Improvements for Implicit Linear Equation Solvers

Roger Grimes, Bob Lucas, Clement Weisbecker
Livermore Software Technology Corporation

Abstract

Solving large sparse linear systems of equations is often the computational bottleneck for implicit calculations. LSTC is both continuously improving its existing solvers, and continuously looking for new technology. This talk addresses both, describing improvements to the default distributed memory linear solver used by LSTC as well as promising new research. We discuss improvements to the symbolic preprocessing that reduce the occurrences of sparse matrix reordering, a sequential bottleneck and significant Amdahl fraction of the wall clock time for executions with large numbers of processors. Our new approach works for large Implicit models with large numbers of processes when the contact surfaces do not change, or only change slightly. We have also improved the performance of numerical factorization, most significantly with the introduction of a tiled, or two-dimensional distribution of frontal matrices to processors. This improves the computational efficiency of the factorization and also reduces the communication overhead. Finally, we present promising early results of research into using low-rank approximations to substantially decrease both the computational burden and the memory footprint required for sparse matrix factorization.

Reusing the Matrix Reordering and Symbolic Factorization

Solving a sparse linear system by factoring usually proceeds in four phases. In the first phase, the matrix is reordered to reduce the storage and operations that will be required to factor it. The second phase is a symbolic factorization, in which the elimination tree is created, and the numerical data structures are initialized. The third phase is the sparse factorization itself. And finally, solutions can be obtained by forward elimination and backward substitution.

Reordering to reduce fill-in and operations is a critical operation in sparse matrix factorization. Finding the optimal ordering, one which strictly minimizes storage or operations is provably NP-hard, so heuristics are used. The default used for large problems in LS-DYNA® is Metis, a weighted nested dissection based code.

Historically, reordering and symbolic factorization took relatively little time compared to factorization. However, parallel scaling of the factorization phase has been studied for over three decades, and substantial progress has been made, whereas reordering remains a largely

sequential process. As LS-DYNA is run on increasingly large numbers of processors, the time for reordering has emerged as the dominant aspect of many problems. A relatively small example is the one million element NCAC Silverado model. Running on a 16-core workstation, reordering and symbolic factorization takes 23 seconds whereas sparse matrix factorization takes only 15. An LS-DYNA implicit user reported that on a larger, proprietary model, reordering and symbolic preprocessing took 1917 seconds, while factorization, running on 768 cores, took only 142. Over a simulation with hundreds of implicit time steps this represents a huge potential reduction in execution time.

LSTC is working on addressing the reordering problem with a new scalable algorithm. But even when that is available, we will want to minimize the number of times that we reorder the matrix. Towards that end, we have modified the code that permutes and redistributes the input matrix to allow it to use a previous ordering. If off-diagonal coefficients are dropped from the matrix, they are simply filled in as zeroes. If new off-diagonal coefficients are added to the matrix, then we have to check to see if they are consistent with the previous symbolic factorization. If they are, the previous ordering is reused, even though it is likely suboptimal, and reordering is avoided.

The important key to reusing the reordering is predicting any additional nonzeroes added to the stiffness matrix due to slight changes in contact. Consider the following model. As the blade moves outward due to centrifugal forces the blade slides slightly outward. If we can predict this slight change and add appropriate terms in the stiffness matrix the reordering can be reused. The ability to activate this new feature is on the new keyword *CONTROL_IMPLICIT_ORDERING.

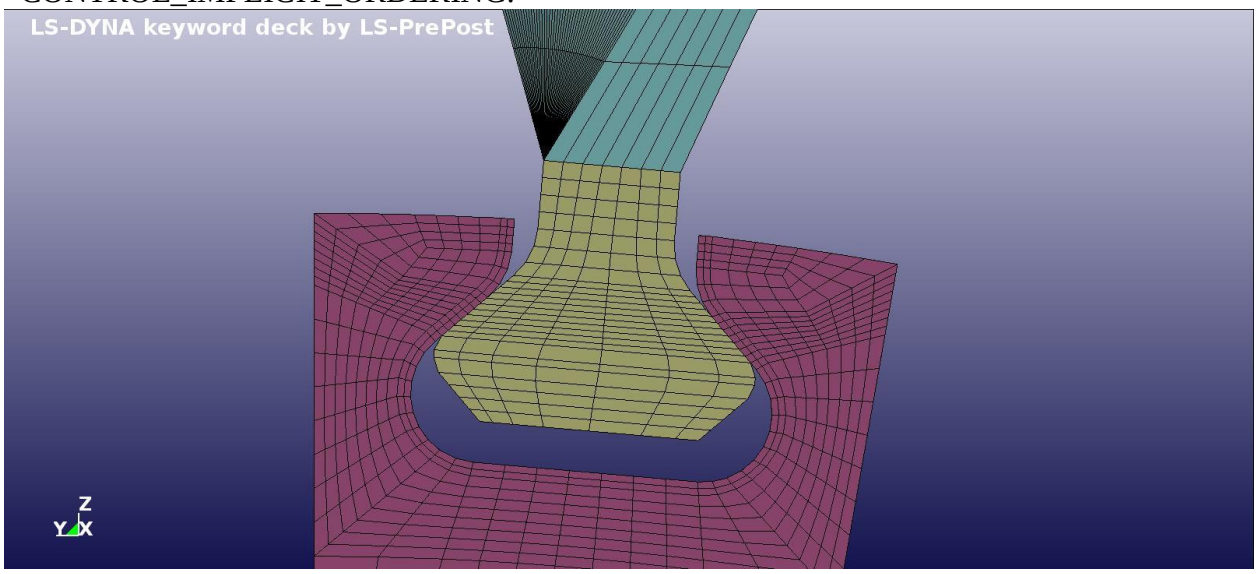

Figure 1

Savings from reusing the sparse matrix reordering can be significant. But the savings is only for large models using both many processes MPI and many implicit time steps where the reordering can be reused.

Performance Improvements to the Numerical Factorization

Research into parallel multifrontal codes goes back to the early 1980s, and distributed memory multifrontal codes were first implemented in 1986. LSTC has been using distributed memory versions of the multifrontal method since the introduction of scalable implicit processing at the turn of the century. We use a subtree-subcube technique to distribute the branches of the elimination tree to different processors. Near the root of the tree, where there are more processors than supernodes, then multiple processors are assign to the factorization of each of these MPP frontal matrices.

When the code for the MPP frontal matrices was first written, large-scale implicit users had O(10) processors. Designing to scale well to 32 seemed to provide plenty of head room for future growth. It was known in the mid-1980s that distributing the matrix by columns could scale to 100 processors. Furthermore, this one-dimensional distribution localizes evaluation of the ratio of each diagonal value to its off-diagonals, which is needed to determine if pivoting is required. Therefore, LSTC has been using 1D MPP frontal matrices for nearly two decades. Even for jobs with over 100 processors, where performance could tail off for the supernodes near the root of the elimination tree, their children or grandchildren would benefit from additional processors, and the overall factorization would speed up.

Today, given the exponential growth in the number of cores that we are experiencing, LS-DYNA jobs are now running on thousands of cores. To enable sparse matrix factorization to continue to scale, LSTC has had to add a new generation of tiled, or two-dimensional frontal matrices. The processors are organized into a N by M Cartesian grid, where N*M is less than, or equal, to the number of processors. The frontal matrix is then distributed across the 2D grid of processes. When the factored pivot columns are distributed, there are N simultaneous MPI broadcasts, each amongst M processors. Each of these broadcasts transmits 1/N the amount of data a 1D broadcast would convey. A down side is that pivot rows also have to be broadcast, but surprisingly, the 2D factorization kernel can be faster than the 1D kernel on as few as 4 processors using a 2 by 2 grid.

Figure 2 depicts the relative performance of 1D versus 2D kernels factoring a simulated symmetric frontal matrix. The plateaus in the 2D curve reflect processor totals that don't map to a two-dimensional grid, with an acceptable aspect ratio. Examples are 13 and 14, which are turned into a 3 by 4 grid, with the extra processors idled. Forming the 2D frontal matrices is more complicated than their 1D counterparts, so LS-DYNA doesn't switch to 2D unless there are at least eight MPI ranks assigned to the supernode.

As more and more cores are brought to bear, the advantage of 2D frontal matrices increases. Of course, the vast majority of supernodes in the elimination tree are assigned to individual processors, and their throughput is unchanged by the 2D kernels. Still, the frontal matrix of the root supernode is disproportionately large, so the overall impact is still significant. Consider a 3D implicit model supplied by an LS-DYNA user, which requires 10.57 quadrillion (10^15) floating point operations to factor. When run on all 128 cores of an 8-node, dual-socket, 8-core cluster, constraining LS-DYNA to 1D frontal matrices requires 7124 seconds for the first factorization. Enabling 2D frontal matrices brings that down to 4931 seconds. The root supernode had 137259 equations, and 1D throughput was 754 Gflop/s whereas 2D throughput was 1862 GFlop/s.

<<<< a chart goes here but I cannot get it to stay >>>>>>

Figure 2

Block Low Rank Approximation

There is no limit to the imagination of LS-DYNA users, and the size and complexity of the analyzes they need to perform. Unfortunately, sparse matrix factorization scales superlinearly, both in terms of memory and operations. This scaling threatens to be a constraint on the ability of LS-DYNA users to achieve their objectives. To address that, LSTC is examining the use of block low rank approximations (BLR) to the factors of the matrix.

Over the past twenty years, many theoretical studies have been carried out on data-sparse algorithms. They allow for algebraically compressing the numerical data involved in linear systems by taking advantage of the properties of the underlying elliptic operator, at the price of a controllable loss of accuracy. This process transforms a dense block into a product of much smaller blocks: the low-rank representation. These blocks, in turn, allows for faster matrix-matrix products when they are combined.

Many low-rank representations have been proposed in the literature. Some of them are recursive (the so-called hierarchical representations) and obtained through complex compression algorithms. Even though theory has shown that this is the most efficient kind of representations, they are often not flexible enough to be used in pre-existing modern, parallel, fully featured direct solvers. For this reason, simpler representations have been designed in order to compromise between efficiency, ease of use and numerical stability. The Block Low-Rank (BLR) format has been shown to fulfill these requirements and it is implemented in a research version of the multifrontal solver MUMPS. A BLR representation of a dense block is obtained through a rank-revealing factorization and consist of a product of a tall and skinny matrix with a short and wide matrix. The classic multifrontal algorithm has to be adapted to handle BLR blocks in a way that pays off in as many stages of the factorization, both in terms of memory footprint and operation count.

MUMPS BLR is coupled to LS-DYNA as a research tool to evaluate the potential of low-rank technologies on our implicit problems. Preliminary studies have shown promising results both in terms of performance and accuracy. On a 100x100x100 rubber simulation problem (yielding a linear system of 3 millions of unknowns), the size of the factors is reduced by a factor of 2, and the number of operations to perform the factorization is reduced by 5. Because operations on low-rank blocks are of smaller granularity, they are also slower. Consequently, the operation reduction translates to a speed-up of only 2 for the overall factorization CPU time, including the overhead due to low-rank compressions. Another decisive property of this new approach is that the larger the problem, the better the relative efficiency of the low-rank approach. For instance, on the same problem but with a 50x50x50 model, the operation count was reduced by a factor of 4 instead of 5 for the larger equivalent model.

Summary

LSTC is continuously improving LS-DYNA, adding new capabilities. It is also addressing the impact on existing software of changes in the computing platforms available to our users. This talk highlighted three examples of this process, applied to multifrontal linear solvers. We have reduced the need to reorder the sparse matrix when small changes are made to the model.
We have developed a new frontal matrix factorization kernel, that exploits MPI Cartesian meshes to reduce communication overheads and improve load balance. And finally, we are exploring the utility of block low rank approximations, which offer the promise substantial reductions in the storage and operations required to solve a sparse linear system of equations.