Improving LSTC's Multifrontal Linear Solver

Roger Grimes[3], Robert Lucas[3], Nick Meng[2], Francois-Henry Rouet[3],
Clement Weisbecker[3], and Ting-Ting Zhu[1]
[1]Cray Incorporated
[2]Intel Corporation
[3]Livermore Software Technology Corporation

1 Overview

Implicit time steps in LS-DYNA require the solution of large, sparse systems of linear equations. The multifrontal method of Duff and Reid is a particularly attractive algorithm for directly solving such linear systems as it transforms the sparse matrix factorization into a hierarchy of dense matrix factorizations. The vast majority of the floating-point operations can be performed with calls to highly tuned BLAS3 routines, and near peak throughput is expected. Such computations are performed today on clusters of multicore microprocessors.

The computational complexity of factorization means it has historically been the bottleneck for implicit calculations, both in terms of the storage used as well as elapsed time. Because of its importance to LS-DYNA, research into how to improve multifrontal solvers is a continuous effort, and this paper discusses some of the recent highlights. Section 2 discusses the impact of changes in microprocessor architecture and the emergence of new technology for solid-state disks. Section 3 discusses how we are improving the performance of factorization on large-scale distributed memory systems. Section 4 describes research efforts into distributed reordering and Block Low-Rank approximation that should enable LS-DYNA to handle even larger implicit models in the near future.

2 Adapting to computing system evolution

The end of Dennard scaling has forced an end to microprocessor clock frequency scaling. Nevertheless, microprocessor vendors have continued to increase the peak arithmetic processing performance of individual processors by expanding their architectures to include single-instruction/multiple-data (SIMD) function units. Intel's Advanced Vector Extensions (AVX) allow one Pentium processor to retire four fused multiply-add operations per cycle, and even eight of them on the Xeon Phi. These SIMD operations are most effectively utilized by calling BLAS3 functions from mathematical libraries, such as Intel's MKL, and LSTC's multifrontal code now makes extensive use of the matrix-matrix multiplication (e.g., DGEMM) and triangular solve (e.g., DTRSM) routines.

Unfortunately, the performance of standard Fortran and C code has plateaued along with the clock frequency. Therefore, small overheads that used to seem irrelevant when compared with the time invested in dense matrix arithmetic kernels are increasingly visible. For example, if a symmetric matrix is stored as a triangle, then accumulating the output of a matrix-matrix multiplication which was performed by DGEMM requires reformatting the output from a rectilinear array, to a trianglur one. For problems derived from shell elements, this could take as much as 5% of the runtime of sparse matrix factorization. We are addressing such problems by revisiting our arithmetic kernels, and doing things like opportunistically storing symmetric matrices as full ones, when storage is available. This allows operations like DGEMM to be performed in place, reducing unnecessary data movement.

New memory technology is being brought to market, and of particular interest is the 3D XPoint (3DX) technology recently announced by Intel and Micron. This is a persistent memory that is cheaper and denser than DRAM and faster than FLASH, though more expensive. Intel is initially bringing it to market as a faster solid-state disk (SSD). As depicted below in Figure 1, the impact on the performance of the linear solver when operating out-of-core is dramatic. Normally, the triangular solves are starved for bandwidth, and when out-of-core can approach the factorization's elapsed time (top row), even though they do orders-of-magnitude less arithmetic work. With the new 3DX SSD, they perform almost as if in-core (bottom row). Given the relative cost of DRAM versus the new SSDs, users may wish to rethink how they configure their computing systems.

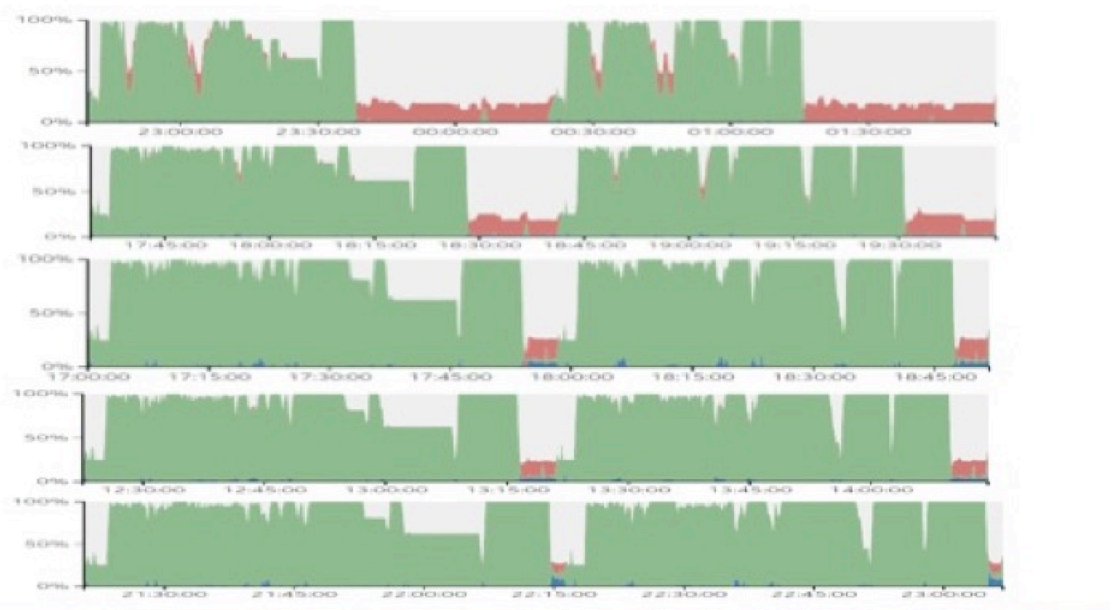

Figure 1: Performance impact of 3DX SSD on LS-DYNA implicit running out-of-core. From top to bottom: Hard disk, FLASH SSD, 3DX SSD, FLASH RAID, 3DX RAID

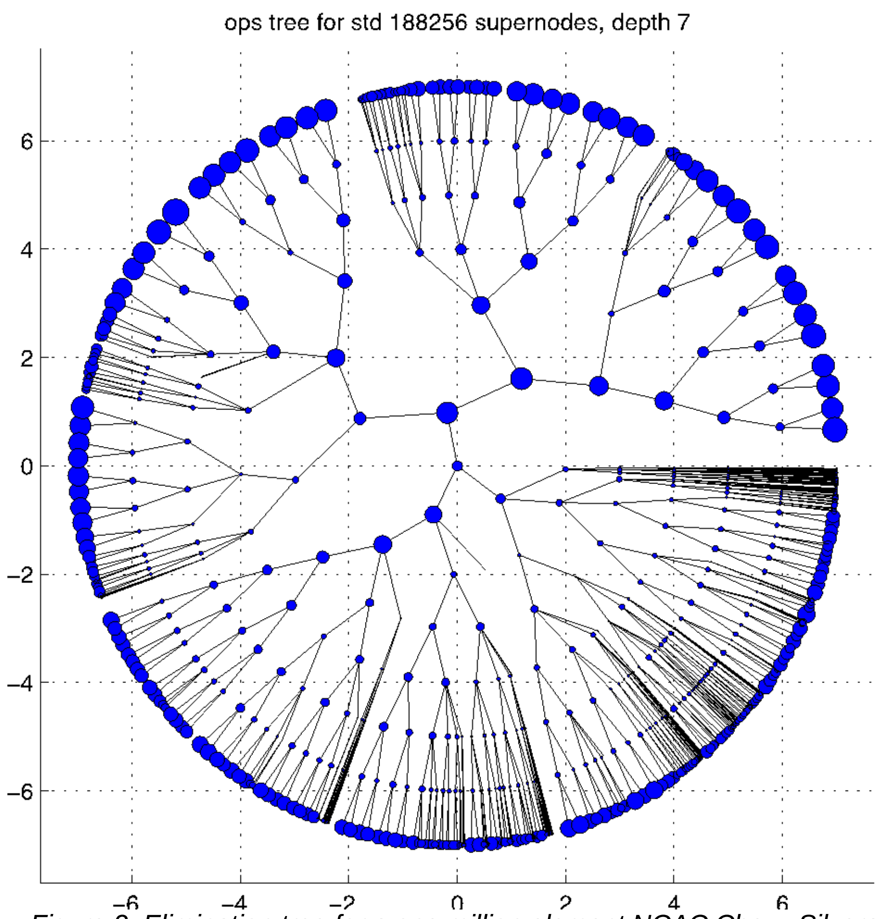

Figure 2: Elimination tree for a one million element NCAC Chevy Silverado model.

3 Processor scaling

LSTC has been using distributed memory versions of the multifrontal method since the introduction of scalable implicit processing at the turn of the century. The multifrontal method turns a sparse matrix factorization into a tree of dense factorizations. Figure 2 (above) depicts such a tree, derived from a model of a Chevrolet Silverado pickup truck. Each circle represents a dense frontal matrix, and is scaled to reflect the relative amount of arithmetic worked needed to factor it. We use a subtree-subcube technique to distribute the branches of our elimination tree to different processors. Near the root of the tree, where there are more processors than branches, multiple processors are assigned to the factorization of each of the individual frontal matrix.

When the code for the distributed 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. Cleve Moler knew in the mid-1980s that distributing the matrix by columns could scale to 100 processors. Furthermore, this panel-based (1D) 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 distributed frontal matrices for nearly two decades. Even for jobs with over 100 processors, where performance could tail off for the frontal matrices near the root of the 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 2D, 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 assigned to the 2D frontal matrix. 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 8 processors, a 2 by 4 grid. Figures 3 and 4 depict the relative performance of 1D versus 2D kernels factoring a simulated symmetric frontal matrix on a large-scale Cray XC40 system.

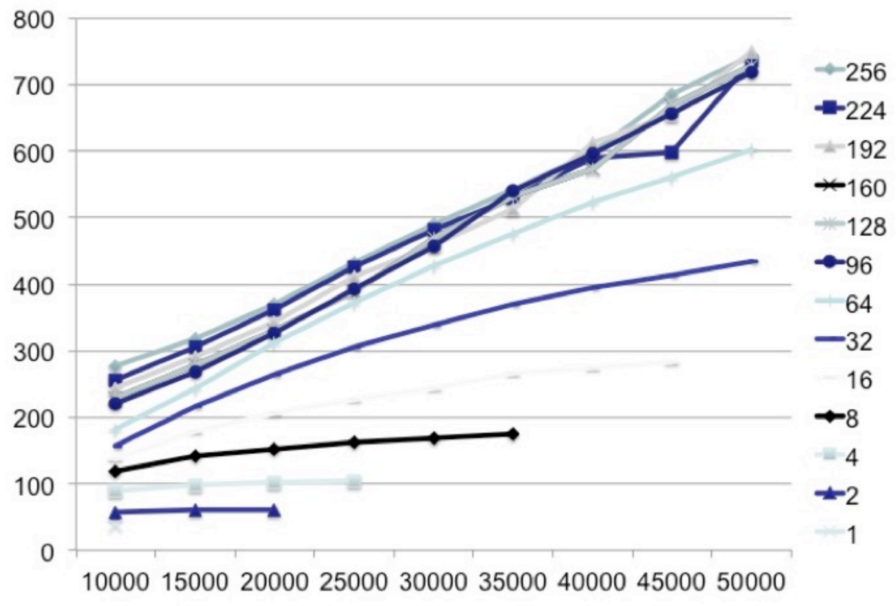

Figure 3: Scaling of a panel-based (1D) symmetric factorization kernel on a Cray XC40 GFlop/s (Y) vs. dense matrix rank (X) on varying numbers of processors

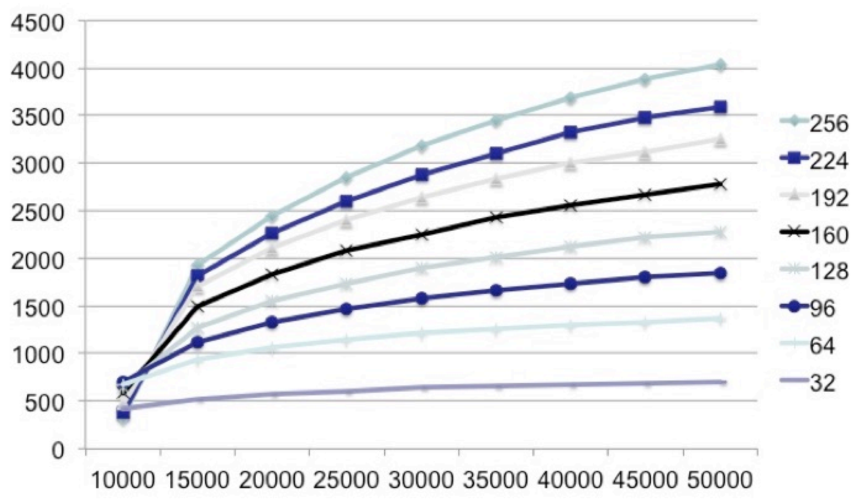

Figure 4: Scaling of a tile-based (2D) symmetric factorization kernel on a Cray XC40 GFlop/s (Y) vs. dense matrix rank (X) on varying numbers of processors

4 Towards larger models

LS-DYNA users are creating ever-larger models, and in today's computing environment, that requires increasing numbers of processors to store and process them. LSTC's multifrontal solver was originally written over two decades ago to run on Cray mainframes. On such machines, reordering to reduce the size and operation count needed for factorization was a modest overhead, O(1%) of the total run time. Now, with as few as a dozen AVX-accelerated processors, reordering can be the bottleneck, both in terms of time elapsed as well as the memory required on any one processor.

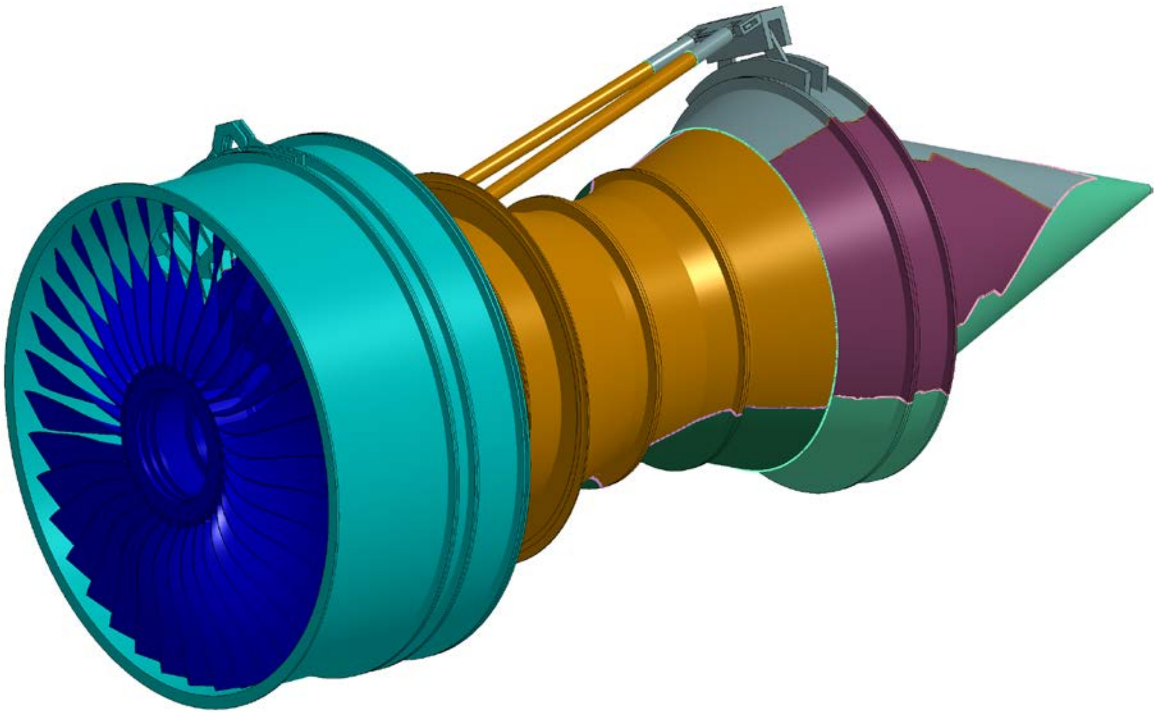

Figure 5: Nested dissection of a dummy engine model supplied by Rolls Royce

The first step towards fixing this is to develop a new scalable algorithm for graph partitioning, the heart of a nested dissection reordering. This was presented at the 2016 LS-DYNA User's Conference, and LSTC has developed it into a new distributed memory, nested dissection heuristic. This new reordering code has been integrated into LS-DYNA and is undergoing testing. Figure 5 depicts its results when applied to a dummy engine model created by Rolls Royce. This model has almost 200 million equations, and the reordering was performed on a 8-node, 128-core, Linux cluster. The successful deployment of a scalable reordering function is the first step towards eliminating the last vestiges of the sequential bottleneck in LSTC's symbolic preprocessing.

Of course, an effective parallel ordering will enable even larger implicit models to be solved, and there is no limit to the imagination of LS-DYNA users, and the size and complexity of the analyses 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.

One way to address this is to replace blocks of coefficients within the factors with low-rank approximations (block low-rank approximations, or BLR). These low-rank approximations are created using rank-revealing QR factorization. BLR introduces error into the factorization, and we can bound its magnitude for any particular block being compressed. This is an idea that we have explored in the past, and dismissed. However, as models get bigger and research improves the algorithms, we think it is time to revisit that decision. To facilitate this work, LSTC joined the MUMPS consortium. We have added BLR to our multifrontal code (sequential only to date), and are using it to evaluate matrix compression and the overall impact that the error it introduces has on finite element models. We are also in the process of extending it to save operations, as well as storage.

Figure 6 illustrates the impact of BLR on the solution of four implicit models, as a function of the amount of error tolerated in the low-rank approximations. The accuracy of the solution (and indirectly the convergence speed of the underlying non-linear process) can be controlled directly by the user through an explicit numerical parameter called the low-rank threshold. Increasing the threshold decreases the accuracy of the solution and increases the memory savings. Early results, depicted in Figure 6, show that the memory footprint of the factorization can be substantially decreased, while preserving a convergence rate close to the full-rank case. The optimal choice is a trade-off between the two and can be problem dependent, which is both an inconvenience and an advantage, since it provides more control over the numerical behavior of the solver and more flexibility.

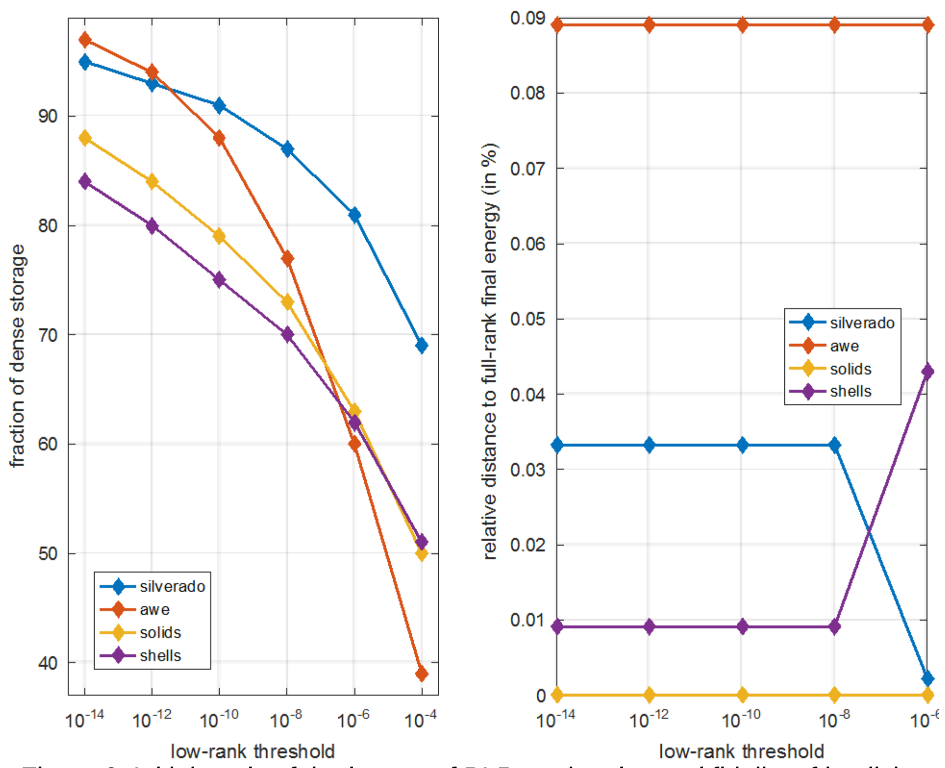

Figure 6: Initial study of the impact of BLR on the size and fidelity of implicit models.

5 Summary

LSTC and its partners, including Cray and Intel, are continuously improving LS-DYNA, and adding new capabilities. We are also adapting to changes in the computing platforms available to our users. This talk highlighted three examples of this process, applied to multifrontal linear solvers. We have modified to code to exploit new technology such as SIMD arithmetic processing units, and demonstrated significant performance improvements, both in and out-of-core. We have developed a new 2D frontal matrix factorization kernel that exploits MPI Cartesian meshes to reduce communication overheads and improve load balance. We are deploying a new distributed reordering algorithm that will reduce our peak memory requirement, allowing larger implicit models. And finally, we are reconsidering 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.