Dynamic Load Balancing

Brian Wainscott

LSTC

1 Introduction

MPPDYNA begins each simulation by splitting the model into multiple pieces (domains), and assigning each domain to a CPU core. This is referred to as "domain decomposition." Efficient MPP processing requires that, as much as possible, every CPU core is kept busy doing useful work. That is to say, each domain should represent the same amount of work to be done. If one core is assigned a domain that is too large, then at certain points in each timestep cycle the other cores will be idle, waiting for this core to finish its calculations.

The initial decomposition is based primarily on measured execution times for the different types of elements and the different material models. And it has been the case that once the domains are determined, they persist through the duration of the calculation. This approach has two significant problems. First, the element costs used during decomposition are not perfectly accurate. As material types are added, routines are modified, compiler options changed, and new CPUs are available, keeping this decomposition timing information up to date is simply infeasible. But even if that could be done, the second issue is that for most materials the computational cost of the material changes during the calculation. As elements distort, or exceed their elastic limit and begin to experience plastic deformation, the element evaluations can become more time consuming. This leads to the inevitable conclusion that any static decomposition will result in at least some computational imbalance. As core counts increase, the need for dynamically adjusting the decomposition will also increase.

2 Background

One significant obstacle to dynamic rebalancing has been the use of static arrays for storing all the simulation data. During the time LS-DYNA was initially developed, modern dynamic memory allocation was simply not available. Over the past few years, new dynamic memory capabilities have been added to LS-DYNA based on modern FORTRAN standards. The amount of work required to convert all data storage to this new scheme is substantial. But without this conversion, true rebalancing is not possible. To move nodes and elements between cores, we must be able to resize all the related arrays.

3 Approach

The current approach is to do the job in pieces, one capability at a time. Arrays can be converted to dynamic memory without changing the algorithms, and without affecting results. Once all the necessary arrays for some feature have been converted, it then becomes possible to consider what needs to be done to be able to migrate that feature between processors.

Nodes are a natural place to start. As of this writing, approximately 50 node related arrays have been converted to dynamic memory. To move a node from one core to another, data must be extracted from each of these arrays and communicated to the new core, where it must be inserted into the arrays on that core. All features that reference nodes must be renumbered on each core to account for the new node distribution, and a number of node related data structures must be rebuilt. A similar approach has been taken for elements, contact nodes and segments, discrete elements, joints, etc. With over 30 years of LS-DYNA development to catch up on, complete support for all features is not expected any time soon. But some kinds of problems with limited feature requirements should be supported in the near future.

4 Examples

4.1 Metalforming

The initial work on dynamic load balancing was performed in conjunction with changes to metalforming adaptivity [1]. The mechanism of moving nodes and elements between processors shares many features with that of creating nodes and elements when adapting. But simply adding nodes and

elements on each processor as required for adaptivity can lead to load imbalances due to localized element creation. So rebalancing the result is desirable.
This is a small stamping problem with only about 42K elements in its final configuration, and runs on a workstation in just a few minutes with in-memory adaptivity enabled. By default, this includes regular load rebalancing, and results in a final decomposition as shown in figure 1.

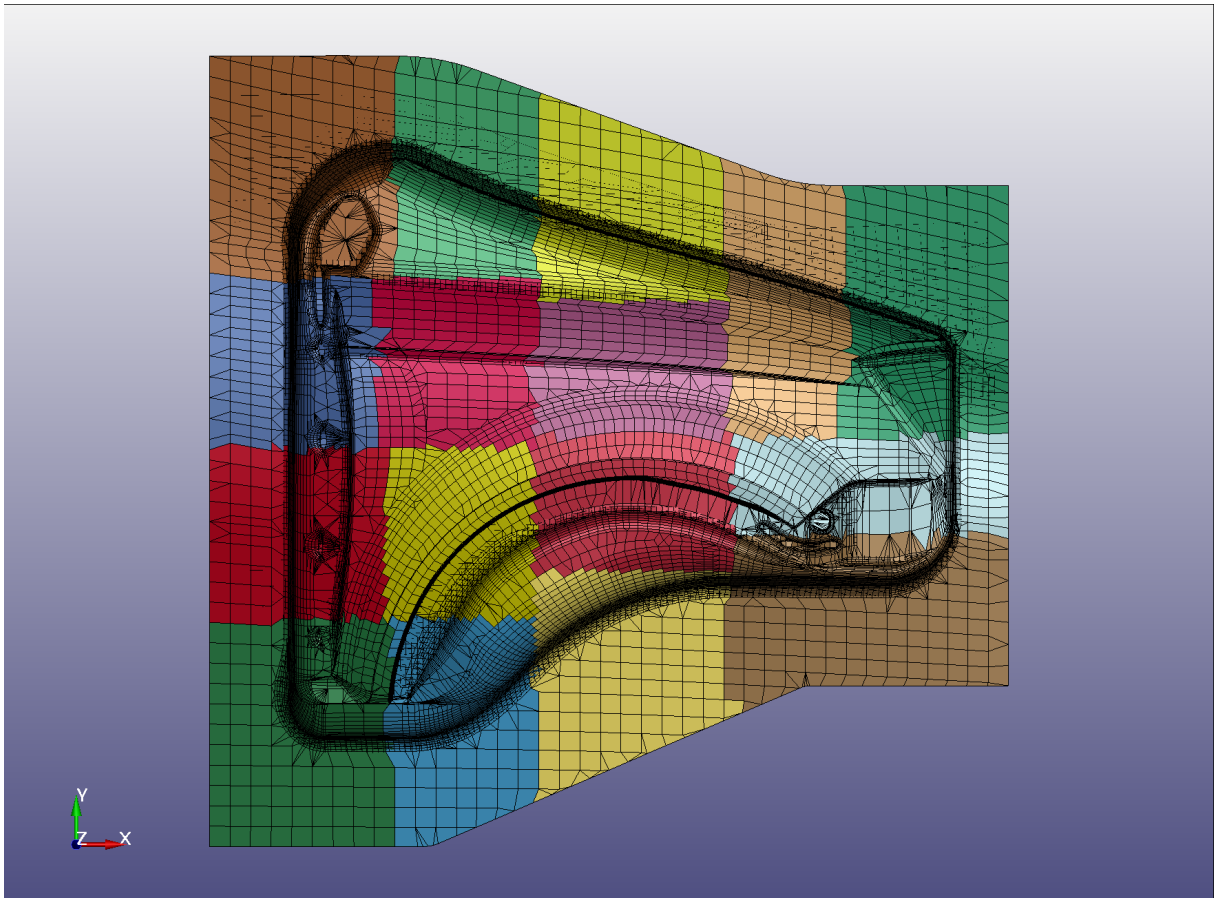


Fig.1: Final decomposition with regular rebalancing

As a test, MPPDYNA was modified to do the adaptivity without any extra rebalancing. This results in a final decomposition (matching the initial decomposition) as shown in figure 2.

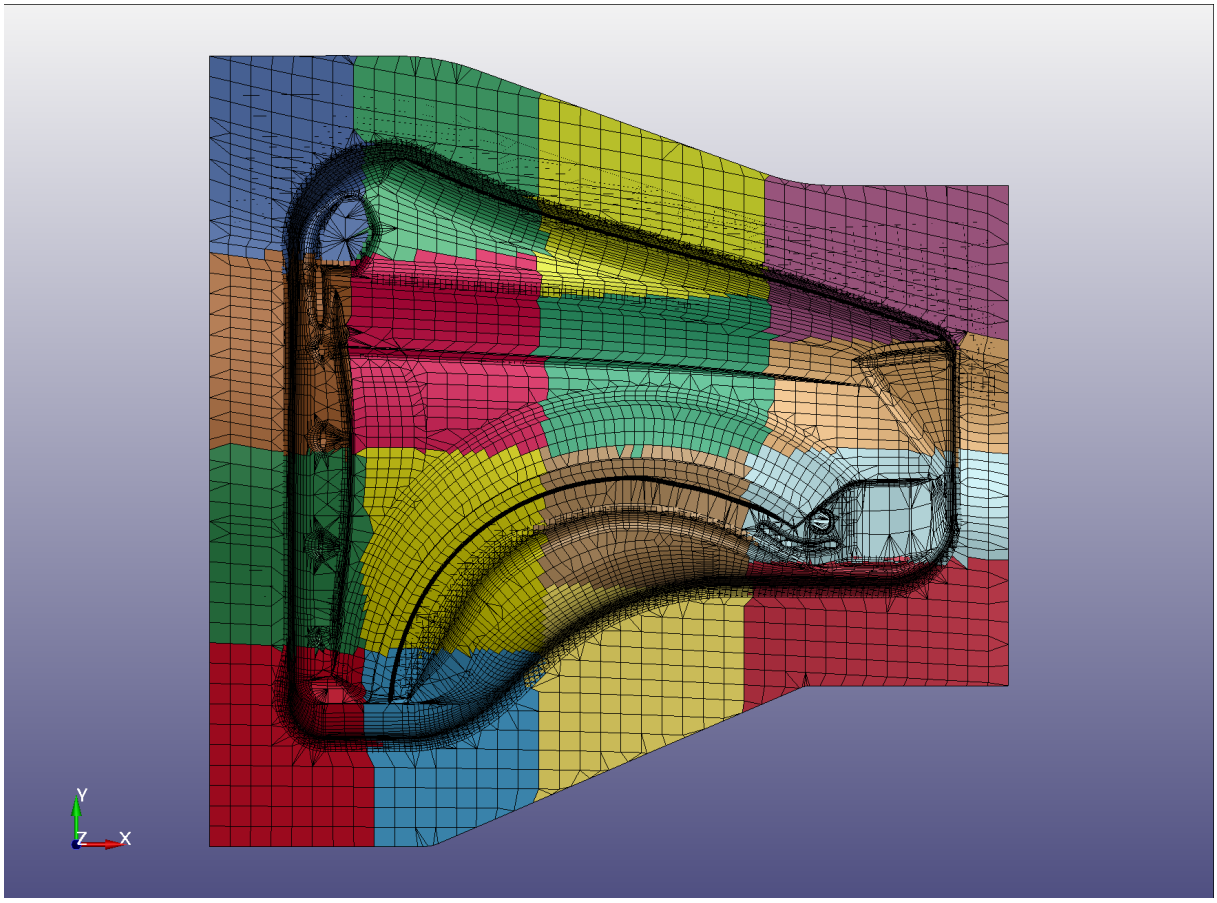


Fig.2: Final decomposition using in memory adaptivity with no rebalancing

The differences may not appear significant, but the non-balanced run took 487 seconds to run, while the rebalanced version ran in 348 seconds. This is a 29% reduction in overall runtime.

4.2 Bending

One of the major motivations for dynamic load balancing is the changing computational cost of materials as they undergo deformation. This of course depends on the material model and element type, but the issue is highlighted by this simple bending example. The material used here is *MAT_SAMP-1, which uses a table lookup for the stress/strain evaluation and so is much more expensive when undergoing deformation. A solid square cross section beam is constrained in one dimenstion at both ends and displaced in the center as shown in figure 3.

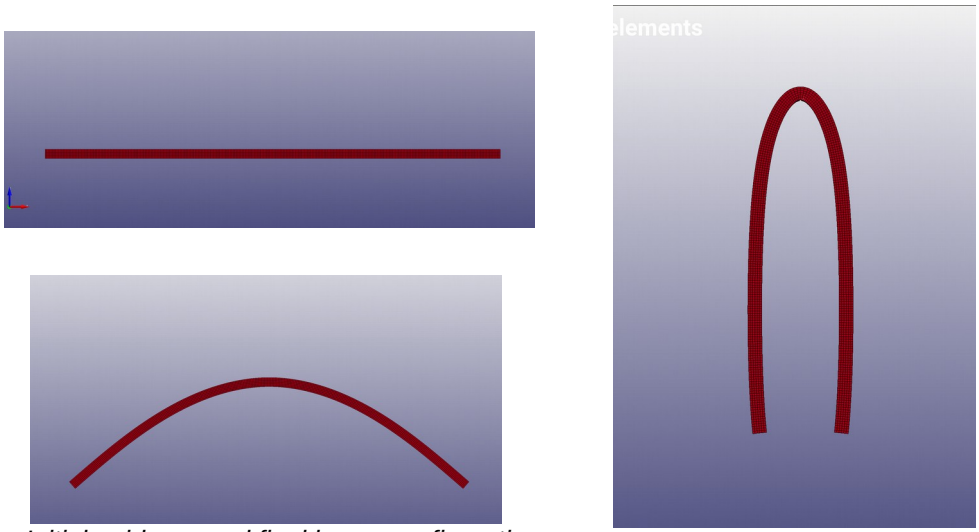


Fig.3: Initial, mid run, and final beam configurations

Without rebalancing, the beam will be divided along its length with each core receiving an equal portion of the beam. This quickly results in computational imbalance as the processors near the middle have much more work to do. But with rebalancing enabled, the elements are redistributed resulting in more balanced computational load. On 15 processors, with rebalancing, the final decomposition is shown in figure 4.

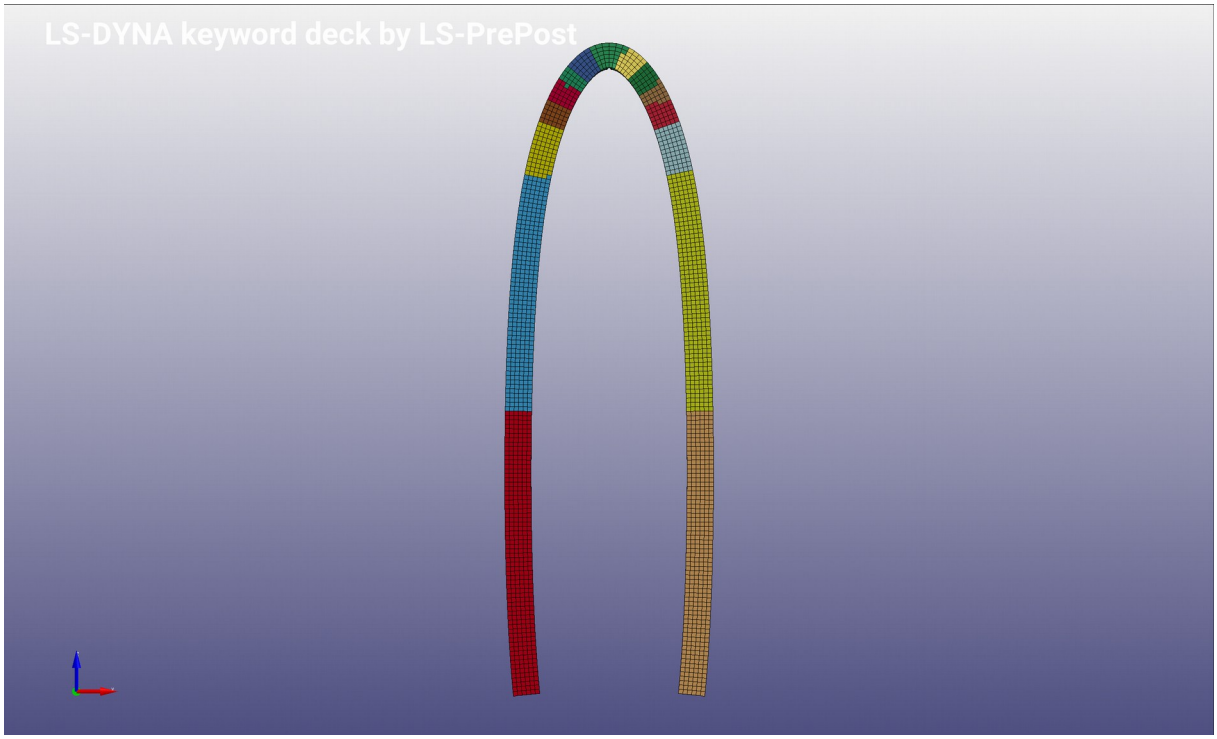
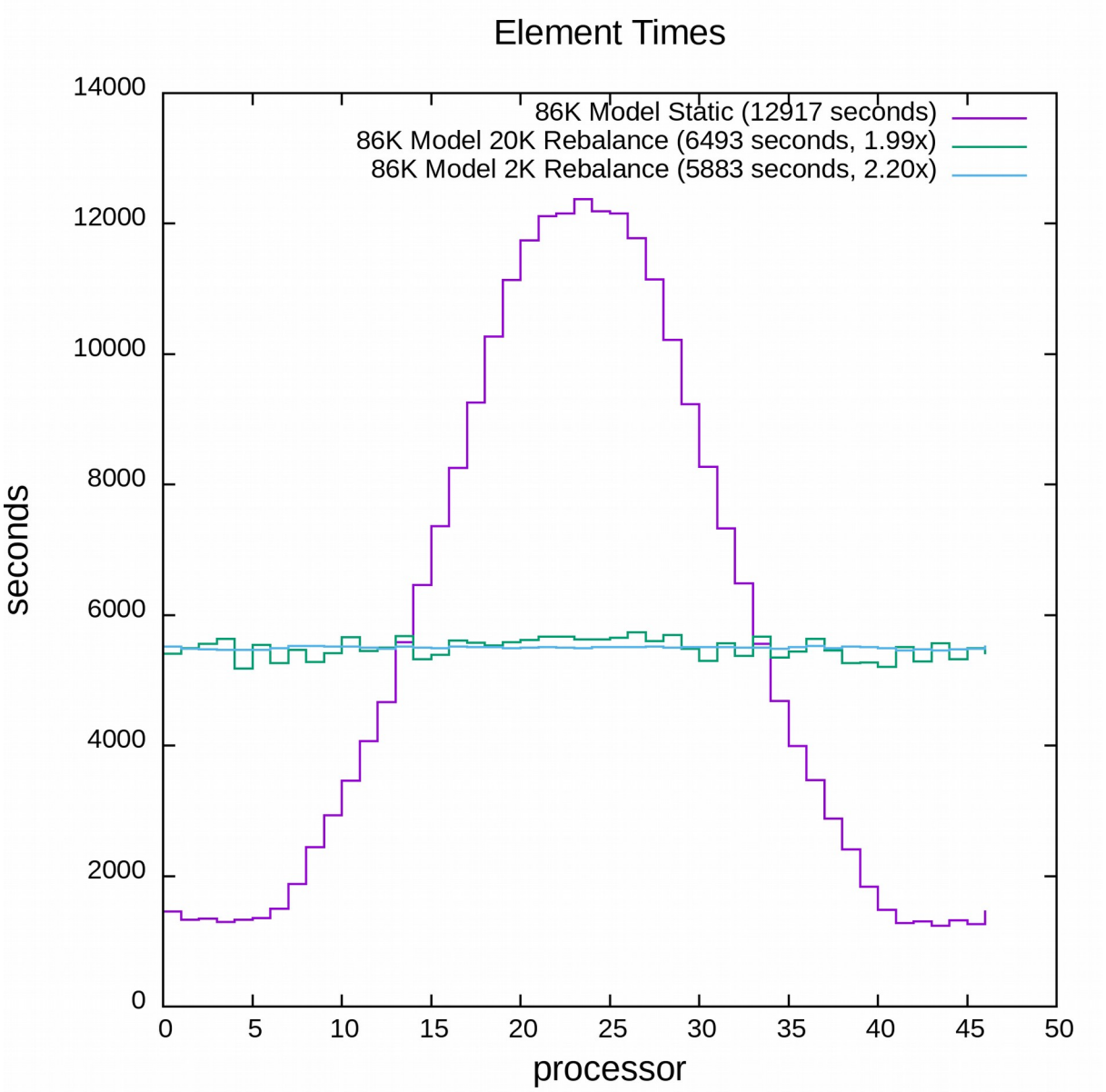


Fig.4: Final decomposition

In this example, nearly 100% of the runtime is due to the elements – there isn't a lot else going on. If we plot the CPU time spent in the elements against the core number, without rebalancing, it is obvious that the cores near the center of the beam are kept busy while the cores near the ends spend most of their time doing nothing (purple line in figure 5). Rebalancing every 20K cycles improves this situation substantially, and rebalancing as often as every 2K cycles is even better. This plot of element computation times shows that, for a beam with 86K elements when run on 47 processors, the simulation is more than 2 times faster with rebalancing. The times given in the figure are total run times.



5 Current Status

Work so far has be focused on fundamental capabilities. Keywords in LS-DYNA typically have a large number of options, and most of the keywords listed here have some options that are not currently supported with dynamic rebalancing. But for simple problems, the following keywords can be used:

*AIRBAG (ALE, INTERACTION, PARTICLE, and REFERENCE_GEOMETRY not supported)
*BOUNDARY_PRESCRIBED_MOTION
*BOUNDARY_SPC
*CONSTRAINED_ADAPTIVITY
*CONSTRAINED_JOINT
*CONSTRAINED_NODAL_RIGID_BODY
*CONTACT (most common types except soft=2)
*ELEMENT_BEAM
*ELEMENT_DISCRETE
*ELEMENT_SHELL
*ELEMENT_SOLID
*LOAD_NODE
*NODE

*RIGIDWALL

Other developers at LSTC are using the routines and techniques developed here and are working on the redistribution of *ELEMENT_DISCRETE_SPHERE, *ELEMENT_SPH, and *ALE capabilities.
As the capabilities expand to the point of being widely useful, the issue of repeatability will need to be addressed. Currently, when running a simulation in MPPDYNA, it is expected that the results will change if the number of processors or the decomposition is changed. But if the same model is run multiple times on the same number of processors with the same decomposition, the result will be unchanged. Dynamic load balancing redistributes the elements based on the measured execution time of each block of elements. This will almost certinaly lead to different redistributions of the elements every time a problem is run, and thus different results. For many cases this may be acceptable, but this lack of repeatability will always be an issue for some users. Completely consistent results, independent of the processor count or decomposition, are currently possible for some problems [2]. The techniques for this consistency will need to be applied to each feature as load balancing is enabled throughout the code. This will result in not only the ability to do dynamic load balancing, but to have processor and decomposition independent results as well.

6 Summary

Computational load imbalance is a significant cause of poor parallel performance. As the number of cores used contiunues to increase, it will be more and more of a problem. The performance gains that can be made by redistributing the nodes and elements during the calculation are substantial, and worth the time and effort required to implement the necessary code modifications. The required framework has been developed, and several fundamental capabilities have been implemented. There is still much work to be done to support all MPPDYNA features but, as the supported feature list continues to grow, customers should begin to be able to benefit from the increased performance possible. MPP consistency will continue to be developed in step with dynamic rebalancing, to allow for repeatable results even when rebalancing is used.

7 Literature

[1] Wainscott, B and Fan, H: “In Core Adaptivity”,Proceedings of the 15th International LS-DYNA Conference

[2] Wainscott, B and Han, Z: “Processor Count Independent Results: Challenges and Progress”, Proceedings of the 11th European LS-DYNA Conference