Load Balancing Update

Brian Wainscott
LSTC

Abstract

One of the keys to efficient parallel processing is having an evenly distributed workload so that every processor has the same amount of work to do. Because there are unavoidable synchronization points in the simulation process, when one processor has more work then the other processors will have to wait, which wastes CPU time. LS-DYNA® has many options available for controlling the initial distribution of elements to processors to help achieve a good load balance. Unfortunately, for many simulations the distribution of work changes during the run. This can make it nearly impossible to have a good load balance over the whole simulation with a static decomposition. The capability to move nodes and elements between processors during the simulation to maintain good load balance has been under development for some time. The current state of this ongoing work is presented.

Introduction / Background

Work toward dynamic load balancing began several years ago with the introduction of Fortran90 based dynamic memory support in LS-DYNA. Much has been done, and work is still ongoing, to reorganize the fundamental data structures and storage mechanisms in LS-DYNA. This reorganization makes it possible to efficiently change the number of nodes or elements on a processor. The first application of this work was to implement adaptivity in metal stamping simulations without the need to dump all memory to disk and reload the model [1]. New nodes and elements are simply created as needed and inserted into the proper data structures. This can result in dramatically decreased run times for these simulations. The ability to migrate some features between processors was presented last year at the 12th European Conference [2], including nodes, elements, and a limited set of other features. The ultimate goal is that all features would be supported so that all models can benefit from the improved efficiency possible due to dynamic load balancing.

New Features

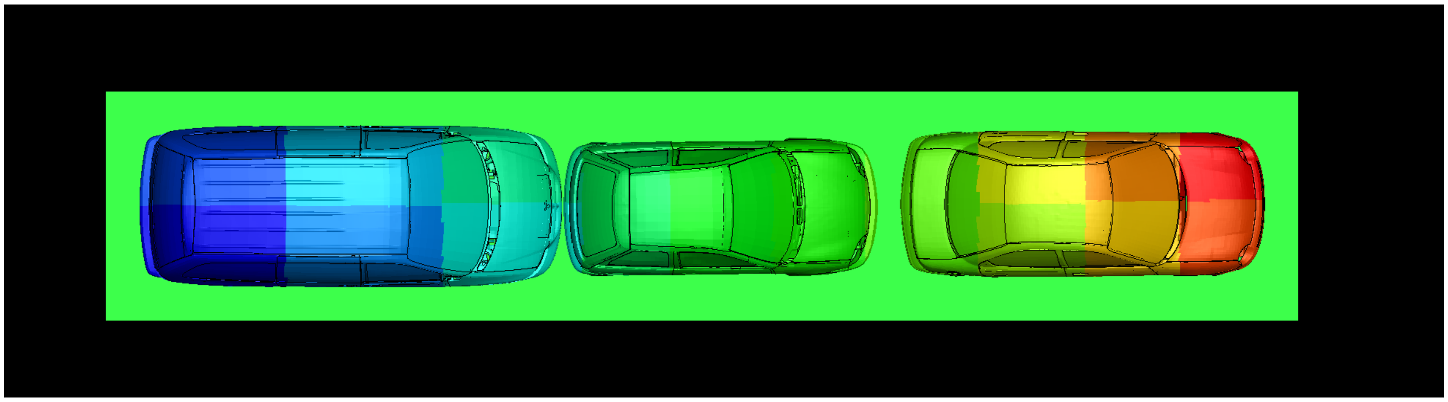
Since the last conference, support has been added for rebalancing these keywords:

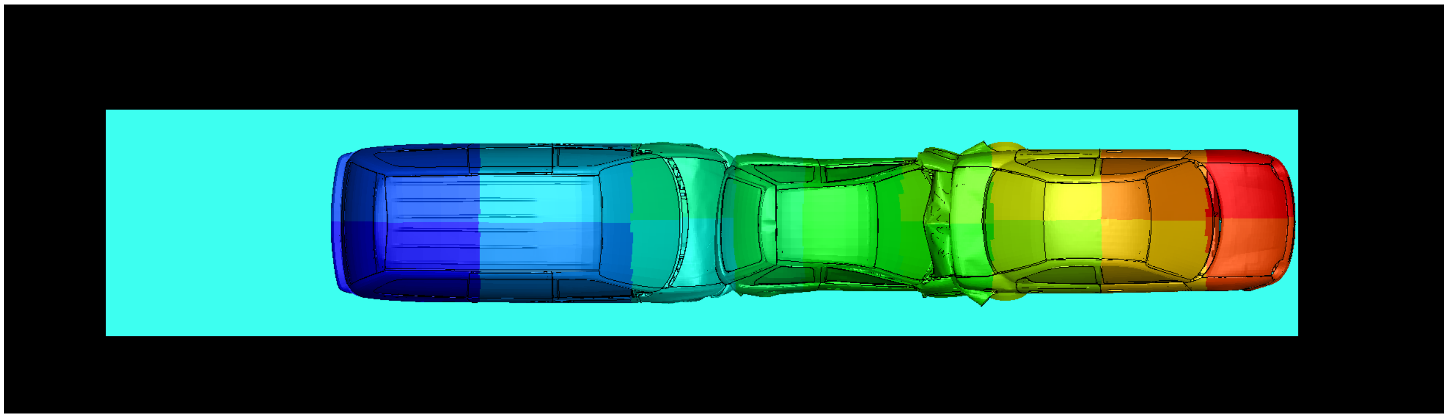
```
*AIRBAG_WANG_NEFSKE
*BOUNDARY_SPC
*CONSTRAINED_*
*DATABASE_CROSS_SECTION
*ELEMENT_DISCRETE
*ELEMENT_MASS
*ELEMENT_SEATBELT_*
*ELEMENT_THICKSHELL
*JOINT_STIFFNESS
*LOAD_BODY
```

Additional work has been done to make sure that as features are migrated between processors the following output data are handled properly: bndout, deforc, elout, jntforc, matsum, rwforc, sbtout. These are in addition to the ones that were previously supported, which include abstat, glstat, nodout, rcforc, and sleout.
Contact support previously required that all contacts have the "groupable" option set. While this is still recommended (and is automatically enabled on contacts that require it when rebalancing is activated), *CONTACT_NODES_TO_SURFACE, *CONTACT_ONE_WAY_SURFACE_TO_SURFACE and *CONTACT_SURFACE_TO_SURFACE are now supported without the groupable option.

Example problems

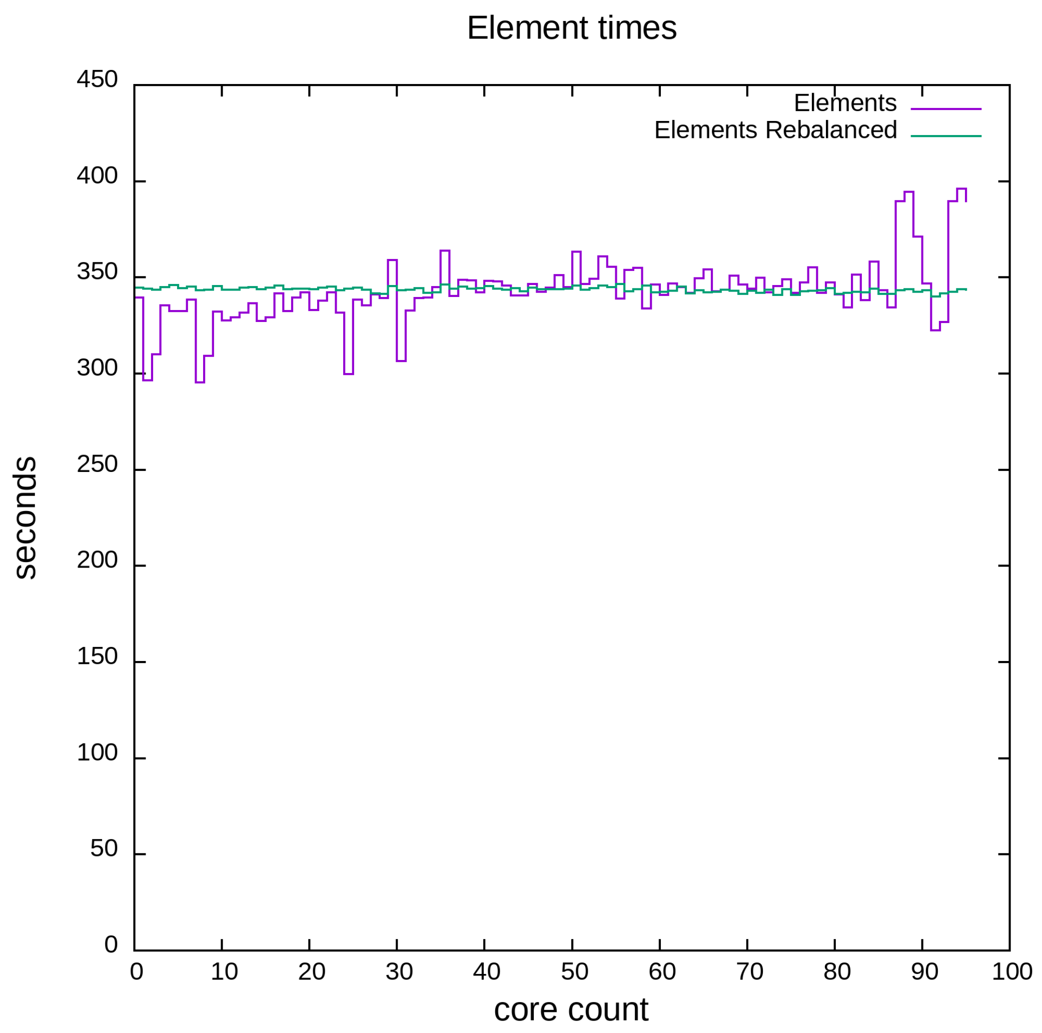
With all of the newly implemented features, it is possible to run some full crash models. For example, this model of 3 cars in a rear end collision can now be run with dynamic rebalancing. This image shows the initial decomposition on 16 processors. A close comparison with the final configuration illustrates the change in decomposition boundaries that took place during the simulation (e.g. the yellow portion of the lead vehicle).


Initial decomposition

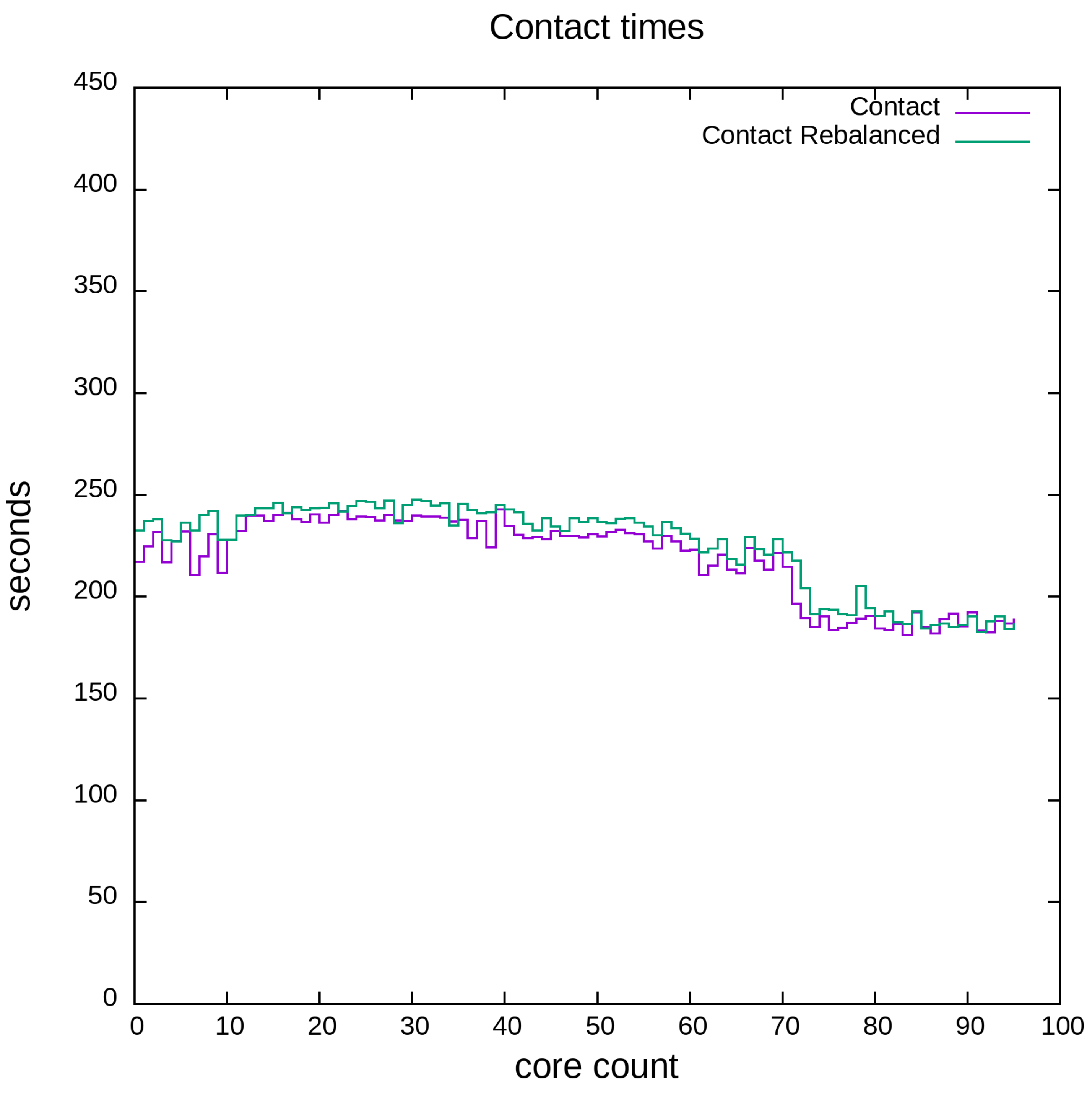

Final Decomposition

Challenges

The primary emphasis of this work has been to robustly repartition the simulation while it is running. That is, addressing the issue of how to move nodes and elements between processors. A related but open issue is determining the best repartition scheme: where should the nodes and elements go? Currently, this is primarily determined by the actual time spent computing each element. The elements are assigned a computational weight based on the measured execution time, and the problem is decomposed based on those weights. There are several issues to address here. Element processing does not account for 100% of the runtime, and balancing the element times can have unintended side effects. For example, when the 3 car model mentioned above was run on 96 cores, the run times with and without rebalancing were the same 829 seconds. The rebalancing itself (every 5000 cycles in this case, resulting in 29 rebalance steps) took 9 seconds of that time, but ironically this was exactly the number of seconds of reduction in the actual calculation. This graph shows, by processor, the total element execution times for the normal and rebalanced runs:



As can be clearly seen, the element time based rebalancing had its intended effect. The element execution time on each processor is nearly identical with rebalancing. On the other hand, the time each processor spent in contact is not well balanced, and in fact went up on average, as seen here:



In addition, there are numerous other features that will be redistributed as nodes and elements are relocated, such as rigid bodies, spotwelds, and other constraints. Moving these will also impact execution times and wait times at synchronization points. There is clearly work to be done both in the area of finding an optimal distribution of the elements across processors, and in minimizing the number of synchronizations and their impact.

Another area of concern is repeatability. In a non-rebalanced execution, the behavior of the code is completely deterministic. Running the same problem on the same number of processors gives the same result. However, with rebalancing as currently implemented, the distribution of elements will change from run to run based on the measured element execution times. The initial element distribution will of course be the same, but there is effectively no chance that the timing routines will return exactly the same measured execution time if the job is run more than once. This will lead to different rebalancing results, and hence simulation results that are not bitwise identical. For stable models, the results should be similar – along the lines of changing the decomposition or the number of processors. For some work this is acceptable and perhaps even desirable, but for others it is not. There are ways to guarantee the same results with any decomposition and any number of processors, and work has been done in LS-DYNA toward achieving this. But the resulting performance is not yet as good as hoped.

Future Direction

Many crash customers use the soft=2 contact variant, which does not currently work with rebalancing. Work is underway to implement this feature. Other crash related features will continue to be added, so that soon it should be possible for some customers to begin using load balancing during routine crash simulations. But there are many other capabilities to be added and much work to be done. As usual, the decision about where to expend development effort will largely be driven by customer demand. If you use a feature that you'd like to see supported, please contact LSTC.

References

[1] Wainscott, B and Fan, H: "In Core Adaptivity",Proceedings of the 15th International LS-DYNA Conference
[2] Wainscott, B: "Dynamic Load Balancing", Proceedings of the 12th European LS-DYNA Conference