science, College Station, TX. Email {`mclendon, rwerger`}`@cs.tamu.edu`.

‡Sandia National Labs, Albuquerque, NM 87185-1110. Email {`bahendr, sjplimp`}`@sandia.gov`.

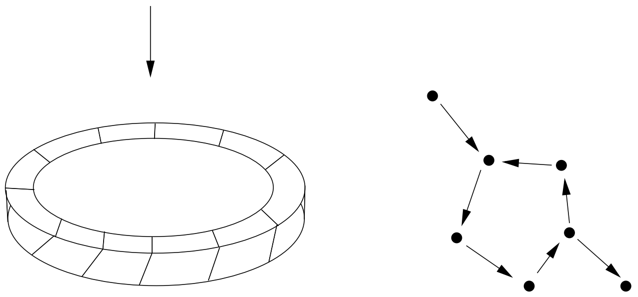


Figure 1. *A twisted ring of mesh elements that induces a cycle for the shown angle (left). Directed graph with a cycle (right). A sweeping method will deadlock when it encounters a cycle such as this.*

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. 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 situations occur frequently in 3-D unstructured grids and in multi-physics problems where the underlying discretized space (the mesh) deforms over time. Fig. 1 shows a twisted grid and the resulting cycle created in the dependence graph. To avoid deadlock, cycles in the set of ordinate dependence graphs must be detected and handled before the sweep can be performed. 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 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 [1]. However, depth-first search is known to be difficult to parallelize – the special case of lexicographical depth first search is P-Complete [7], which in practical terms means it will not scale well on a parallel machine.

There are some parallel algorithms for detecting SCCs that do not rely on depth first search. Gazit and Miller have an NC algorithm for locating SCCs that uses matrix multiplication [10]. Vishkin and Cole further improved this algorithm [9], but it is still requires ($O^{2.376}$) 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 [11]. Another parallel algorithm for planar graphs is due to David Bader [8], but our applications are non–planar.

In this paper we will describe a recently proposed parallel algorithm due to Fleischer, et al. [2] and describe our implementation of it in MPI. We will also quantify its performance by presenting experimental results obtained on two different machines: An HP V-class server and ASCI Red.

2 The Algorithm

The algorithm of Fleischer, et al. [2], DCSC, is a recursive, divide–and–conquer approach for finding the SCCs that does not rely on depth-first search. Its main idea is that by traversing a dependence graph in topological order forwards and backwards (akin to a breadth-first search or BFS) we can divide the graph into pieces in such a way that no SCC crosses between two pieces. We can then apply the algorithm recursively to each piece.

Our implementation improves on the algorithm presented in [2] by adding *trimming* steps. A *trim* is a topological sort of a dependence graph – vertices are visited after all of their ancestors have been visited. All vertices which can be visited in this way are in trivial SCCs consisting only of themselves. Thus the trimming phase reduces the size of the graph which must be searched for non-trivial SCCs. Fig. 3 sketches our ModifiedDCSC algorithm.

The trim step continues until either all nodes in the graph are removed or a cycle is encountered. Since we only remove nodes which have an in-degree of 0, cycles will block the traversal since the in-degree of a node can only reach 0 if that node is not in a cycle. Any node in the dependence graph that is dependent on a node in a cycle will also not be reached by the trim. We say that nodes which are not reached because they are dependent on nodes contained in a cycle are in the *shadow* of that cycle.

If the forward trim encounters a cycle it will trim all nodes it can reach. The resulting shadow is further trimmed in the backwards direction to remove as many nodes as possible from the graph. Similar to the forward trim, the backward trim repeatedly visits vertices all of whose descendants have been visited. If the backward trim encounters another cycle within the shadow of the forward trim a second shadow will be cast by this cycle into the previous shadow. We will refer to overlapping shadows from the forward and backward trims as a *dark shadow*. Any strongly connected components in the graph are fully contained inside the dark shadow region.

Fig. 2(a) illustrates how trimming the mesh up to the cycles can be used to eliminate nodes from the graph. The white spaces are the regions of the graph removed by one of the trim steps, while the dark region is not reached during either

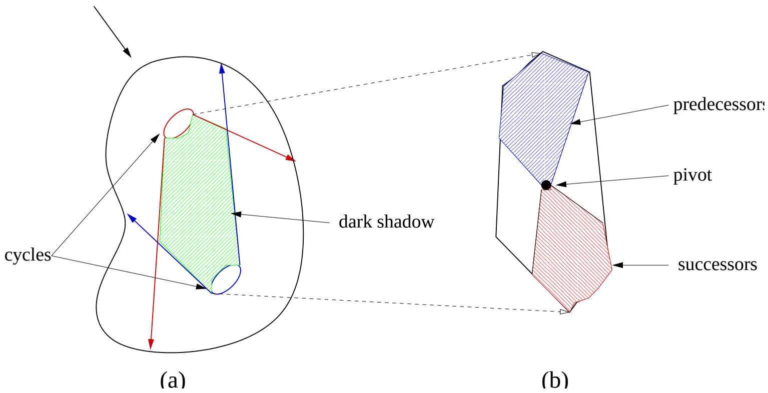


Figure 2. *A finite element mesh (2-d blob on* **(a)***) containing two cycles (the circular rings). The white spaces are the regions of the graph removed by the trim steps, while the* shadows *are the regions not reached during trimming. The* **(b)** *figure shows the pivot v, the effect of subsequent marking of its predecessor and successor nodes (the MARK step of ModifiedDCSC) and the partioning in 3 subgraphs of the* dark shadow.

trim.

The effectiveness of the trim is dependent on how close the cycles are to the starting points of the trim sweeps. 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 nodes in one sweep.

Following the trims, ModifiedDCSC applies DCSC to the remaining portion of the graph. It selects a node, called *pivot v*, from the *dark shadow*. Then the *dark shadow* is divided into 4 different regions, as illustrated in Fig. 2(b):
Predecessor (P) - Nodes from which the *pivot* can be reached along some path.
Successor (S) - Nodes that can be reached along a path from the *pivot*.
SCC - Nodes that are both predecessors and successors.
Remainder (R) - Nodes that are neither predecessors nor successors.

The *dark shadow* is decomposed into these partitions by a *marking* step which traverses the dark-shadow region of the dependence graph forwards and backwards in BFS order starting from the *pivot* node. After marking is complete, if there are any nodes that are both predecessors and successors, they comprise a SCC (including the pivot node). We extract the SCC from the graph which effectively splits it into 3 distinct subgraphs containing **predecessors**, **successors**, and the **remainder**. The key observation of [2] 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 algorithm proceeds until all SCCs have been found. Fleischer, et al. have shown that the expected time complexity of DCSC (without trimming) is $O(n \log n)$, where n is the number of edges in the dependence graph.

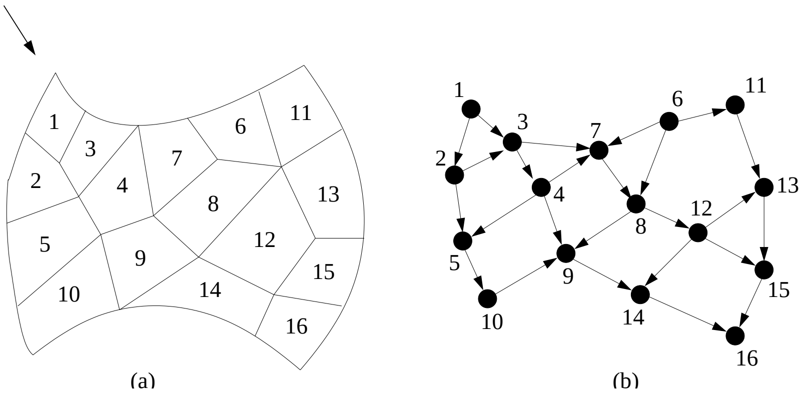


Figure 3. *An unstructured finite-element mesh (left) and its associated directed dependence graph (right).*

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 [3]. 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. Fig. 3 illustrates a 2-d finite element mesh and its associated dependence graph for a particular $(mesh, angle)$ pair.

We have implemented the ModifiedDCSC algorithm in C with MPI. Our code takes as input the finite element grid and a list of ordinate (angle). 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. Fig. 3 shows the recursive routine as it is implemented for a list of graphs G.

Algorithm 3.1 ModifiedDCSC

```
SCC-Search(Mesh, Ordinates)
ORDS = Select 1 ordinate from each pair
G = BuildGraphs(Mesh, ORDS)
ModifiedDCSC(G)

ModifiedDCSC(G)
IF every G_i is empty THEN return
TRIM every G_i in forward direction
IF some G_i is not empty THEN
    TRIM every G_i in backward direction
    Select pivot_i from the dark shadow of each G_i
    MARK predecessors and successors in every G_i
    SCC = predecessors(pivot_i) ∩ successors(pivot_i)
    G' = { P, S, R }
    ModifiedDCSC(G')
ENDIF
```

The pivot node selection is currently made randomly from the nodes remaining after a trim operation. This is in accord with the algorithm presentation in [2] where the expected running time of DCSC is shown to be $O(n \log n)$.

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.

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 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 by 1/2. 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.

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

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 for our experiments 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. 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.

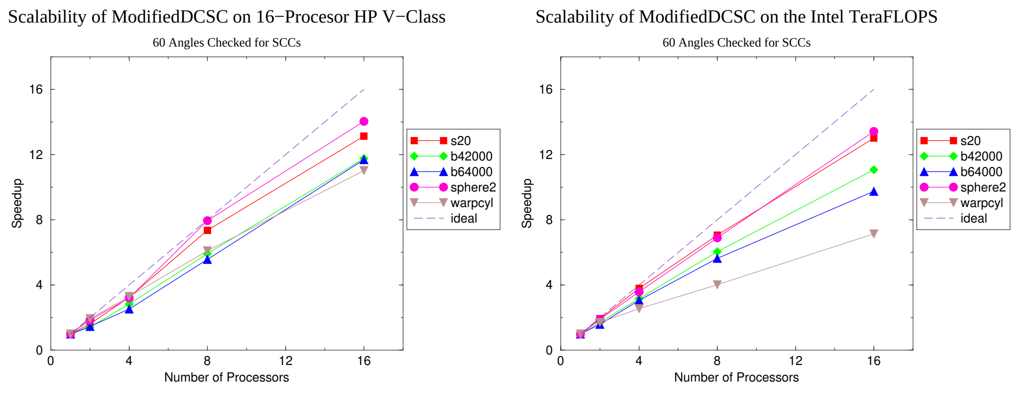
We conducted a series of experiments to illustrate the performance characteristics of ModifiedDCSC. Specifically we investigated the scalability of our algorithm, mesh deformation dynamics, and message aggregation effects.

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

Fig. 5 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 larger speedup.



(a) (b)

Figure 4. *Scalability on* **(a)** *16 processor HP V-Class Server and* **(b)** *ASCI Red.*

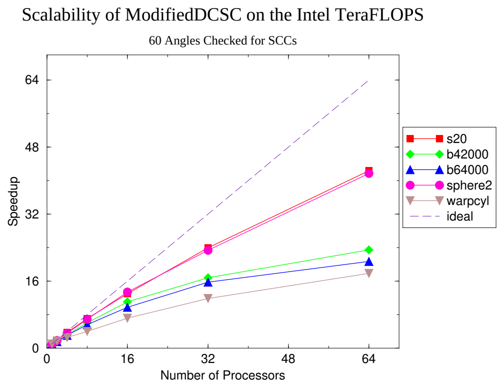


Figure 5. *Scalability of ModifiedDCSC on ASCI Red up to 64 processors*

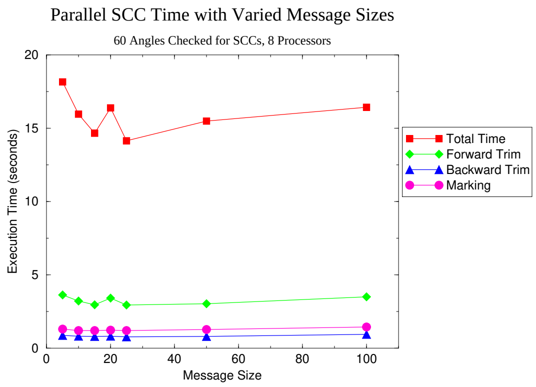


Figure 6. *Effect of message aggregation on HP V-class for mesh b42000, 8 processors, 60 Angles.*

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 non-trivial SCCs, divided by the number of nodes in the graph. Summing these

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

contributions over all ordinates gives a normalized measure of the amount of the graph involved in cycles.

Fig. 7(a) and (b) shows scalability of our algorithm on the deformed meshes both on the HP and on the ASCI Red machines. 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 of the average number of nodes contained in the SCCs. The higher 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.

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

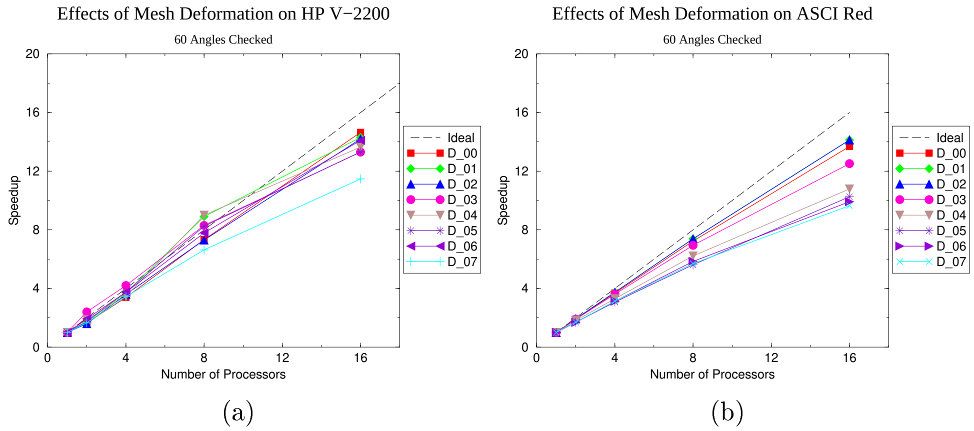


Figure 7. *Scalability of ModifiedDCSC for increasing mesh deformations on* **(a)** *HP V-Class and* **(b)** *on ASCI Red.*

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. 6) that the minimum execution times on 8 processors for the HP machine is obtained for a message size of 25.

5 Conclusions

We have presented an implementation of an algorithm 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 SCC's 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 deformation as time advances.

Bibliography

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

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

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

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

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

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

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

[8] 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.

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

[10] 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.

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