Increasing the Scale of LS-DYNA® Implicit Analysis

Cleve Ashcraft[2], Jef Dawson[1], Roger Grimes[2], Erman Guleryuz[3],
Seid Koric[3], Robert Lucas[2], James Ong[4], Francois-Henry Rouet[2],
Todd Simons[4], and Ting-Ting Zhu[1]
Cray[1],
Livermore Software Technology Corporation[2],
National Center for Supercomputing Applications[3],
Rolls-Royce[4]

Abstract

Cray, LSTC, NCSA, and Rolls-Royce formed a partnership to explore the future of implicit computations as the scale of both finite element models and the systems they run on increase. Rolls-Royce created a family of dummy engine models, using solid elements, with as many as 200,000,000 degrees of freedom. NCSA ran these with specialized LS-DYNA variants, generated by Cray, on their Blue Waters machine, a hybrid Cray XE/XK system with 360,000 AMD cores. Processing and memory bottlenecks revealed themselves as the number of processors increased by an order-of-magnitude beyond that familiar to today's developers and users, and LSTC made improvements to LS-DYNA. This talk will discuss the challenges encountered, enhancements made to LS-DYNA, and the results when extending the limits both in terms of the scale of the model, and the number of processors. This is ongoing work, and we will conclude by discussing the path forward that has been illuminated.

Introduction

Only time and resource constraints limit the size and complexity of the implicit analyses that LS-DYNA users would like to perform. Therefore, Cray, LSTC, NCSA, and Rolls-Royce have formed a partnership to explore the future of implicit analyses as both the finite element models and the systems they run on increase in scale. This paper reports on the challenges encountered, how some have been overcome, and the lessons learned. This ongoing project could not have been conducted alone by any of the institutions involved, as each brings unique skills and assets to the partnership. While many of the scaling issues were known to LS_DYNA's developers, and had simply not yet been addressed due to the relative priority of other issues, we also discovered unknown problems, ones that only manifested themselves when processing hundreds of millions of equations, on thousands of cores.

This paper begins by discussing the most pressing challenge encountered, the need to be able to reorder extremely large sparse matrices to reduce factorization storage and operations. This led to many broader changes in LS-DYNA, as well as the discovery of surprising bugs in familiar software. We then present initial results derived when running a smaller, 105M DOF version of the dummy engine model, scaling to as many as 16,384 cores on Blue Waters. This unveiled to us three scaling bottlenecks, one hitherto unknown, which are now being addressed. Finally, we discuss our ongoing work, plans for the near future, and summarize.

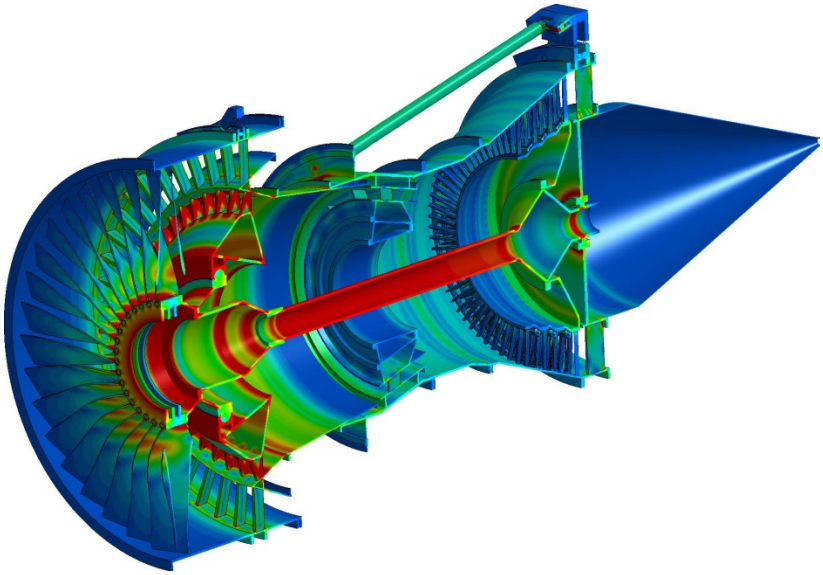

Figure 1
A cross-section of the the Rolls-Royce Dummy Engine Model.

The Reordering Challenge

Rolls-Royce created a family of dummy engine models that could be shared with their collaborators, to push the boundaries of implicit analyses, both in terms of the scale of the model, and the scale of the computing system. An initial implicit load calculation with the large model (200M DOF) was performed on an internal Rolls-Royce server. The default reordering method, nested dissection [George 1973] by Metis [Karypis 1995], failed, so multiple minimum degree (MMD) [Liu 1985] was used instead. The resulting job ran out-of-core, and took 160 hours to complete on a 16-node, 224-core, Linux cluster. While a milestone in terms of the scale of the model, the turnaround time was excessive, and Rolls-Royce invited its colleagues (Cray, NCSA, and LSTC) to demonstrate using the Blue Waters system, with its 360K cores, that such analyses could be performed in a shorter period of time, suitable for engineering applications.

Nested dissection is normally a more effective reordering for factoring the large linear systems used in implicit finite element analysis, and LS-DYNA has used Metis for many years. Therefore, we made an initial effort to get Metis to reorder a smaller version of the dummy engine model (105M DOF) on Blue Waters. The jobs crashed, in a manner that indicated memory was being overrun on the 64 GByte nodes of Blue Waters.

Metis dynamically allocates its storage, while LS-DYNA historically allocated the memory used for the linear solver statically, and it was largely unused while Metis executed. To increase efficiency by allowing the same memory to be utilized for both reordering and the linear solver, LSTC modified LS-DYNA to dynamically allocate and free storage, as needed, during implicit analyses. To enable the freeing and reallocation of large volumes of memory, too large to coexist in core, any existing state is temporarily saved in a scratch file. We discovered, much to our surprise, that if files are too large, one does not always read what one wrote. To overcome this, we had to adopt a strategy of reading and writing in blocks of modest size. Dynamic memory allocation, to make more efficient use of memory, was subsequently released for general use in LS-DYNA R11.

Thinking that available memory was still a constraint, we modified LS-DYNA to be able to checkpoint the ordering on one machine, and later read it on another. NCSA researchers then tried to run LS-DYNA, using Metis to reorder, on the Bridges system at the Pittsburg Supercomputing Center, to exploit its 3 TByte large memory nodes. Unfortunately, this failed as well.

Researchers at LSTC had long been exploring LS-GPart, an alternative nested dissection strategy based on half-level sets [Ashcraft 2016]. We next turned to this, integrating the prototype code into LS-DYNA. After exploring a rich parameter space, we eventually settled on an effective heuristic whose results are competitive with Metis. Frustratingly, it ran on every model we tested it with, except the large dummy engine model. With a great deal of effort, we ultimately determined that the random number generator that our ORDRPACK software was using to compress the indistinguishable vertices in the graph of the matrix had a period smaller than 200M. Once we wrote a new random number generator, we were finally able to use LS-GPart to reorder even the largest of the dummy engine models, as shown in Figure 2, which depicts the dummy engine partitioned into eight domains.

Both the random number generator problem and the file read/write inconsistency are examples of familiar software failing at scales inconceivable to those who wrote them, often decades prior. We fear that as models continue to grow in the future, many more such problems will manifest themselves.

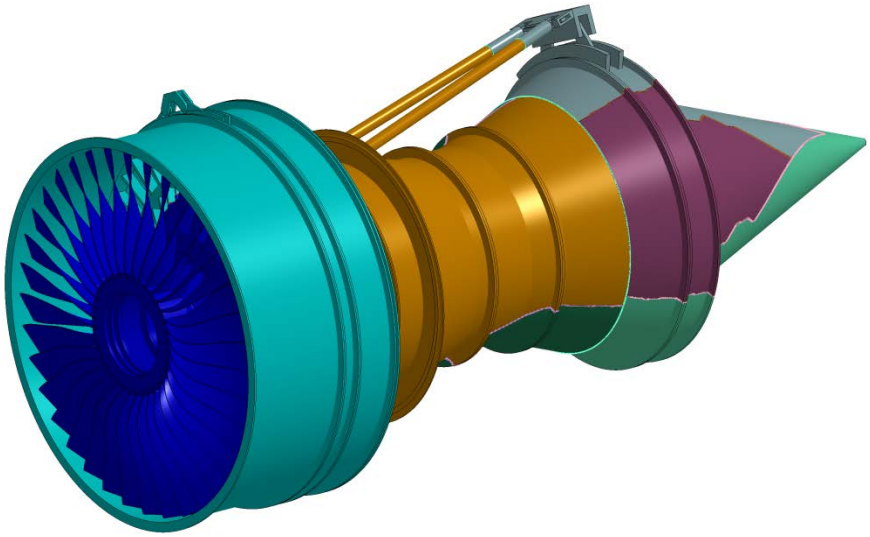

Figure 2
Eight-way partition of the Rolls-Royce Dummy Engine Model.

Initial Results with the Small Engine Model

Once LS-GPart had been effectively integrated into LS-DYNA, we were able to execute the small dummy engine model on Blue Waters, and begin our analysis to the scaling of LS-DYNA. Figure 3a shows the time consumed by two of the more time consuming aspects of the computation, reordering the 105M DOF engine model with LS-GPart and then sparse matrix factorization with LSTC's multifrontal linear solver. There are eight threads per MPI rank, and the x-axis depicts the total number of threads employed, which equals the number of cores. LS-GPart starts out taking relatively long for a reordering code, but scales well, and will do even better once OpenMP directives are added. Performance of the multifrontal code flattens out beyond 8000 threads. Figure 3b shows their relative scaling, normalized to 1024 threads.16,396 cores is a scale well beyond that accessible to developers at LSTC, or similar small independent software vendors, so the developers of these codes have never before had the opportunity to observe performance at this scale, much less optimize for it.

Given our previous experience with the scaling of a similar multifrontal code [Koric 2016], we are optimistic that ours can be significantly improved.

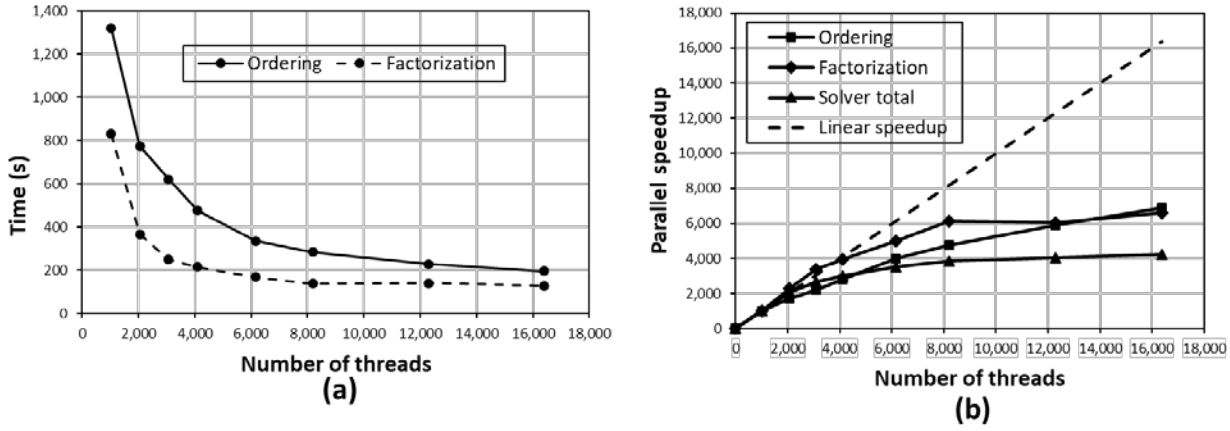

Figure 3
Scaling of reordering and sparse matrix factorization versus threads on Blue Waters.

As discussed above, memory consumption is a first order concern. NCSA developed a real-time tool that runs in the background and queries each node of Blue Waters to determine how much memory is available. This helped LSTC to identify areas of peak memory utilization in LS-DYNA. Examples are show below, in Figure 4, which illustrates how memory is allocated and freed as LS-DYNA moves from one phase of the computation, to another. Results are presented for 2048, 4096, 8192, and 16384 threads, eight threads per MPI rank.

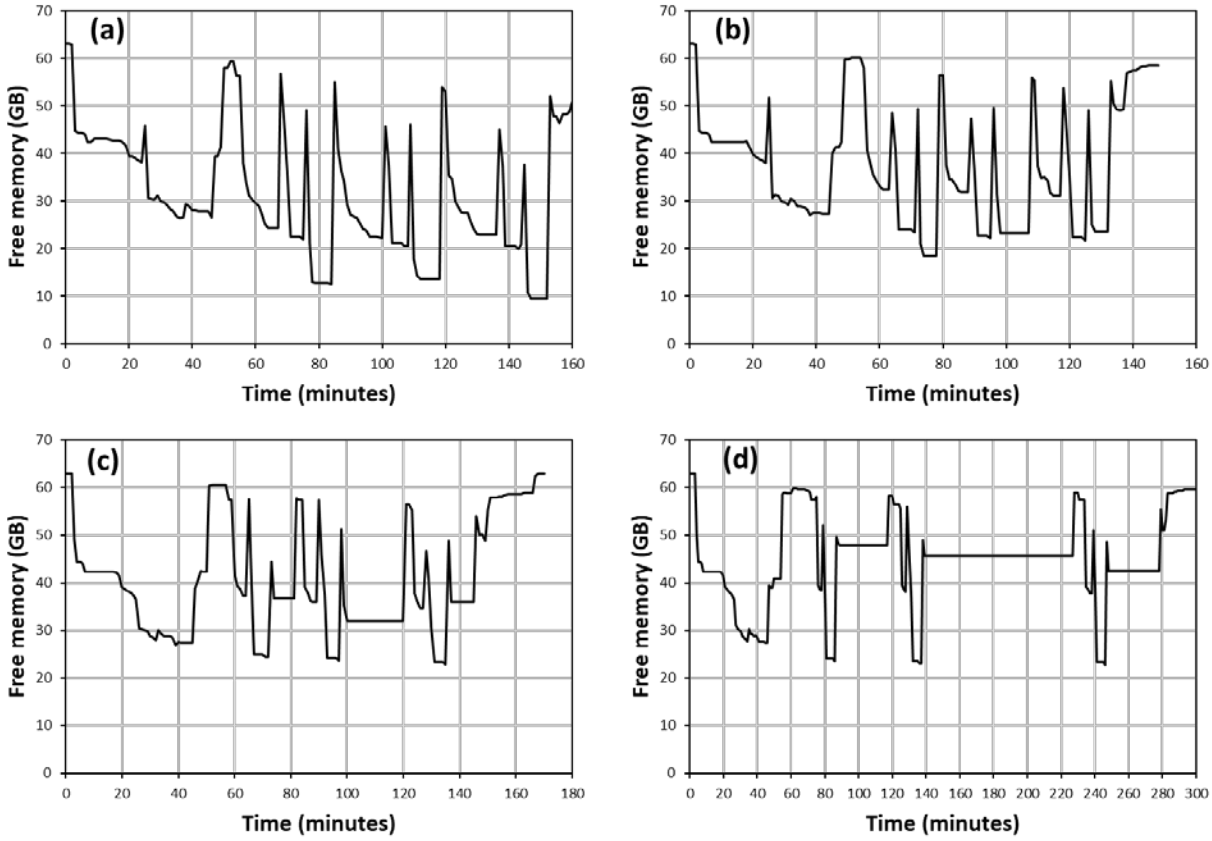

Figure 4
Available memory versus time when running the small dummy engine model on Blue Waters. The total threads employed are 2048 (a), 4096 (b), 8192 (c), and 16384 (d). There are two MPI ranks, each with eight threads, on the Blue Waters nodes.

The 16,384-thread run in Figure 4d highlights the impact of three sequential bottlenecks, input processing by MPI rank 0 (the first 60 minutes), symbolic factorization (the available memory minimums), and the constraint processing (the three flat plateaus). Input processing ought to be done off-line, however that software had not kept up with changes in LS-DYNA implicit, and had to be updated. In the future, this will no longer be a significant bottleneck for large-scale parallel runs, idling thousands of cores for an hour. Symbolic factorization has long been done sequentially in LS-DYNA, which was rationalized by the fact that reordering with Metis was also sequential, and almost always took significantly longer. The scalability of LS-GPart led to symbolic factorization times exceeding reordering, and development of a scalable symbolic factorization is now well under way. Finally, the "plateaus" in Figure 4d are constraint processing. Presently done in serial, the CPU time for constraint processing (both analysis and computations) had previously been noise in the overall run time of LS-DYNA. However, as we increase the number of MPI ranks to 1024 or more on Blue Waters, the noise becomes deafening.

Ongoing Work with the Large Engine Model

Once LS-GPart was integrated into LS-DYNA, and we could work around the problems we were having with Metis, we turned our attention back to the large dummy engine model. An attempt to run it on Blue Waters failed due to a memory allocation failure. To help understand how much memory is necessary, Cray made a system available for testing with newer nodes, containing 192 GB of memory per node, three times that of a typical Blue Waters node. Using 64 such nodes, the large dummy engine model ran, in-core, in 12 hours.

Once we had the results of the in-core run at Cray, we were able to determine that input processing on MPI rank 0 required more than 64 GB, the memory of a standard Blue Waters node. NCSA then tried again, this time pinning MPI rank 0 to one of Blue Water's 96 large memory nodes, which have 128 GB. This enabled LS-DYNA to parse the large dummy engine model, and successfully compete one sparse linear solve, before the job was halted. Figure 5 below shows the available memory on MPI rank 0, as a function of time. The minimum is during the input processing, when over 80 GB are in use. The symbolic factorization sequential bottleneck requires the second most memory (1:10 – 1:30 AM), and barely fits in 64 GB.

Only one Blue Waters run with the large dummy engine model was initiated before our 2017 allocation was exhausted. In April 2018, we will be able to renew our investigation of the scaling of LS-DYNA on Blue Waters. We will offload the input processing, and distribute the symbolic factorization and constraint processing. This should allow us to finally run the large model to completion on Blue Waters, and begin a campaign to study and improve its parallel performance. We have little doubt that once we can observe the performance of the arithmetic kernels on first one, and then two orders-of-magnitude more cores than are available to developers at LSTC, we will be able to significantly improve their parallel scaling.

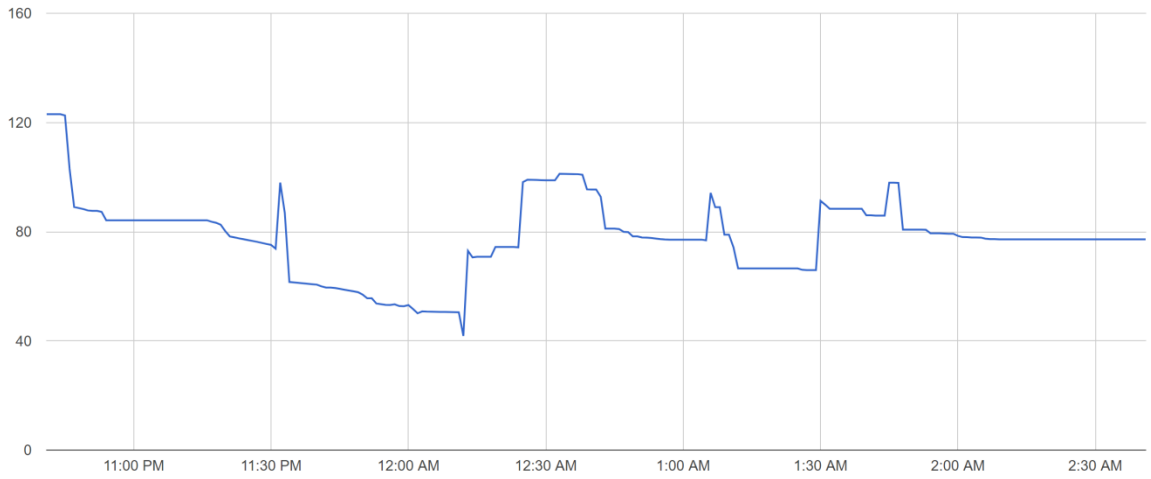

Figure 5
Available memory versus time on MPI rank 0 when running the large dummy engine model on 8096 cores of Blue Waters. There were 1024 MPI ranks, each with eight threads, one per Blue Waters node.

Summary

Rolls-Royce challenged Cray, LSTC, and NCSA to demonstrate that implicit finite element analyses of large-scale models could be performed in a timely manner using large-scale computing systems. The engine models created for this study are, to our knowledge, the largest implicit models ever run with LS-DYNA. They led to the discovery of numerous problems in LS-DYNA and the familiar software infrastructure that it utilizes. It accelerated the transition of LS-DYNA to dynamic memory allocation and the deployment of the LS-GPart nested dissection heuristic. After two years of effort, we have been able to run the model in-core, with more processors and a nested dissection ordering, which together yielded a 13-fold reduction in overall run time. We are now at a stage where we can begin studying the scaling behavior of LS-DYNA, identifying and overcoming performance bottlenecks. This should make LS-DYNA suitable for use on even larger models, and larger computing systems.

References

[Tinney 1967] W. F. Tinney and J. W. Walker, Direct solution of sparse network equations by optimally ordered triangular factorization, Proceedings of IEEE, 55 (1967), pp. 1801-1809.

Liu 1985] J. W. H. Liu, Modification of the minimum degree algorithm by multiple elimination, ACM Transactions on Mathematical Software, 11 (1985), pp. 141-153.

[George 1973] J. A. George, Nested dissection of a regular finite element mesh, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345-363.

[Karypis 1995] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, Proceedings of the 1995 International Conference on Parallel Processing, 1995, pp. 113-122

[Ashcraft 2016] C. Ashcraft and F.-H. Rouet, A global, distributed ordering library, SIAM Workshop on Scientific Computing, 2016.

[Koric 2016] Koric S. and Gupta A. Sparse Matrix Factorization in the Implicit Finite Element Method on Petascale Architecture. Computer Methods in Applied Mechanics and Engineering, 2016 v.32:281-292