New Ordering Method for Implicit Mechanics and What It Means for Large Implicit Simulations

Roger Grimes, Cleve Ashcraft
Livermore Software Technology Corporation

Abstract

The most egregious serial bottleneck for Large Implicit Mechanics modeling for distributed memory parallel execution, independent of the application package, is the sparse matrix ordering for the direct matrix solution. LSTC is developing a new distributed memory ordering algorithm that is at least as effective as the serial algorithm METIS but is a fully scalable implementation. We will give an overview of the algorithm and the impact on some benchmark problems.

Problem Overview

Implicit solution methods for Finite Element modeling is dominated by the solution of the linear algebra problem Ku = f. The evolution has gone from band methods, frontal methods, envelope methods, and now the multi-frontal method. Since the 1960s there have been band minimization methods, envelope reduction methods, and frontal width minimization. These have been to reduce the computational resources for the numeric factorization. This is known as the symbolic processing or reordering phase which precedes the numeric factorization. Since the introduction of sparse matrix methods, especially the multi-frontal method, in the 1990's, the problem is still the same, how to reorder the matrix to reduce the computational cost for the numeric factorization. Since the mid 1990s the leading software for this problem has been Metis which is a public domain software package that is used by most commercial finite element packages for this reordering phase. Metis does a great job of quickly and robustly finding a permutation of the rows and columns of the matrix. But it does this using a global view of the matrix and is a serial implementation. That is the serial bottleneck for a fully scalable implementation of a direct linear equation solver in the distributed memory parallel environment.

The current implementation of LS-DYNA® Implicit uses a direct solution approach to solve the linear algebra problem based on 4 steps:

- Matrix Assembly and Constraint Processing
- Symbolic Processing
- Numeric Factorization
- Numeric Solution

The first and last 2 steps are fully scalable for the MPP implementation. The matrix is constructed and constraints processed to build a subset of the columns on each process. The numeric factorization and solution are also completely scalable. It is the symbolic factorization phase that is not.

The symbolic factorization phase constructs a compressed graph representation of linear algebra problem. This is close, but not quite the same, as the node and element connectivity of the finite element problem while accounting for contact and constraints. This compressed graph representation is a global data structure that has to exist in memory on at least one process. For every process that has enough memory to hold that global data structure and that has enough extra memory for the execution of Metis then executes Metis. Each execution has a random seed so each such process gets a different ordering. At the completion of Metis the processes then vote on who has the best. That ordering is then shared with all of the other processes. After this phase, all of the computations are fully scalable.

There are competing software packages to Metis. In addition, there is a parallel implementation of Metis. But none produces an ordering that has the same high quality results as Metis. A poor quality ordering causes a significant increase in the computational resources for the following numeric phase.

LSTC has such a good scalable matrix assembly, numeric factorization and solution phases that more computer time and computer storage are required for the symbolic factorization phase for large problems using lots of processes. This is the serial bottleneck that is the current limiting factor of LS-DYNA Implicit and all finite element software packages.

Our New Approach

LSTC starts with a matrix that is distributed across processes. We now have an enhanced process for building the compressed graph representation without using global data structures. This required recasting the compression algorithm as a sparse matrix multiplication and as an iterative algorithm.

We have gone back to algorithms from the 1970s that were used for the band width minimization, front width minimization, and envelope reduction methods. These were all based on using level structures. These methods relied on finding a pseudo-peripheral nodes that represent the extreme points of the adjacency structure. All nodes adjacent to one pseudo-peripheral node forms the first level of the level structure. The nodes adjacent to the first level form the second level. This continues until all nodes have been accounted for. The goal was to have a level structure as long as possible. For the multi-frontal method we take level near the middle of the level structure as a separator that cuts the problem into two halves of equal size. We also want the separator to have as few nodes as possible. And then apply that recursively.

But finding pseudo-peripheral nodes and level structures are expensive in MPP. And the 1970's level structures have difficulties with irregular geometries. LSTC has made several improvements. First we have extended these approaches to use "half-level" level structures to get better separators for irregular geometries. We have already filed for and have been granted a U.S. patent on this technology. We have also generalized the concept of pseudo-peripheral nodes to source points. We use a "sonar" like approach to find "source points" near the periphery. We then generate a level structure for each pair of source points. The cost of generating many level structures in the MPP environment is only slightly more than the cost of generating one level structure. This gives us a set of candidates for separators.

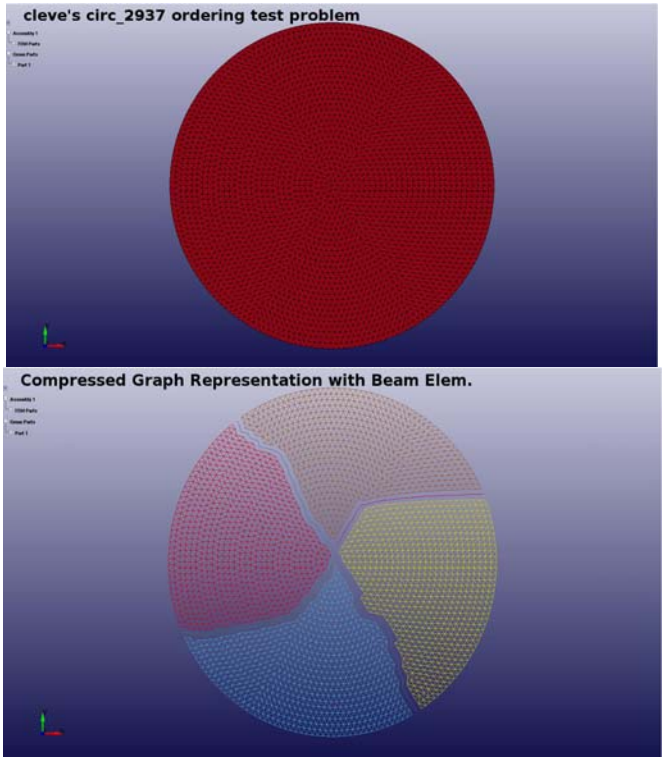
We now have a set of candidates for a separator. We also have made several simplifications to generate these candidates. So we have been developing algorithms to improve separators. This is akin to iterative approaches to solve a system of linear equations where you start with a good solution and make it better.

To date we have implemented one improvement algorithm in MPP and results based on that is shown later in this paper. We have a second improvement algorithm implemented and test in Matlab. We will be implementing that in the LS-DYNA MPP source in the near future.

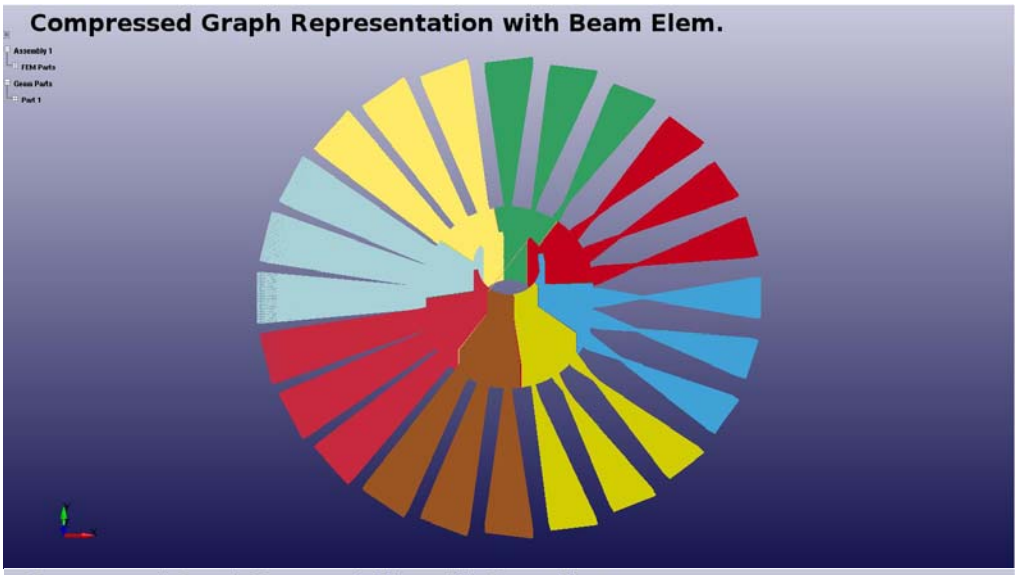
We apply this finding of a separator recursively to divide the original model into domains, one domain for each process. We then apply Metis to compute an ordering for each subdomain. Finally the separators are ordered appropriately to compute an ordering for the numerical factorization phase.

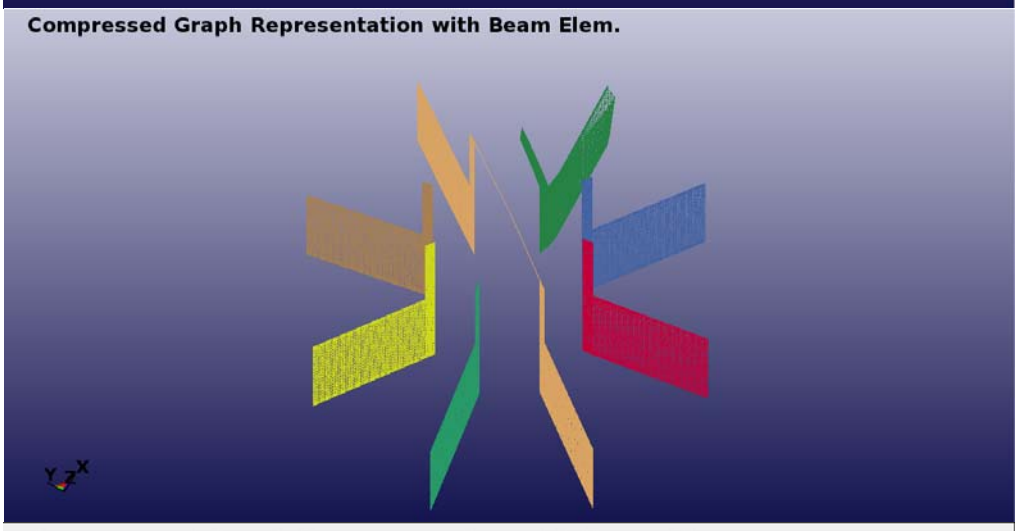
Examples

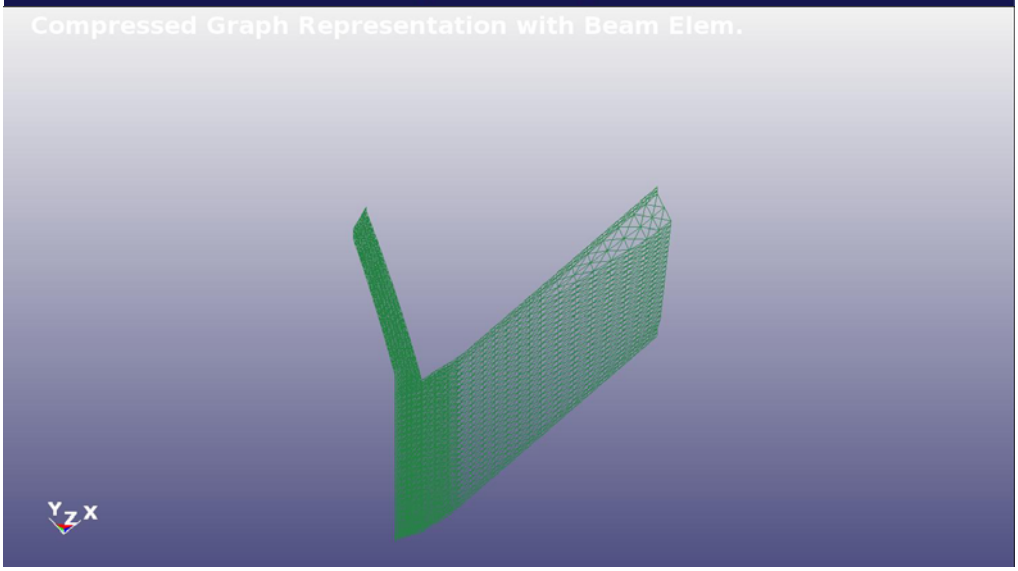
We will now show results on two examples. The first is a circle in a plane discretized with a triangular mesh. This was run using 4 processes in the MPP environment to divide the model into 4 domains. We can capture the ordering information and turn it into a keyword file that can be viewed by LSPrePost. This picture shows the 2 levels of separators and the 4 subdomains.



The second example is a fan blade model based on 104448 solid elements split into 8 domains using 8 MPP processes


Compressed Graph Representation with Beam Elem.


Compressed Graph Representation with Beam Elem.


Compressed Graph Representation with Beam Elem.

All of these separators are almost perfect except that last one highlighted. They cut the domain into two equal parts with perfect planar cuts. That highlighted separator has some bends. We expect the soon to be implemented improvement algorithm to fix this.

Our goal is a scalable algorithm that computes an ordering as good as Metis. Using the fan blade model and 2, 4, 8, and 16 processes we have a scaling comparison in storage and wall clock time

# of processes	Metis Storage	LSDYNA Storage	Metis WCT	LSDYNA WCT
2	435 Mb	491 Mb	29.0	206.6
4	435 Mb	209 Mb	28.8	36.4
8	435 Mb	156 Mb	29.2	88.8
16	435 Mb	108 Mb	31.1	94.7

Remember that the current implementation in LSDYNA is for development. And each increase in processes increases the number of sub-domains so the work increases. We see the storage scaling although not perfectly. The wall clock time is obfuscated by the fact there is a lot of debug output that is increasing the time. Nor is our implementation production harden as is Metis. But we are achieving our goals of scalability in storage. Importantly there are no global data structures. We expect the wall clock time to improve as we move towards production.

And for these problems, the ordering we are computing is actually better than Metis in reducing the computational requirements for the numeric factorization. We expect that this will reduce the wall clock time for the numeric factorization which will offset the increased wall clock time for the ordering.