Projecting Performance of LS-DYNA Implicit for Large Multiprocessor Systems

Pramod Rustagi and Ilya Sharapov
Sun Microsystems, Inc.
Menlo Park, CA

Abstract

We use a benchmark dataset of a jet-engine impeller to study the performance characteristics of the LS-DYNA Implicit solver, as a function model size and the number of processors available in the system. We analyze a range of models with an increasing number of fans in the jet-engine impeller, from the smallest model of three fans composed of 300,000 nodes to the largest model of ten fans with 1 Million nodes. The resulting stiffness matrices are of sizes .9 and 3 Million Degrees of Freedom (DOF) respectively.

These sizes are typical for today's LS-DYNA runs, but in the coming five years, model sizes will to grow upwards of 50 Million DOF. In addition, the computational system available at that time will offer much higher degree of parallelism. In this paper we estimate the performance and scalability of large LS-DYNA runs on these future machines.

This material is based upon work supported by DARPA under Contract No. NBCH3039002. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or DOI/NBC.

Introduction

LS-DYNA [3] is a general-purpose Finite Element package with a wide range of applications, such as - crashworthiness, metal formation, vibration, and structural analysis. LS-DYNA works on a discretized model made up of nodes and elements. The nodes form the vertices of the solid elements, and the elements represent the material characteristics. LS-DYNA computes the nodal displacements in response to loading forces on the model.

Although LS-DYNA provides a variety of solvers specialized for the various application areas, this paper focuses on the the implicit solver that is used for structural analysis. In the next 3 to 5 years, the node count of a "typical" model will grow 15 fold, requiring two orders of magnitude more processing time than is used today. To put this in perspective, for example, today's typical workload that executes as on overnight job will become a more complex workload that will require more than a month of time for execution on a comparable system. The problem is compounded by the slowing rate of processor clock speed increases in modern systems.

To meet this computational challenge, LS-DYNA will continue to improve upon its use of parallelism on upcoming machines of the future, as these machines will most likely be configured with many multi-core processors, each with many hardware strands of execution per core. Multiple multi-core processors are inter-connected with a high-speed optical network with high bandwidth across the full range of the machine. Application software maps its threads of execution to the resulting vast number of hardware strands.

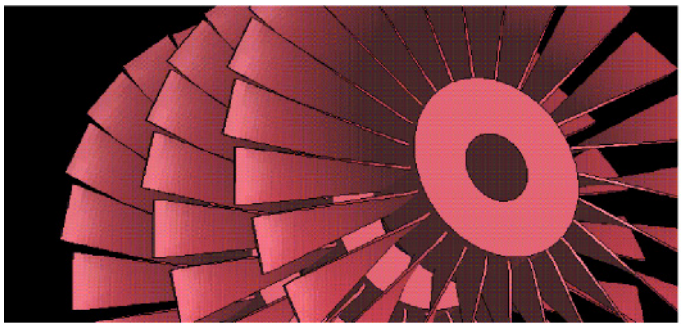

Figure 1. Jet engine impeller with three fans.

To examine how LS-DYNA might run on such machines, we look in detail at a benchmark dataset scaled at an increasing number of nodes and elements to observe LS-DYNA's computational characteristics. We look at a range of benchmarks with an increasing number of fans in a jet-engine impeller, starting with a model of three fans composed of 300,000 nodes and the largest model of ten fans with 1 Million nodes. The benchmark was inspired by a benchmark dataset from Rolls-Royce Group.

The finite-element models of the impeller were modeled using the TrueGrid® package from XYZ Scientific Applications, Inc. As most elements in the model are adjacent to other elements, this results in formation of a stiffness matrix of dimension 3X by 3X, where X is the node count of the model. The models we considered ranged from .9 to 3 Million Degrees of Freedom (DOF).

These model sizes are typical for LS-DYNA runs today. In the coming five years, model sizes will to grow upwards of 50 Million DOF. Using LS-DYNA Version 971 to examine LS-DYNA's performance characteristics, we examine LS-DYNA runs on these smaller models, and project the important factors for models of larger sizes. Arriving at a performance model as a function of available computational threads and size of the finite-element model, we project the performance of LS-DYNA for upcoming multi-core machines.

Analysis Approach

To begin to understand how LS-DYNA might scale on massively parallel machines, we at first need to determine the phases LS-DYNA goes through as it performs its work. This will help in identifying areas to focus towards developing our performance models. One way to do it is to visualize hardware processor metrics as LS-DYNA executes. For example, we can measure the CPI, or Cycles per Instruction, derived from two hardware counters, number of processor cycles, and number of instructions executed, that are sampled for a specified interval. As the CPI varies over the program's duration, due to a number of factors, such as memory stalls, the plot gives a visualization of the "application signature" as it executes.

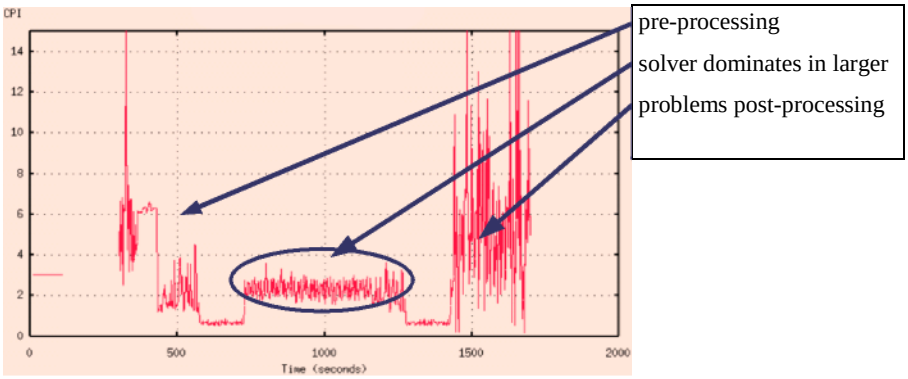

Figure 2. CPI of LS-DYNA executing as one thread.

In Figure 2, we plot CPI time on the x-axis. we can see a preprocessing phase, a "solver" phase, which is circled, and a post-processing phase. In this LS-DYNA run, we have only a few time steps, but in a typical run, there would have many more time steps, hence the solver phase will dominate.

Implicit Solver

The multi-front solver uses the sparsity of the stiffness matrix to its advantage, as the computation time can be substantially reduced by excluding calculation of matrix entries that are zero. The sparse matrix is transformed into a dependency tree [1] of dense sub-matrices, known as supernodes (SN's) [2]. In the diagram, Figure 3, of an elimination tree, the size of a supernode is represented by the area of the triangle. The tree is constructed such that the solution at a given level of the tree depends only upon the solution at the next lower levels of the tree.

The tree is traversed from bottom level to top, performing a Gaussian forward solve on all supernodes at a level of the tree before proceeding up to the next higher level. The tree is traversed again from the bottom to top for the backward solve. These steps are repeated until convergence is achieved. If at any time, the error residual is too large, the elimination tree is rebuilt in a process called factorization. The tree traversal process is performed for each time step in the physical simulation of the model. There is a small number of supernodes at the bottom of the tree, and a large number at the mid-levels of the tree. As the top levels of the tree is approached, there are fewer, but larger supernodes. At the top, there is only one large supernode.

We used the Sun™ Performance Analyzer [5], we identify bottlenecks in performance and scalability in the current version of LS-DYNA. This tool allowed us to generate the profile information for the runs and pinpoint the hot spots of the execution. The limiters of performance and scalability that we identified were the computation in the factorization step and the forward & backward solve steps.

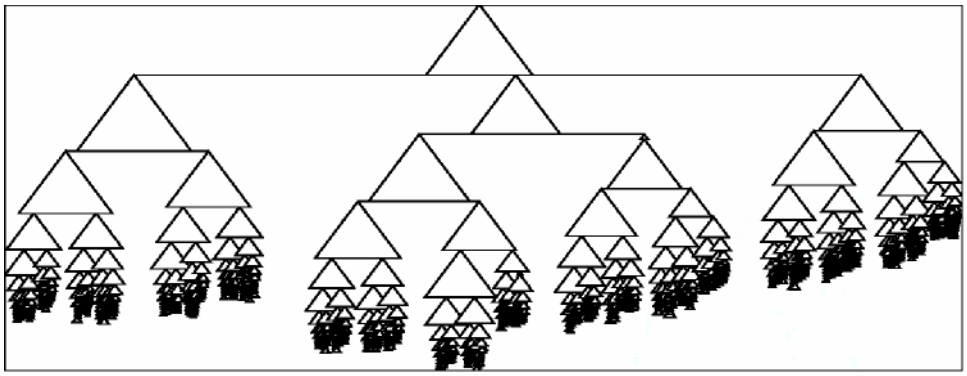

Figure 3. Sample elimination tree, from [1].

We now look at how to traverse the tree for parallel execution. As the processing of the elimination tree proceeds from bottom to top, and a level of the tree is completed before proceeding up. LS-DYNA assigns one compute thread to each supernode. This approach results in good parallel execution for multiple small supernodes in a level. However, if at any level, the number of available threads exceeds the number of supernodes, some threads will remain idle. As a result, unless we exploit parallelism within the larger supernode, the scalability of the code will be suboptimal.

During our investigation, we noticed that it would be desirable to assign multiple threads to the larger supernodes to increase scalability. This would be a good enhancement for inclusion into the DYNA solver that we were looking at. As you'll see later, for our performance projections, we'll assume that multiple threads are assigned to large supernodes to the extent that it is efficient to do so.

Jet Impeller

Let's look at the jet impeller datasets from the viewpoint of analyzing the elimination tree. Figures 4 and 5 show characteristics of the elimination trees as a function of the model size for models up to 3 Million DOF in size. These plots demonstrate that the number of supernodes, and also the amount of memory required to store the supernodes, grows linearly with the model size.

In the 3D plot, Figure 6, the size of the largest supernode is plotted on the z-axis as a function of the tree level, and the DOF, which are plotted on the x and y axes. This plot shows that the size of the largest supernode, which is located at the top of the tree, grows linearly with the model size. Based on the linearity of growth, we can estimate various characteristics of the tree for a given model size. Specifically, we can estimate: (1) the total number of SN's in the tree, (2) the number of levels in the tree, and (3) the minimum and maximum supernode size for each level of the tree. We observe the trends in the characteristics of the tree for our models up to 3 Million DOF, and we project these characteristics to trees reflecting model sizes to 50 Million DOF.

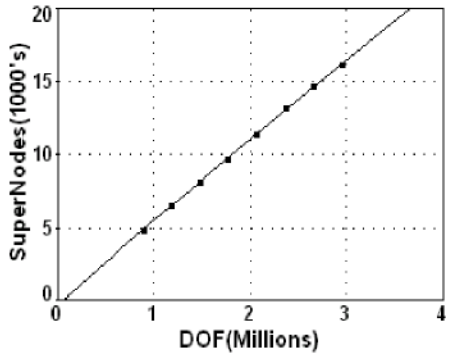

Figure 4. DOF vs. Number of SN's

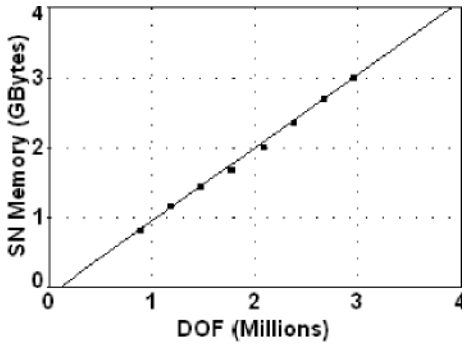

Figure 5. DOF vs. SN memory

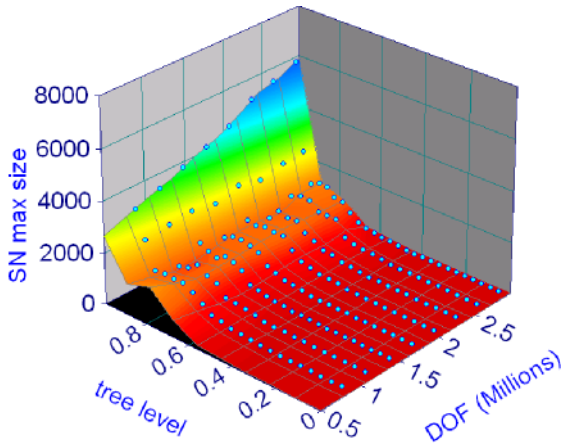

Figure 6. Largest supernode size per level.

Modeling

Now that we have projected the characteristics of the elimination trees for larger model sizes, we simulate the time it would take to traverse these large trees, aggregating the time for performing the Gaussian factorization and solve of the matrix at each supernode.

A computation that involves a large matrix can effectively utilize multiple threads. We have parameterized these rules by measuring the time required to perform factorization and back substitution using the DGETRF and DGETRS subroutines, respectively, from the Sun Performance Library™ [4] on a Sun Fire™ server with 24 dual-core UltraSPARC-III processors. We have measured the DGETRF and DGETRS time as a function of number of threads and matrix size. These routines serve as proxies for the factorization and solve times in LS-DYNA.

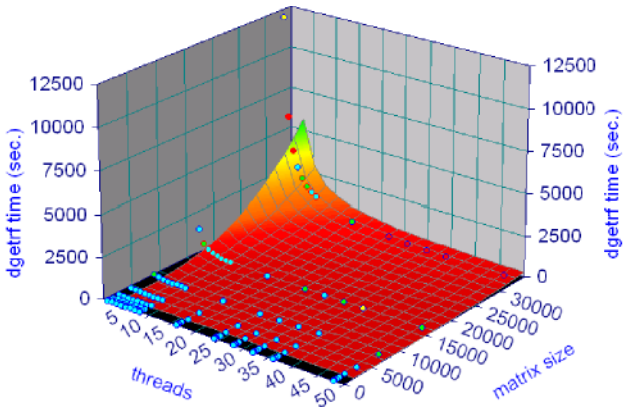

Figure 7. Surface fit on DGETRF

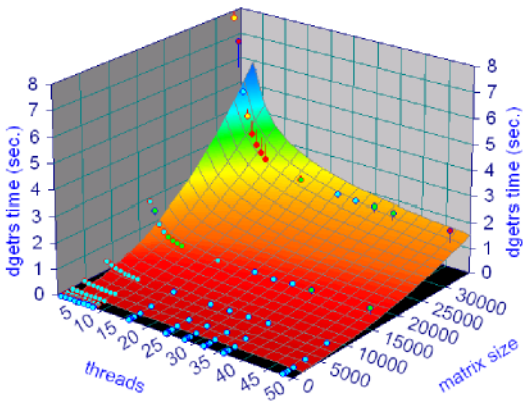

Figure 8. Surface fit on DGETRS

Based on surface fit of these measurements, and assuming scalability as observed, we can predict the optimum number of threads to assign to a matrix of a given size, as well as the time required to process that matrix. We have used this simulator to predict the execution time on a future machine for a large number of threads for models up to 50 million DOF.

In LS-DYNA, the user has the ability to control certain parameters of the solver, this influences the number of invocations of the factorization step during the run, and consequently the number of iterations required to achieve convergence in each time step. For our simulation, we assume a factorization is performed every time step, resulting in a average of 5 iterations of the solve step, for a total of 50 time steps. Additionally, we assume that currently serial computation in the time step loop will be optimized for multiple threads of execution, such as creation of the plot files and calculation of displacements.

Projections

From our simulations of the tree traversal for trees representing models up to 50 Million DOF and at thread counts up to 4096 threads, we develop a surface that models the runtime performance of LS-DYNA. Figure 9 shows the surface fit of LS-DYNA performance for a 50 time step job as function of size of model in DOF and number of available threads. Using this surface, we can project scalability of LS-DYNA at larger thread counts for future machines. In

Figure 10, we plot the speedup of LS-DYNA for hundred-thousands of threads that would be available on future multi-core multi-processor systems.

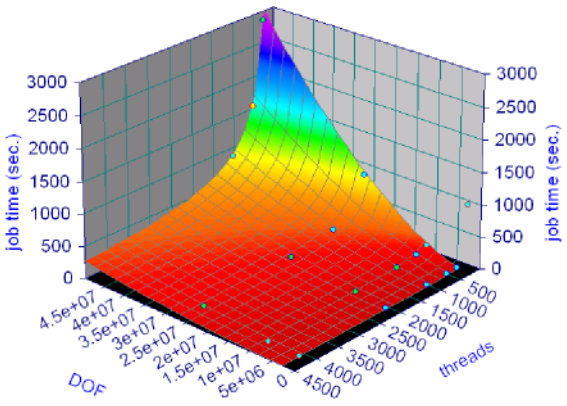

Figure 9. Job times as function of model size expressed in DOF and available threads.

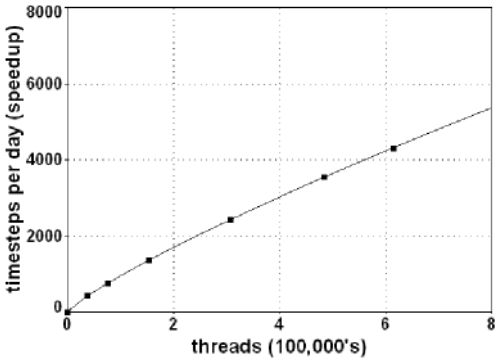

Figure 10. Speedup of time steps per day as function of available threads for the 50 Million DOF model.

Concluding Remarks

We have developed a model to predict performance of LS-DYNA implicit solver on a model of a jet engine impeller. In this paper we demonstrated that the two main computational phases of the implicit solver of LS-DYNA, the factorization and the solver stages, have a very high scalability potential. The concurrency in the computations can be exploited if parallelization is applied at multiple levels: between elements of the elimination tree and within large elements of the tree. Additional exploration is needed to analyze the parallelization potential of the remaining parts of the solver to ensure that the entire program is highly scalable.

Although the current version of LS-DYNA has limited scalability, we expect that the proliferation of massively parallel multi-core multi-processor systems will motivate additional parallelization in the code. As a result, the time it takes to perform a typical simulation will dramatically decrease, which may lead to new usage models for LS-DYNA, for example to interactive simulations.

References

1. Lucas, R.: “Towards a Petascale Multifrontal Method”, Information Sciences Institute, USC, presentation, August, 2005
2. Duff, I. S., “Direct Methods”, Technical Report RAL-TR-1998-054, Rutherford Appleton Laboratory, Oxfordshire
3. Livermore Software Technology Corporation, LS-DYNA Keyword User's Manual
4. Sun Microsystems, Inc., “Sun Performance Library User's Guide”, http://docs.sun.com
5. Sun Microsystems, Inc., “Performance Analyzer”, http://docs.sun.com