# Structured ALE Solver with Large Models

Hao Chen
*Ansys Livermore*

## Abstract

*In 2015, LS-DYNA® introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions.*

*In this paper we will introduce the recent enhancements to facilitate running large models using S-ALE solver.*

## S-ALE Solver

In the past two decades, we have observed quite some changes in the simulations our ALE solver users. First, ALE model size has grown rapidly. Sixteen years ago, when the author started his career as ALE developer, a typical model was of several hundred thousand elements and ran on 8 or 16 cores. Today, 10 million element models is considered normal if not small. And running with thirty or forty-some cores is definitely a start point. Secondly, the type of applications changed. Twenty years ago, people used ALE ELFORM=5 to study severely deformed Lagrange solid parts. Nowadays, most, if not all ALE users, running their models with ALE multi-material formulation (ELFORM=11), to simulate multiple fluids flowing in an ALE mesh. Finally, mesh also changed. While in the past, in quite some models, meshes were constructed to conform the material interfaces; now more than 90% of models are using rectilinear, structured ALE meshes.

This motivated LS-DYNA to develop a separate solver dedicated to solve ALE problems using structured mesh with multi-material formulations. The idea was, the logical regularity of the mesh could lead to algorithmic simplifications, memory reductions, and performance enhancements, which are impossible in unstructured mesh geometries. In 2015, LS-DYNA introduced a new structured ALE (S-ALE) solver option dedicated to solve the subset of ALE problems where a structured mesh is appropriate. As expected, recognizing the logical regularity of the mesh brought a reduced simulation time for the case of identical structured and unstructured mesh definitions.

## To handle large models

S-ALE solver could handle large models much better. First, the mesh is generated in the solver, not by reading in each element and node. This reduced the geometry and connectivity information from gigabytes of data to almost nothing. Before, the process was to a). generate the mesh using pro-processor; b). save the mesh in a keyword input file; c). LS-DYNA reads in the keyword input file and generates its meshing database. Now, the S-ALE solver simply reads in the user-provided mesh spacing info along 3 directions, the origin and the local coordinate system. Then it generates the mesh and store it in the database. The reduction in time and storage made handling large models much easier.

Secondly, the regularity of the mesh could bring huge reductions on memory usage too. In the solver, there are quite some memory used to store the geometry and connectivity data, also bookkeeping arrays. The author put in a lot of effort to minimize the memory usage, and in the meantime, applying minimum changes to the overall LS-DYNA structure. The memory reduction project, is still an ongoing process and not yet completely finished.

Finally, efforts are being made for a better post-processing for S-ALE models. It is a joint project by the LS-PrePost® team and the S-ALE developer. The idea is to one the solver side, come up with a much more compact post-processing database; and on the LS-PrePost side, optimize the algorithm and reduce the memory usage by taking advantages of regularity of the mesh.

## Memory is the bottle neck

The major obstacle to run a huge model is the memory limitation. Please note here as MPP has to be used to run large models we limit the discussion in this paper to MPP simulations. In a LS-DYNA MPP run, the maximum memory requirement mostly likely happens in the PHASE 1 of the initialization process. In this phase, the head node (processor 0) reads in the model and constructs the database to store it. Later, the whole model is decomposed into small, separate models to be run on each individual processors. And by that time, the memory burden is much more alleviated. So, one way to overcome the memory limitation is to build our computing system so that the head node is equipped with a huge memory. This way, in PHASE 1, the head node has enough memory to hold the whole model and later all nodes only use much less memory to handle their pieces of the model.
However, it is not practical and also quite wasteful. Another way used was to find a standalone machine with a huge memory. Do decomposition there and store the decomposition file. Later at the cluster machine, jump the decomposition phases and instruct each processor to read the decomposition file and start from the small, decomposed model directly. It is quite a cuber-some process so that even the most experienced user needs to go through a few iterations before a final success.

## Memory reduction in S-ALE

S-ALE handled today's ALE models with no problem. Typically, the number of elements goes from several million to 10, 20 million. But realizing the rapid growth in model size which is supported by growing computing power, the author decided to overhaul the initialization process for S-ALE solver to better prepare it for the future.

The idea is quite simple. The maximum memory usage is at PHASE 1, while the head node needs to store the whole model. So, the only way to reduce the memory usage is to reduce the model size in this phase. While it is quite difficult, if not impossible for unstructured meshes, it is relatively simple and definitely doable for S-ALE meshes. As the mesh is constructed by the solver, at which phase we construct it is at our discretion. So why do not we construct the S-ALE mesh at each core instead of construct the whole mesh at core 0? This way, the huge memory needed to process the whole S-ALE mesh on the head node is not needed any more. And the memory requirement on the head node is reduced to like a hundredth of the original one.

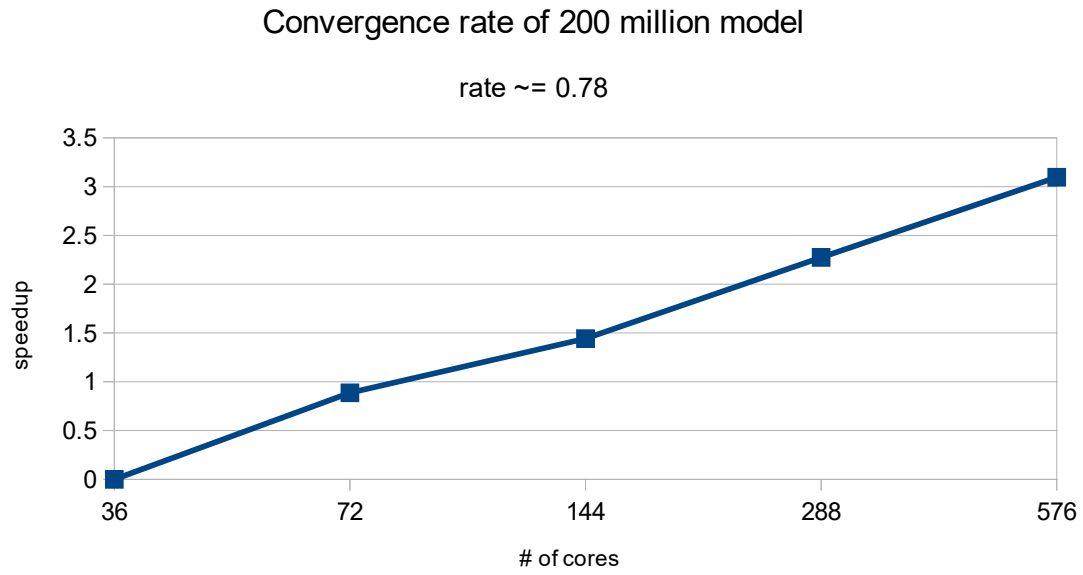
At the beginning of this year, the author finally found some time to put these thoughts into action. By overhauling the whole initialization process, S-ALE mesh only comes to live at the very end of initialization process. At that time, each processor has its own sub-model and only needs much less memory to handle that sub-model.

## A 200 million test model

For illustration purpose, the author picked a small projectile penetration model which is composed of 387,000 elements and refined it 8 times in each direction to get a model containing 198,144,000 elements. The input deck and problem description could be downloaded from http://ftp.lstc.com/anonymous/outgoing/hao/sale/models/cthrod/. The executable used is mppdyna dev version 134214 single precision. It was run on a cluster contains 768 cores and each 12 cores resides at one of 72 nodes. It was built using Xeon(R) CPU E5620 and each node has ~ 100GB memory.

Five runs were made, using 36, 72, 144, 288, 576 cores, respectively. Below are the running times and MPP convergence plot.

| # of cores | 36 | 72 | 144 | 288 | 576 |
|---|---|---|---|---|---|
| Total time(s) | 84473 | 45711 | 32113 | 17738 | 9738 |
| Time per cycle (s) | 10.056 | 5.4405 | 3.7014 | 2.0785 | 1.1761 |
| Speedup | 1.0 | 1.8484 | 2.7168 | 4.8381 | 8.5503 |




## Conclusions

We introduced the recent effort in reduce the memory usage in LS-DYNA Structured ALE solver in this paper. This memory reduction enabled S-ALE solver to handle large models. The S-ALE developer at Ansys Livermore is committed to continually work with our users to improve.