Modular Contact in LS-DYNA: Progress and New Capabilities

Matthias Birner[1], Brian Wainscott[1], Albert Ziegenhagel[1]

[1]Ansys, part of Synopsys

1 Introduction

Recently, the "modular contact" has been introduced to LS-DYNA to form a *modular*, *generic* and *universal* framework for contact algorithms [1]. The implementations of node-to-segment and edge-to-edge algorithms have been proven to give improved performance compared to established implementations in LS-DYNA, especially in highly parallel scenarios with many MPI cores involved. Based on those fundamentals, the modular contact framework was extended to support a broader field of use-cases. For adoption by users, it is necessary to provide at least the most popular contact kinds and features, so that switching existing models from previous contact versions is feasible and thus allowing as many use-cases as possible to benefit from the improved contact framework. In this paper we present two new capabilities of the modular contact that work in that direction: support for eroding contacts and a segment-to-segment contact implementation. In the next two sections, we describe those features and demonstrate their capabilities in numerical experiments.

2 Erosion in Modular Contact

To handle contact between two parts, we need to extract the surface of those two parts and then operate on those surfaces. While the surface for thin shell elements is simply defined by offsetting the shell elements from their mid-surfaces, for volumetric element types like solids or thick shells the surface needs to be extracted explicitly. To this end, we need to determine those *faces* (or *segments*) and their nodes of all solid and thick shell elements that are exposed to the exterior (i.e. the faces that are not connected to any neighboring element). In general, this is done during the initialization step of LS-DYNA for all part-based contact inputs.
Now, in the case of solid or thick shell element failure during the simulation, the contact needs to react to these changes, since new element faces could have been exposed and consequently the exterior surface of the part has changed. *Eroding contact types* are intended to deal with those surface updates. Reacting to the surface changes usually means we need to find solutions to two main problems:

1. Update the contact surface by finding all newly exposed contact segments and/or nodes.
2. Find potential contact partners for those new segments and nodes to be checked for penetration until the next global search (bucket-sort) is performed.

In the "classic" (i.e. non-modular) contact implementations in LS-DYNA, task 1. is usually tackled by pre-computing all possible segments (i.e. exterior *and* interior segments) during initialization but only mark the exterior segments/nodes as active and all interior ones as inactive. Then, after element failure, additional segments and their nodes are marked active/inactive as necessary.
The approach to handle task 2. is to simply perform additional bucket-sorts that find new contact pairs for the updated (and entire) contact surface. This can be done by just choosing a high, regular bucket-sort frequency. Alternatively, additional bucket-sorts outside of the regular bucket-sort frequency might be performed. E.g. the node-to-segment contact (SOFT=0/1) typically performs a bucket-sort immediately in each cycle a solid or thick shell element fails. An alternative strategy, typically used by the segment-to-segment contact (SOFT=2), is to make sure a bucket-sort will happen no later than in δ_B cycles (with δ_B typically being 5). While the first approach should be very robust, it can become very expensive when there is element failure in a high percentage of all cycles. The second approach mitigates these costs by slightly postponing the bucket-sort and hence potentially doing a single bucket-sort for multiple cycles with erosion. But it can still result in a significant number of additional bucket-sorts for problems with a lot of element failure, and there is a slight risk[1] to miss some contact with the newly exposed surface for a couple of cycles.

[1] In practice, this risk is really very low, since the failed element leaves a gap that the surrounding elements usually need to overcome before they can come into contact with the newly exposed surface.

Fig.1: Snapshots of the solution at progressing time steps (from left to right) of Engine Related Impact Failure (ERIF) test cases from the LS-DYNA Aerospace Working Group (AWG).
Top row: Fan Blade-Off Rig Test – AWG ERIF Test Case 2.1[2]
Bottom row: Fan Blade-Off Containment Test – AWG ERIF Test Case 2.2[3]

Nowadays, with meshes becoming ever finer, these approaches are starting to become more and more of a bottleneck. Finer meshes mean that there are not only more elements in general, but refining a mesh makes the number of interior faces in parts meshed by solid elements grow faster than the exterior faces. This means the overhead of pre-allocating all interior segments becomes increasingly significant, not only due to the added memory requirements, but also due to the implication on memory bandwidth that comes with it. In addition to that, more and smaller solid elements lead to them failing faster and more of them failing. Hence, the share of cycles where at least one solid element fails is also increasing, which leads to many additional and expensive bucket-sort steps being performed. To react to those changing circumstances, different approaches to tackle the tasks 1. and 2. have been chosen in the modular contact framework.
Instead of pre-computing all possible segments and nodes to tackle task 1., an *incremental update* of the contact entities is performed for the modular contact. This is, whenever an element erodes, its neighboring elements are checked for newly exposed faces and new segments/nodes are created accordingly. This is a dynamic process and requires highly flexible internal data structures to work efficiently. Task 2. is not solved by additional bucket-sorts, but instead *existing information* about close element relationships *in the vicinity of the failed elements* is used to form new, potential contact pairs. These will be used until the next regular bucket-sort is performed. Since all these steps take local information around the failed elements into account only, the runtime characteristics are only related to the number of failed elements and not to the size of the global problem (in contrast to the globally operating bucket-sorts). This in turn can greatly improve performance while still making sure no contact is missed until the next regular bucket-sort.

2.1 Experiments

To demonstrate the benefits of the new approaches, we are presenting results for models from so-called *Engine Related Impact Failure (ERIF)* tests. More specifically, we look at Fan Blade-Off tests where one blade within a yet engine gets loose from its platform and is crashing into the outer casing of the engine, where it starts to bend, then break, while potentially hitting trailing blades or other parts of the engine. Exemplary results for tests from the *LS-DYNA Aerospace Working Group (AWG)* are shown in Fig.1:.

[2] https://awg.ansys.com/QA+Test+Example+2.1
[3] https://awg.ansys.com/QA+Test+Example+2.2

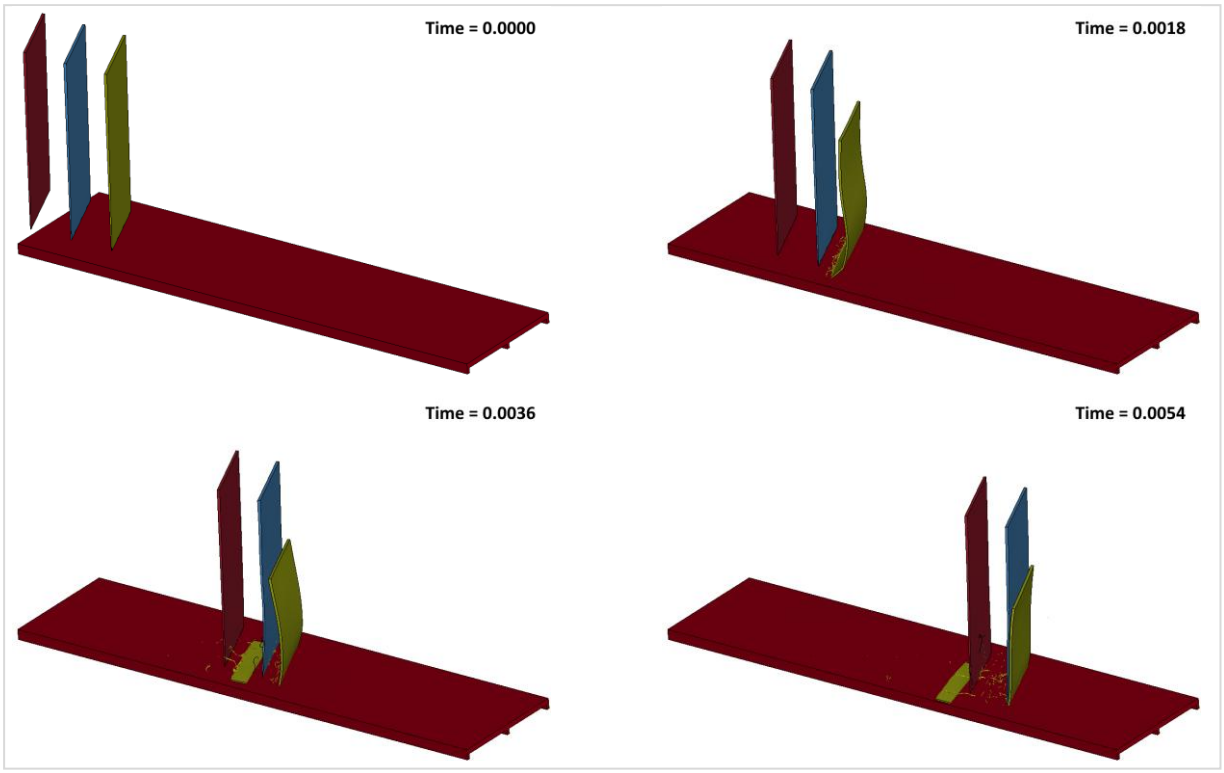

Fig.2: Snapshots of the solution of the simplified & refined Blade-Off test case (row-wise from left to right). The front most blade is released and moves downwards towards the base, where – after hitting the base - it starts to bend, then brake. Eventually, the first trailing blade reaches the released blade and gets affected by the crash, also.

More recent models used in production utilize finer meshes than the publicly available test cases from the LS-DYNA AWG. Hence, in the following we will show performance results for a simplified, but significantly refined version of a Blade-Off containment test case (see Fig.2:). This model consists of a total of 3M hexahedral solid elements, where each blade is made off 800K elements, progressively getting finer towards the downwards pointing tips of the blades. The model incorporates five contact interfaces. A single-surface interface for the front-most (released) blade. An additional surface-to-surface interface to model the contact between the first blade and the base and three more surface-to-surface interfaces to model the contact between each pair of blades.
The model was run three times, using different contact implementations to demonstrate and compare their runtime behavior: with the “classic” MPP node-to-segment (SOFT=1) contact, the “classic” MPP segment-to-segment (SOFT=2) contact and finally with the new node-to-segment modular contact implementation. Each of the runs was executed on a cluster[4] using two compute nodes and a total of 96 CPU cores.

Table 1: Number of total cycles, cycles with erosion and cycles with bucket-sorts for the simplified & refined Blade-Off test case. Comparing the “classic” contact implementations (SOFT=1 and SOFT=2) to the node-to-segment modular contact implementation.

	Total Number of Cycles N_T	Cycles with Erosion N_E	Erosion Cycles Percent N_E/N_T	Cycles with Bucket-Sort N_B	Bucket-Sort Cycles Percent N_B/N_T	Desired Bucket-Sort frequency f_B^*	Effective Bucket-Sort frequency $f_B = N_T/N_B$
“Classic” Contact SOFT=1	333334	145060	43.5%	146019	43.8%	200	2.28
“Classic” Contact SOFT=2	333334	129559	38.9%	73159	21.9%	100	4.56
Modular Contact Node-To-Segment	333334	134399	40.3%	1666	0.5%	200	200.08

[4] Each compute node in the cluster consists of 2x *Intel® Xeon® Gold 6442Y* processors with 24 cores each. Compute nodes are connected via an *InfiniBand: Mellanox ConnectX-6*.

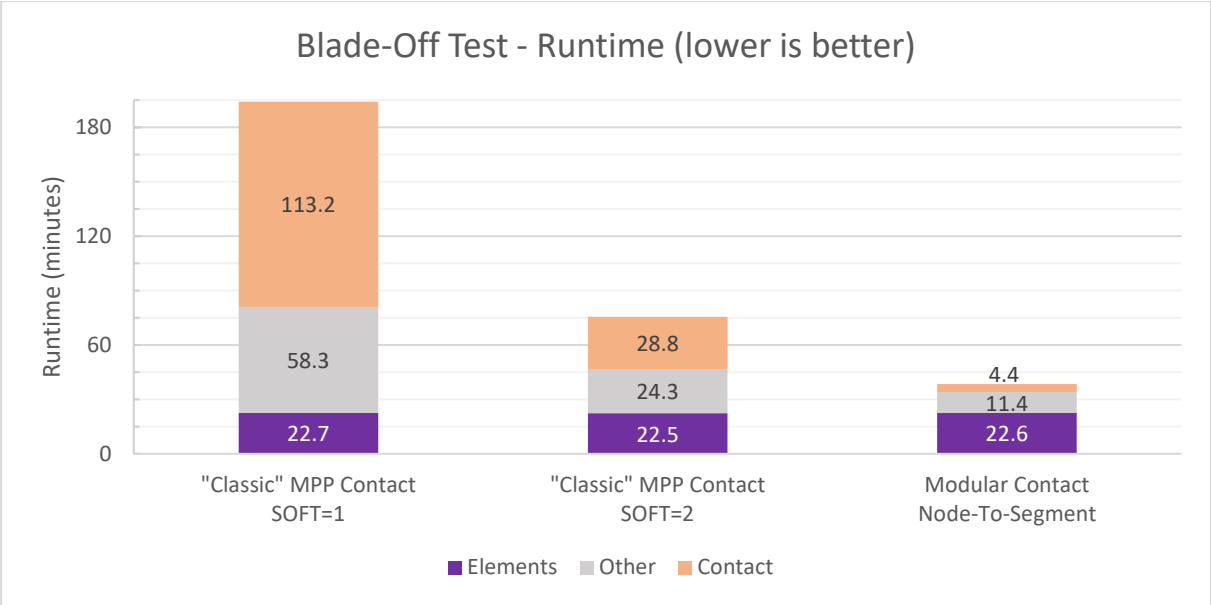

Fig.3: Runtime results for the simplified & refined Blade-Off test case. Comparing the "classic" contact implementations (SOFT=1 and SOFT=2) to the node-to-segment modular contact implementation. Timings for the "Element processing" (Elements) and "Contact algorithm" (Contact) timer regions from the timing summary table are given explicitly. In addition, the sum of all remaining timers is given (Other).

Performance results are depicted in Fig.3:. To better understand those results, we will start by looking at Table 1:. First, we can see that all variants were running for the same total number of cycles N_{T}, which is important to make the results more comparable. In addition to that, all variants have erosion (i.e. element failure) in about the same share of the cycles (~40%). Slight differences arise due to slightly different treatment of the contact. But the biggest difference can be seen in the number of cycles a bucket-sort is performed N_{B}. For the classic SOFT=1 contact, a bucket-sort is performed in every cycle an element fails, in addition to the cycles where a regular bucket-sort is performed due to the default bucket-sort frequency $f_{\mathrm{B}}^{*} = 200$. Hence it is $N_{\mathrm{B}} \geq N_{\mathrm{E}}$ and thus a bucket-sort is performed in more than 40% of all cycles, which results in an effective bucket-sort frequency of $f_B \approx 2.28$ (or in other words, a bucket-sort is performed in pretty much every other cycle). Since these bucket-sorts are very expensive, we can see that the largest part of the total runtime (nearly 60%) is consumed by the contact workload (see Fig.3:). Also, bucket-sorts are usually hard to load-balance, since they operate on the surfaces of all parts involved in the contact. This manifests itself in larger parts of the runtime being accounted to synchronization steps, which are summarized in the "Other" group of the chart in Fig.3:. Overall, the simulation runs for more than 3 hours with the SOFT=1 contact implementation.
Comparing this to the SOFT=2 implementation, we can see a significant drop in the number of bucket-sorts to only about 20% of the total cycles. This results in an effective bucket-sort frequency $f_B = 4.56$, which is close to cycles we allow between an element failure and the next bucket-sort $\delta_{\mathrm{B}} = 5$. Although comparing runtimes between a node-to-segment and segment-to-segment implementation involves more than just the bucket-sort frequency, we can nevertheless see a significant drop in the contact time (nearly 5 times faster than the SOFT=1 run) as well as the total runtime (about 2.5 times faster). But the contact is still the dominate single group in the runtime distribution (about 38% of the total runtime).
Now, in the modular contact implementation, there is no extra bucket-sort performed after element failure. Hence the effective bucket-sort frequency and the (default) desired bucket-sort frequency match up to some rounding, i.e. $f_{\mathrm{B}} \approx f_{\mathrm{B}}^{*} = 200$. As a result of this, the total contact time reduces by an additional factor of 6 compared to the SOFT=2 implementation, which leads to reduction of a factor of 25 (or -96%) compared to the SOFT=1 implementation. Together with reduced load balancing and other details that speed up the modular contact, the total runtime ends up being about 2 times faster than the SOFT=2 implementation and 5 times faster than the SOFT=1 implementation. Also, the contact computations do now only make up a small share of the total runtime at around 11%. Instead, the element routines are becoming the dominant computational work which make up 59% of the total runtime.

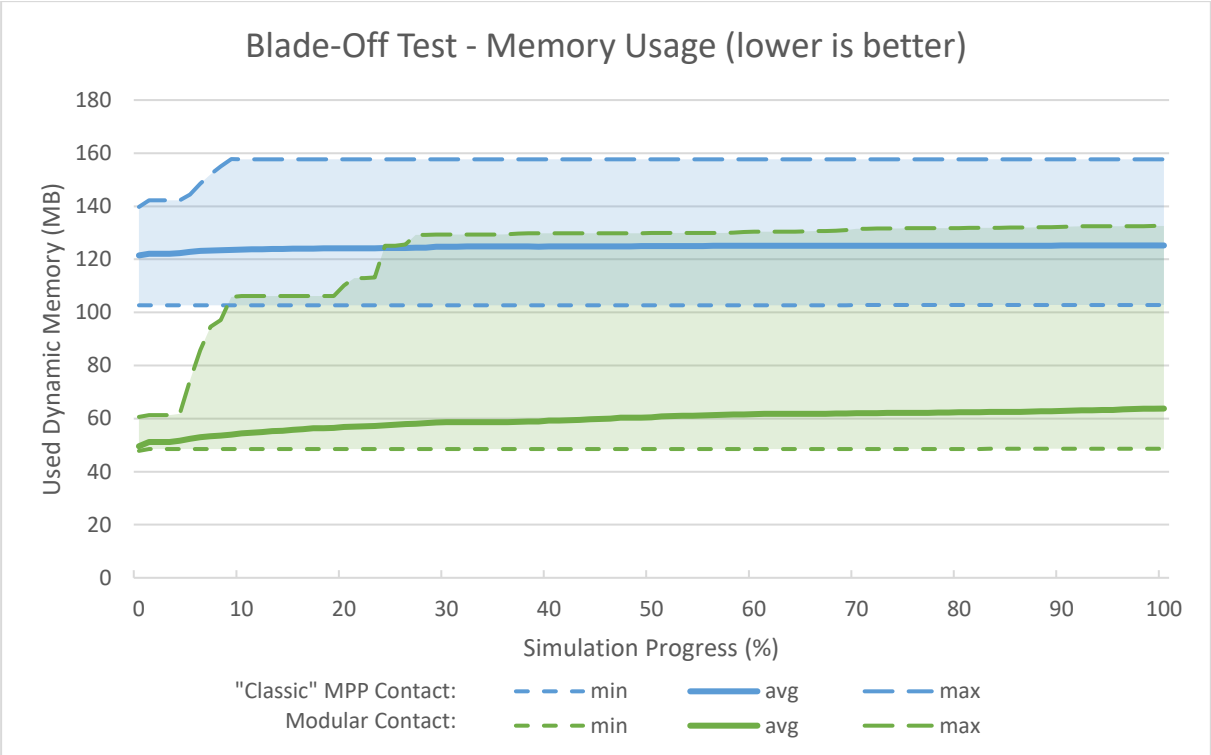

Fig.4: Memory consumption results for the simplified & refined Blade-Off test case over the course of the simulation. Comparing the "classic" (SOFT=1) implementation to node-to-segment modular contact implementation. For each implementation, the range of memory consumption from all MPI ranks is given. Showing the minimum and maximum consumptions by dashed lines and the average consumption as bold line.

Finally, let us look at the memory consumption of the runs. Since node-to-segment and segment-to-segment based implementations have significantly different storage requirements, we will only compare the classic SOFT=1 implementation to the modular node-to-segment implementation. Results are depicted in Fig.4:. In the SOFT=1 case, we can see that the average memory consumption across MPI ranks starts at a relatively high level, slightly increases in the beginning, but then stays mostly constant after about 15% into the simulation, after much of the initial contact was established.
In contrast to that, the first thing to notice is that the modular contact implementation starts a lower base-line consumption. One of the main reasons for this is that the interior segments and nodes are not stored from the beginning. On the other hand, the average consumption keeps increasing slightly throughout all the simulation, since new segments are constantly added. Furthermore, the difference between average and maximum consumption is larger, since only MPI ranks that hold parts where erosion happens are affected by the new segments. In summary, this means that the modular contact will potentially have a lower overall memory consumption, but the variation across ranks might be higher, especially towards the end of the simulation. Note that the effect on memory consumption will likely heavily depend on the specific model, particularly the number of solid elements involved in contact interfaces, the volume-to-surface ratio of the bodies those elements form and the amount of element failure.

3 Segment-to-Segment Based Modular Contact

Modular contact is a framework that hosts different contact implementations. These implementations share code for common operations as described in [1]. This eases development and reduces code duplication. They further share all MPI communication which improves parallel performance in models with several contacts. Contact implementations differ according to the entities that are tested for contact. Previously modular contact had two contact implementations. A node-to-segment contact, where contact forces are computed between a node of the A-side surface and a segment of the B-side surface. And an edge-to-edge contact where edges are considered on each surface. A third contact type available in LS-DYNA is segment-to-segment based contact, accessible by setting SOFT=2, where segments are considered on both surfaces for contact computation. By using the actual segments this contact type computes more accurate contact forces and allows contact between edges of segments to be properly handled. Due to that it is popular especially in crash simulation. In this paper we introduce

a segment-to-segment based contact implementation in modular contact (MC-STS), such that modular contact now provides most popular contact types available in LS-DYNA.

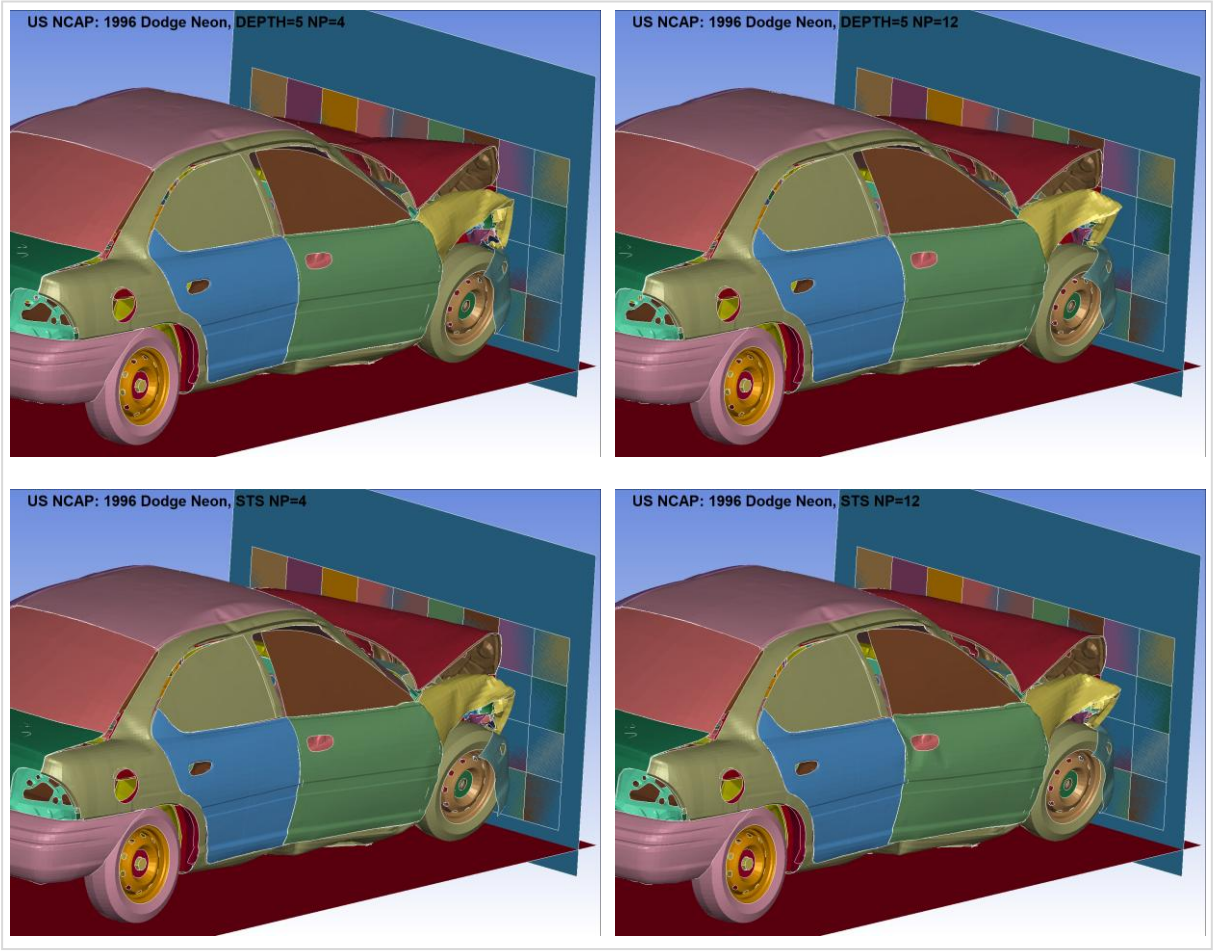

Fig.5: Final state of the NCAP 1996 Dodge Neon crash model for the SOFT=2 DEPTH=5 contact (top row) and the MC-STS contact (bottom row), executed with 4 MPI processes (left) and 12 MPI processes (right).

SOFT=2 contact itself has an option that fundamentally changes how penetration between two segments is determined: the DEPTH option. At least in crash simulation, DEPTH=5 is widely used as the default version. For backwards compatibility, major improvements to the DEPTH=5 option resulted in options DEPTH=25, 35 and 55. The latter, DEPTH=55, provides more accurate detection of the intersection area of segments and thus more accurate surface contact forces. It further improves detection of edge contact. To achieve this, DEPTH=55 does more work than DEPTH=5 and is thus slower. To build on these incremental improvements, we based the new MC-STS contact implementation on the same principles as the DEPTH=55 option. Note however that it does not try to replicate the exact same contact response.

3.1 Experiments

3.1.1 NCAP 1996 Dodge Neon Crash

The segment-to-segment contact implementation in modular contact (MC-STS) is meant as a drop-in replacement for SOFT=2 contacts in simulations. While it does not aim to replicate the exact contact response of any existing SOFT=2 contact version, it should capture the same overall behavior. A good measure for this is that switching to MC-STS should not change results more than using a different number of processes with an existing implementation. With our first experiment we demonstrate that this is indeed the case.

To this end, we revisit version 0.7 of the NCAP 1996 Dodge Neon car crash model that was already used in [1] to test the node-to-segment modular contact implementation. This model is publicly available from the US National Highway Traffic Safety Administration (NHTSA) [2]. It consists of 270804 elements and simulates a frontal car crash into a rigid wall at 35mph. In the original model, a single SOFT=0

(node-to-segment) single-surface contact is used, which we replace by either a SOFT=2 contact with DEPTH=5 or the new MC-STS contact. For comparison, we run the simulation with 4 and 12 processes. Fig.5: shows the final state of each of the four simulations. Visually, differences are of the same magnitude when comparing results between different numbers of processes (columns) and different contact implementations (rows).
We further compare the results through characteristic curves extracted from the simulation results. Fig.6: shows the average velocity and acceleration of the two front seats, the accelerations of a reference point at the bottom and the top of the engine respectively, the total wall force and the vehicle displacement. In addition to the versions with the new and existing segment-to-segment contact, we add the results of a single node-to-segment (NTS) run with four processes, since the Neon crash model was calibrated for that kind of contact. We use the modular contact implementation here.
In Fig.6: we can observe that indeed all contact types capture the same overall behavior in this crash simulation. All segment-to-segment contacts have more oscillations in the computed accelerations than the node-to-segment contact, which may be due to switching between surface and edge contact internally. Oscillations have already improved slightly with MC-STS, but should be addressed in future work on the new modular contact implementation. We did not include results for the DEPTH=55 contact as the plot is crowded enough already. However, its characteristic curves follow the trendline of MC-STS but with more oscillations.

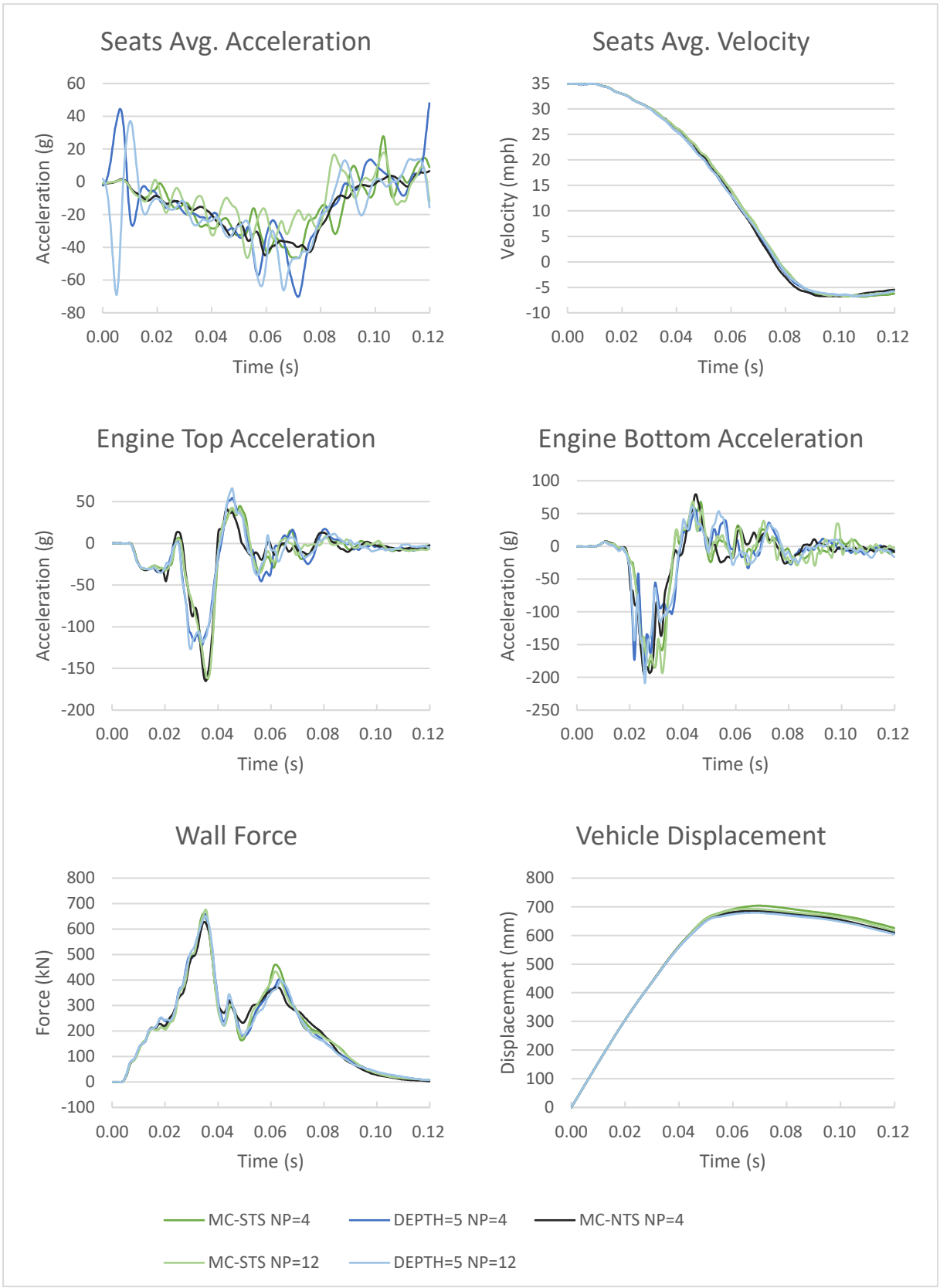

Fig.6: Acceleration, velocity, force and displacement results from the NCAP 1996 Dodge Neon runs, comparing the SOFT=2 DEPTH=5 contact implementation to the new segment-to-segment modular contact implementation (MC-STS). Each has been run with 4 and 12 MPI processes. For comparison, we included results of the modular node-to-segment contact (MC-NTS). An SAE-J211 compliant low pass filter according to [3] has been applied to the raw acceleration and force data.

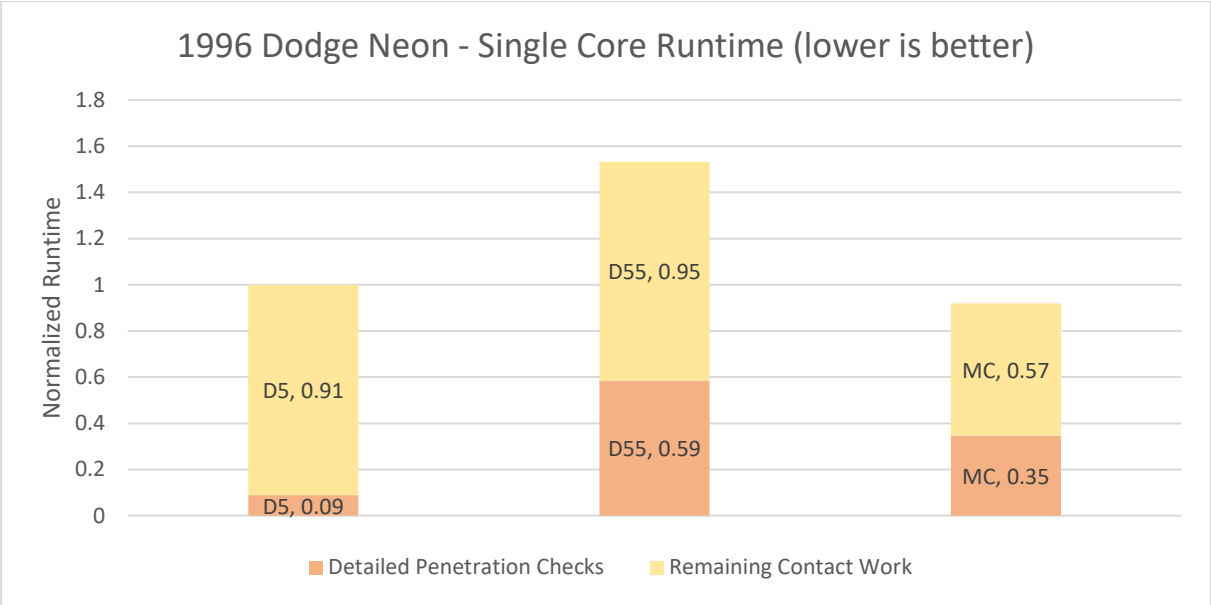

Fig.7: Contact times for the two SOFT=2 contacts with DEPTH=5 (D5) and DEPTH=55 (D55) and the new modular segment-to-segment contact implementation (MC) of a simulation of the 1996 Dodge Neon model on a single core.

Once results are correct, runtime performance of a contact implementation is critical. We use the Neon model to look at the single core performance of the new implementation, before we investigate parallel performance with the next experiment. As described earlier, the MC-STS contact implementation is based on the same principles as the DEPTH=55 option in SOFT=2 contact, which invests more work to improve the computed contact forces. Fig.7: shows the runtime of the DEPTH=55 and the MC-STS contact relative to the DEPTH=5 contact. We show the time taken by segments that are checked for contact in detail and time for the remaining contact work separately. Clearly, detailed checks for penetration are significantly faster in DEPTH=5 contact, while the new modular contact implementation manages to get rid of a substantial amount of the overhead compared to DEPTH=55. We want to note here, however, that both the DEPTH=55 and MC-STS contacts perform detailed penetration checks for more segment pairs. That is, the increased runtime is a combination of more expensive checks and more careful consideration of contact. Next, we can observe that MC-STS has only two thirds of the remaining contact work time compared to the SOFT=2 contacts. As we will see in the next experiment, this has a greater impact on overall runtime in large, highly parallel simulations than the detailed penetration checks, at least in crash type applications.

3.1.2 Large full car crash (ODB-10M)

Modular contact promises improved distributed parallel performance due to its MPI communication scheme. To demonstrate this also for the new segment-to-segment implementation, we run the ODB-10M benchmark model, in which a refined NCAC Ford Taurus model crashes into a modified LST shell offset deformable barrier (ODB), see Fig.8:. This model with slightly more than ten million elements was developed in [4] and is publicly available.
Regarding contacts, the model includes a large segment-to-segment single-surface contact for the car body, which contributes over half of the total contact runtime, and seven smaller node-to-segment contacts mainly in the barrier. For our benchmarks, we created three versions of the model by choosing different contact implementations while keeping the contact options the same. We refer to a version by the implementation chosen for the car body contact. The first two versions have a SOFT=2 car body contact with either DEPTH=5 or DEPTH=55 and the remaining contacts are chosen to be groupable, as this turned out to be the most performant combination prior to modular contact. In a third version, we replaced all contacts with their modular contact implementations. This should highlight an advantage of modular contact, since it now provides both segment-to-segment and node-to-segment implementations that can share their MPI communication steps. Consequently, we do not measure the runtime of only the MC-STS contact in this experiment but rather try to measure the performance impact of switching a whole model to modular contact. Final simulation results of the DEPTH=5 and the MC-STS version are shown in Fig.8:. Overall, both look similar, but of course some shells fold differently due to the evolution of small differences over thousands of time steps.

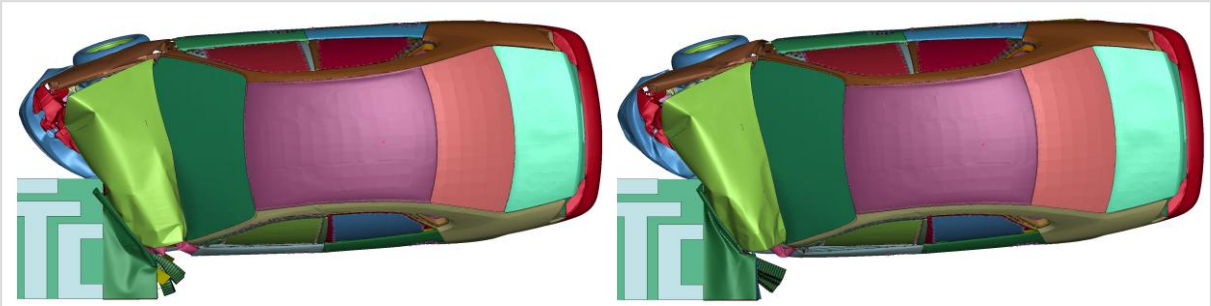

Fig.8: Final simulation result of the ODB-10M model. DEPTH=5 on the left, MC-STS on the right.

Regarding the runtime, we cannot simply state the average contact time provided in the final timing summary table, since wait time due to load imbalance shows up in different timing regions for the SOFT=2 contacts and modular contact, see [1]. Essentially, the first timing region forcing a synchronization of the MPI processes after some work shows the wait time created by that work. In modular contact, the contact work is interleaved with the element work and imbalance of both steps shows up in the "Time Step Sharing" region. In the SOFT=2 contacts, some wait time is already included in the contact timing region and the rest shows up in either "Others", "Force Sharing" or some part of "Rigid Bodies", which we summarize by "rigid bodies+". While we cannot know what fraction of these regions contact related wait time is, we can use the fact that wait time shows up in different regions for different contact implementations as follows. Let $T^{\mathrm{d5}}_{\mathrm{contact}}$, $T^{\mathrm{d55}}_{\mathrm{contact}}$ and $T^{\mathrm{mc}}_{\mathrm{contact}}$ denote the mean contact times over all processes of the model version with the DEPTH=5, DEPTH=55 and modular contact respectively. Let further $T^{\mathrm{d5}}_{\mathrm{time\ st.\ sh.}}$, $T^{\mathrm{d55}}_{\mathrm{time\ st.\ sh.}}$, $T^{\mathrm{mc}}_{\mathrm{time\ st.\ sh.}}$, $T^{\mathrm{d5}}_{\mathrm{rigid\ bodies}}$, $T^{\mathrm{d55}}_{\mathrm{rigid\ bodies}}$ and $T^{\mathrm{mc}}_{\mathrm{rigid\ bodies}}$ denote the "time step sharing" and "rigid bodies+" times of each version. We can now estimate the actual contact runtimes of the simulations by

$$\tilde{T}^{\mathrm{d5}}_{\mathrm{contact}} := T^{\mathrm{d5}}_{\mathrm{contact}} + T^{\mathrm{d5}}_{\mathrm{rigid\ bodies}} - T^{\mathrm{mc}}_{\mathrm{rigid\ bodies}}\ , \tag{1}$$

$$\tilde{T}^{\mathrm{d55}}_{\mathrm{contact}} := T^{\mathrm{d55}}_{\mathrm{contact}} + T^{\mathrm{d55}}_{\mathrm{rigid\ bodies}} - T^{\mathrm{mc}}_{\mathrm{rigid\ bodies}}\ , \tag{2}$$

and

$$\tilde{T}^{\mathrm{mc}}_{\mathrm{contact}} := T^{\mathrm{mc}}_{\mathrm{contact}} + T^{\mathrm{mc}}_{\mathrm{time\ st.\ sh.}} - T^{\mathrm{d5}}_{time\ st.\ sh.}. \tag{3}$$

Timing results for runs of the first 100,000 cycles of the three versions of the ODB-10M model on a cluster[5] are shown in Fig.9:. First of all, we can observe that the DEPTH option has a significantly smaller impact on the runtime than in the previous Neon model, since $\tilde{T}^{\mathrm{d55}}_{\mathrm{contact}}$ is only 14 percent higher than $\tilde{T}^{\mathrm{d5}}_{\mathrm{contact}}$ with 192 processes. Naturally, the gap increases with higher core counts, as slower contact produces more imbalance. In comparison to the NEON model, the ODB-10M simulation spends more of its time handling segment pairs that are close but not yet in contact relative to computing contact forces, simply by having smaller elements. Modular contact can do this more efficiently and together with the improved MPP communication it is almost twice as fast as the DEPTH=5 contact using four compute nodes. With four and 8 nodes, modular contact is as fast as using the DEPTH=5 contact with twice the number of processes. We want to reiterate here that modular contact provides the same improved contact detection from the DEPTH=55 option while being significantly faster than the original DEPTH=5 option of SOFT=2 contact.

[5] Each compute node in the cluster consists of 2x *Intel® Xeon® Gold 6442Y* processors with 24 cores each. Compute nodes are connected via an *InfiniBand: Mellanox ConnectX-6*.

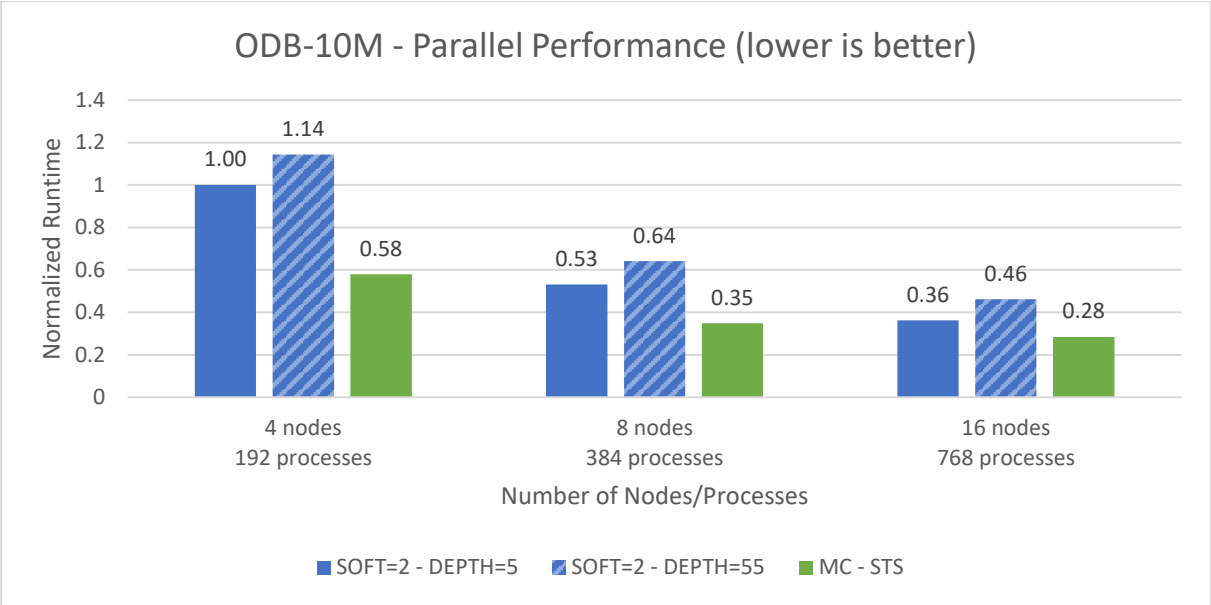

Fig.9: Contact times for the DEPTH=5, DEPTH=55 and the MC-STS contact implementations according to (1), (2) and (3) for the first 100000 cycles of the ODB-10M model with 192, 384 and 768 processes. Times have been normalized to the DEPTH=5 contact time with 192 processes, and we report averages of multiple runs.

It may seem as though the two SOFT=2 contact implementations are scaling better with increasing number of processes in this example. We would argue that instead modular contact is simply closer to the scaling limit set by the domain decomposition already with fewer cores. As discussed in [1], a decomposition with better volume-to-surface ratio for process regions would be required to scale into higher core counts for this model. Alternatively, we can try a larger model with the same number of processes to prove this point. To this end, we ran a car crash model similar to the ODB-10M model with 30M elements. Fig.10: shows the contact runtime results for this problem. Now with more elements the MC-STS contact needs only about a third of the contact time of the DEPTH=5 contact with eight nodes. Additionally, it scales perfectly to 16 nodes with a speedup of 2, while the DEPTH=5 contact achieves a speedup of only 1.49.

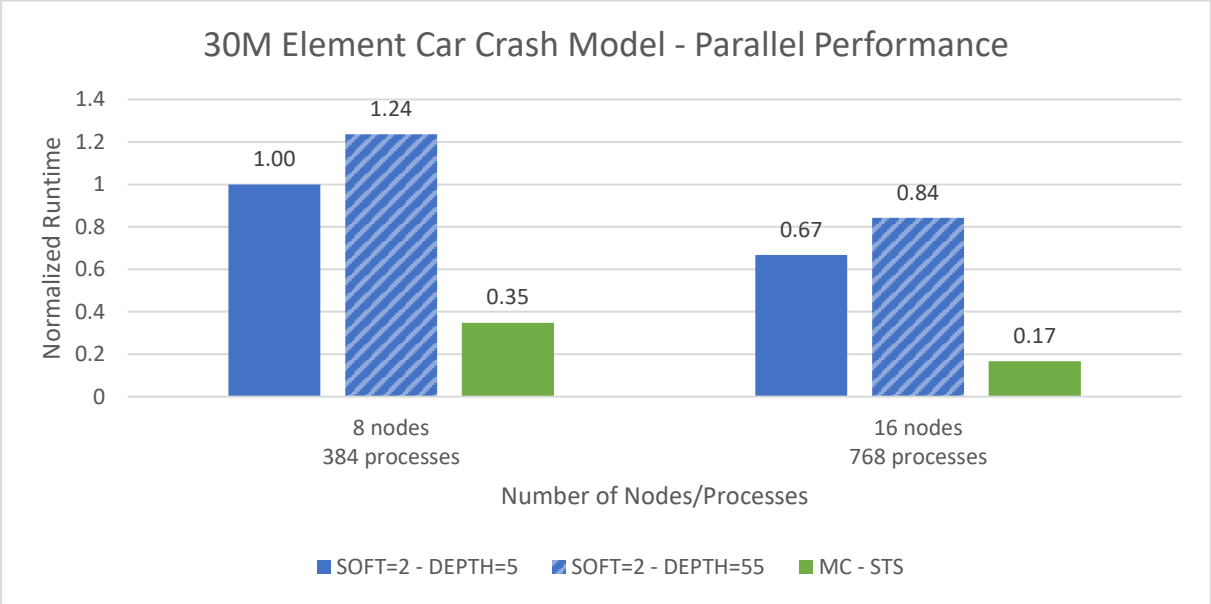

Fig.10: Contact times for the DEPTH=5, DEPTH=55 and the MC-STS contact implementations according to (1), (2) and (3) for the first 100000 cycles of a car crash model similar to the ODB-10M model with 30M elements. Times have been normalized to the DEPTH=5 contact time with 384 processes, and we report averages of multiple runs.

Note that improving the scaling behavior for the smaller ODB-10M model is possible, but would require using a different approach to domain decomposition, as was demonstrated in [1]. Improving the scaling behavior of a problem with a different domain decomposition is significantly easier with modular contact as there is no MPI synchronization between element and contact work. Consequently, a decomposition only has to balance the sum of element and contact work on each process instead of balancing both at the same time.

4 Summary

We introduced two new features added to the modular contact framework in LS-DYNA: support for erosion of solid and thick shell elements and a new segment-to-segment contact implementation. By choosing different strategies to tackle the challenges to deal with element failure, the modular contact enables significant runtime savings compared to previous contact implementations on modern, fine models. This has been demonstrated in numerical experiments from engine related impact failure tests. With the segment-to-segment implementation, the modular contact framework now provides all major contact types in LS-DYNA so that it is now easier to switch existing models to the modular contact framework. We compared the modular to the classic (SOFT=2) segment-to-segment contact in several car crash simulations. Like the results previously presented for the node-to-segment contact, the new segment-to-segment implementation gives comparable results to its classic counterpart, while offering substantial runtime and scalability improvements in large, highly parallel simulations.

5 Literature

[1] A. Ziegenhagel and B. Warinscott, "Modular Contact: A new approach to contact in LS-DYNA," in *17th International LS-DYNA® Users Conference*, Detroit, 2024.

[2] National Highway Traffic Safety Administration (NHTSA), "Crash Simulation Vehicle Models," [Online]. Available: https://www.nhtsa.gov/crash-simulation-vehicle-models. [Accessed 28 August 2025].

[3] Transportation Research Board and National Academies of Sciences, Engineering, and Medicine, Procedures for Verification and Validation of Computer Simulations Used for Roadside Safety Applications, 2011.

[4] M. Makino and S. Yamada, "ODB-10M New Topcrunch Benchmark Data," *8th European LS-DYNA® Users Conference,* 2011.