A Study of LS-DYNA Implicit Performance in MPP

Dr. C. Cleve Ashcraft, Roger G. Grimes, and Dr. Robert F. Lucas

Livermore Software Technology Corporation,

Livermore, CA, USA

Summary:

LS-DYNA models for Implicit Mechanics are getting larger and more complex. We are continually seeing models where the linear algebra problems in Implicit Mechanics have 10M rows and know of at least one that is nearly 40M rows. We expect users wanting to routinely solve problems with 30M very soon. It is these very large linear algebra problems that distinguish the computer requirements for Implicit Mechanics. This paper will present a study of the performance of the MPP implementation of implicit mechanics in LS-DYNA examining such issues as performance, speed-up, and requirements for computer configuration.

Keywords:

LS-DYNA, Implicit, MPP

1 Introduction

LS-DYNA has implicit technology integrated with its explicit technology. Implicit technology provides the capabilities to perform transient analyses with larger time steps as well as many linear analyses including vibration computations. As problems get larger, implicit distinguishes itself from explicit in that it has very different computer requirements. The differences are really highlighted by the requirement to factor matrices involving the global stiffness matrix. As we move to larger problems we move to larger computers which, nowadays, are MPP computer clusters with multiple cores per node.

This paper present a study of implicit requirements and performance on an MPP compute cluster housed at the Livermore offices of LSTC. We will try to set expectations of performance and give guidelines for computer hardware requirements.

2 The Test Environment

For the test problems we are using the cylinder benchmark problems generated by Atomic Weapons Establishment. The problems are the same physical problem with a grading of mesh so that the same problem is available in meshes with 100K, 1M, 2M, 4M, up to 20M nodes. The physical problem is 6 nested cylinders meshed with solid elements and held together with surface_to_surface contact. There is prescribed motion on the top and a load on the bottom. It has a single elastic material and uses solid element formulation #1.

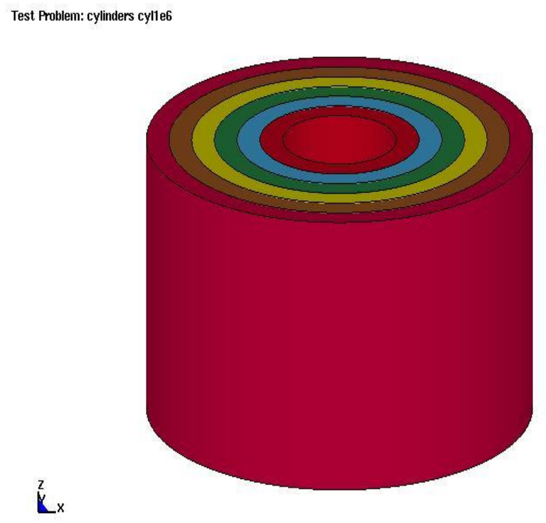

Figure 1: Test Problem

In this paper we use the 2M and 4M node problems which has 0.92M and 1.84M solid elements. These problems had 6M and 12M rows in the global stiffness matrix. All computer runs were made using all 16 nodes of the SGIXE1 compute cluster at LSTC. This cluster has 2 dual quad Harpertown 2.8Ghz compute cores, 16Gbytes of RAM. Runs varied by using 1, 2, 4, or 8 cores per node.

3 MPP Performance

The following table shows the required memory before the linear algebra, the memory available for the linear algebra, factor wall clock time in seconds, and forward/back solve wall clock time in seconds.

No. of cores	Mem. pre LinAlg	Mem. for LinAlg	Factor WCT	Solve WCT
1	55.0M	1445M	17,038	107
2	27.0M	773M	10,275	152
4	13.5M	386M	5,529	215
8	6.8M	193M	3,388	406

Table 2: MPP Performance

One can see that the memory requirements before the start of the linear algebra scales linearly as the number of cores increase. As a function of fracturing the 16Gbytes memory per node across the multiple cores the memory per core for the linear algebra decreases as well. The Factor time speeds up in an acceptable manner allowing for the intra-node conflicts caused by sharing nodal resources across the multiple cores. The solve slow down is due to I/O contention caused by multiple cores utilitizing the single I/O device.

For those that like effective computational rates we achieve 3.1 Gflops per core for the 1 core per node run, an aggregate 50 Gflops. This is for the dominating numerical factorization phase. For 8 cores per node the per core flop rate reduces to 2.1 Gflops while the aggregate soars to 267 Gflops.

4 Serial Memory Bottleneck

We were able to run the 4M node problem using 2 cores per node. The factor time using 2 cores per node was 29,896 seconds. We could have made a 1 core per node run but did not due to limited compute time availabilty. One of the „features“ of the implicit implementation in LS-DYNA is a serial memory bottleneck during the Symbolic Factorization Phase. This is due to the requirement for computing the sparsity perservation ordering on the compressed graph for the entire model.

For solid element problems this serial memory bottleneck is about 90*N where N is the global number of nodes in the model. For the 2M node problem this is 180M which fits in the available memory. But when we go to 4M node problem the requirement is 360M. This requirement is just a bit more than what is available for 4 cores. So we were restricted to 2 cores at the time of the writing of this paper.

We are currently working on reducing this number from 90*N to 60*N. We hope that by the time of the 7th European LS-DYNA Conference we will be able to show the results on 4 cores.

A long term scalable solution requires us to develop an approach for either using the model decomposition as a basis for the ordering or using a distributed memory ordering algorithm. We are also exploring the use of Hybrid paralellism, that is using MPP across the nodes and SMP on each node. While hybrid parallelism does not change the need for a scalable solution it postpones the requirement by not fracturing the memory across cores on each node.

5 Computer Resources

5.1 Memory Requirements

As described in the previous section, memory on a node for the MPP cluster will be a limiting factor. The amount of memory per node should be (440/P + 75)*N where P is the expected total number of cores and N is the global number of nodes in the model. The 440 comes from a simple linear fit to the second column of Table 1.

Remember that the memory on a node not only has to hold the LS-DYNA work array whose length is given by the memory=xxxxM on the command line but also dynamic memory for MPP core utilities including the matrix assembly package and the LS-DYNA executable. The authors always use the following memory specifications on our 16 Gbyte per node cluster

Cores	Memory
1	1500M
2	800M
4	400M
8	200M

The authors also recommend not using the memory2= option on the command line for implicit. The resulting memory imbalance among the processes will negatively affect the overall performance.

5.2 I/O Requirements

To keep memory to an acceptable level the factorization of the global stiffness matrix must be stored on disk. For the 2M node cylinder test problem the global size of the factorization is 222 Gbytes. For the 4M nodes problem this grows to 524 Gbytes. Of course, this is spread out on the disk system local to the nodes. So each compute node requires about 14 Gbytes per node for the 2M problem and 33 Gbytes for the 4M problem to hold just the factorization. Additional scratch I/O requirements during the factorization may be 50% larger. Roughly the requirement of disk space per core is 200*N/P bytes. Remember to multiply this number by the number of cores per node to get the nodal I/O disk size requirement.

At LSTC we have 128 Gbyte disks on each node. This allows us to easily hold the scratch files for the 4M node problem. But we could not hold the 10M node problem from the benchmark suite using our current 16 node configuration.

One should also remember that the multiple cores on a node will contend with the single I/O resource so fast disks are also required. While not covered in this paper, we expect that this I/O contention will have a more negative effect on eigenvalue computations. For those problems fewer cores might lead to better overall performance.

Our experience to-date indicate that local disks on each node give better performance for the scratch files for the numerical factorization.

6 What Happens for Shell Problems

Shell problems have 6 degrees-of-freedom at each node instead of 3 for solid element problems so the resulting linear algebra problem has twice as many rows. However, the nodel connectivity for shell problems is less. On average a node connects to 26 other nodes in solid element problems but only 8 others in shell problems. This reduction in connectivity drops the serial memory bottleneck and the size of the factorization for a comparable number of rows. The effect of twice as many rows is about equally countered by the decrease is problem connectivity. It is reasonable to use the same formulas based on the number of nodes and the number of processes to project the requirements for shell element problems.

It is hope that by the conference we will have results from the gravity loading phase of a car model with around 2M nodes to substaniate this claim.