Maximize LS-DYNA MPP Performance

Eric Day[1], Jason Wang[1]

[1]Ansys LST part of Synopsys

1 Abstract

This paper presents proven practices for maximizing performance in LS-DYNA MPP simulations across diverse hardware platforms. It discusses how to select the most suitable LS-DYNA binary for your system, along with best practices for applying undersubscription and hybrid LS-DYNA to enhance parallel scalability. Methods for identifying parallel inefficiencies and addressing them through user-defined domain decomposition are also covered. In addition, the paper highlights recent LS-DYNA benchmarks on the latest x86 and Arm processors and cloud instances.

2 Achieve Efficient Scaling

Selecting the appropriate LS-DYNA binary is essential to achieving optimal performance and must be aligned with the characteristics of the underlying hardware. Important considerations include the ratio of model size to available cores, the processor vendor, and the supported instruction set extensions (ISEs).

On Intel platforms, Intel-compiled LS-DYNA (ifort or IFX) is the preferred choice. These binaries support SSE2, AVX2, and AVX512 instruction sets. For explicit simulations, AVX2 commonly yields a performance improvement of approximately 10% over SSE2, whereas the benefits of AVX512 are workload-dependent.

On AMD platforms, both Intel-compiled and AMD-compiled (AOCC) LS-DYNA binaries are available. In certain cases, AOCC binaries provide superior performance on AMD hardware, as illustrated in Figure 1. The optimal selection depends on the specific hardware configuration, model characteristics, and solver type. AVX2 remains a dependable option on AMD processors, while AVX512 is available only on Zen4 and Zen5 architectures.

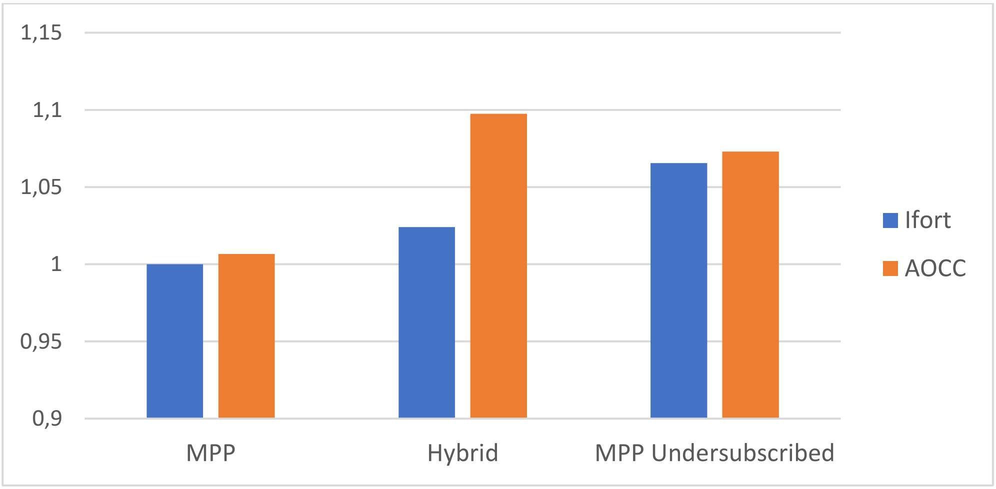


Fig.1: LS-DYNA R16.0 AVX2 normalized performance on Microsoft Azure HBv5 – ODB10M

Intel MPI or Open MPI are the recommended choices for running LS-DYNA. In contrast, Platform MPI is based on the older MPI-2 standard and is no longer maintained. As a result, LS-DYNA features that depend on newer MPI standards are not supported with Platform MPI. In addition, Platform MPI generally delivers weaker scaling performance compared to Intel MPI and Open MPI, as shown in Figure 2.

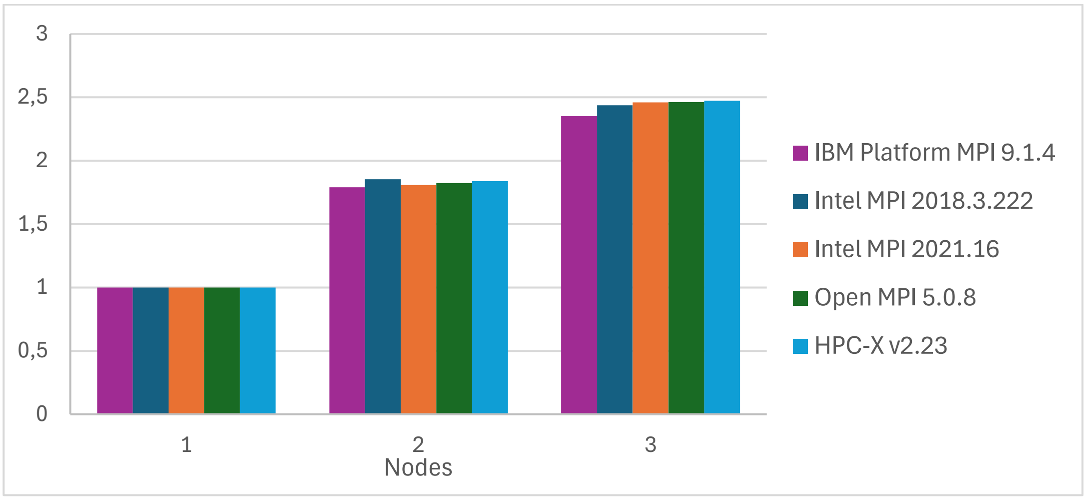


Fig.2: LS-DYNA R16.1 speedup for various MPI on Intel 8568Y+ InfiniBand 2xNDR – ODB10M

Simultaneous Multithreading (SMT) can deliver an additional ~10–15% performance improvement when utilizing an entire system. However, this gain comes at the expense of doubling the license usage. For this reason, it is generally recommended to disable hyperthreading and instead take advantage of additional physical cores, which provide superior speedup for the same license cost.

LS-DYNA also benefits from high clock speeds. Enabling boost clock (turbo frequency) ensures the processor operates at its maximum available frequency, improving overall simulation performance.

The partition file (pfile) offers several options to enhance efficiency. Defining a fast local scratch directory—for example, an NVMe device—using *directory { local /path/to/local/dir }* on a cluster can reduce network latency and mitigate contention for shared filesystems. Depending on the number of ranks and the underlying filesystem speed, this approach can improve performance by 1–5%. Additional efficiency gains can be achieved by suppressing unnecessary file writes and transfers through the options *general { nofull nodump nobeamout }*.

As a rule of thumb, explicit LS-DYNA MPP scales most efficiently when each MPI rank is assigned at minimum 10,000–20,000 elements, ensuring a favorable balance between computation and MPI communication. For instance, a model with 10 million elements should typically be run on no more than ~500 MPI ranks for good efficiency; additional cores will give diminishing returns due to communication overhead. If your model approaches or falls below this threshold—or if you simply have more cores at your disposal—undersubscription or hybrid LS-DYNA can help maintain scalability and make better use of available resources.

2.1 Undersubscription

Undersubscription refers to deliberately running on fewer cores than are physically available, while still taking advantage of the system's full resources. By spacing processes evenly across the machine—using the binding method shown below—we can ensure that each MPI rank accesses a distinct slice of available L3 cache, while also reducing contention for memory channels. This approach can deliver several benefits, including higher memory bandwidth and L3 cache per core, lower MPI communication overhead, improved thermal and power efficiency, and reduced licensing costs.

Intel MPI Undersubscribed Process Pinning

mpirun -genv I_MPI_PIN_DOMAIN=1:spread -np <*ranks*> …

Open MPI Undersubscribed Process Pinning

mpirun --map-by ppr:<*rank per numa*>:numa --bind-to-core -n <*ranks*> …

The binding method also interacts closely with LS-DYNA's domain decomposition. The approach shown above spaces out MPI ranks while ensuring that adjacent ranks are placed physically close to each other within the NUMA hierarchy. This hardware locality is important because ranks responsible for connected subdomains—which must frequently exchange velocities, forces, contact data, and other information—benefit from lower-latency communication and reduced risk of remote memory access. In contrast, some round-robin pinning strategies scatter adjacent ranks across different NUMA domains or sockets in an alternating pattern, which can increase communication costs and degrade performance.

Undersubscription is particularly effective on AMD systems, where memory bandwidth per core is often a limiting factor. Figure 3 represents normalized performance on full and undersubscribed Amazon Elastic Compute Cloud (EC2) instances with Elastic Fabric Adapter (EFA). As illustrated in the figure, full core subscription may still offer the best performance and cost efficiency at lower core counts. However, in larger HPC environments, undersubscription often delivers superior performance and improved scalability.

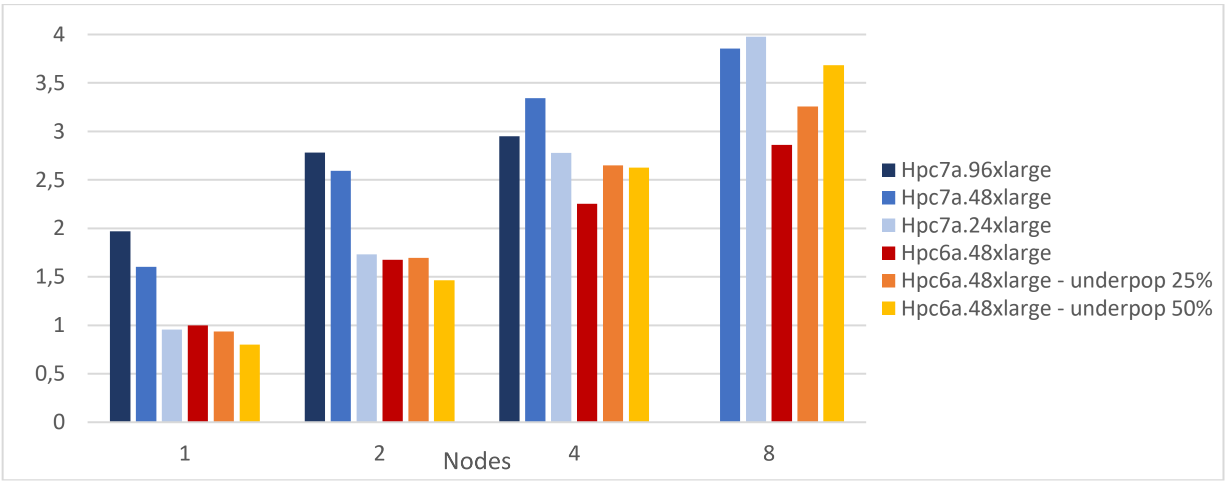


Fig.3: LS-DYNA R13.1.1 normalized performance on undersubscribed AMD Amazon EC2 instances with EFA – ODB10M

2.2 Hybrid LS-DYNA

Hybrid LS-DYNA combines the shared-memory parallelism of OpenMP (as used in LS-DYNA SMP) with the distributed-memory parallelism of MPI (as used in LS-DYNA MPP). At high core counts, pure MPP can suffer from significant MPI communication overhead. Hybrid mitigates this by assigning multiple OpenMP threads to each rank while running on fewer total ranks. Within a rank, threads operate in shared memory, reducing memory access latency and improving cache utilization. The reduced number of ranks exchange data via MPI, which lowers communication costs compared to pure MPP.

The recommended process and thread pinning strategy for hybrid LS-DYNA is described below. Figure 4 highlights the enhanced speedup achievable with hybrid while running on the same number of total cores as pure MPP.

Intel MPI Hybrid Process Pinning

export OMP_NUM_THREADS=<*threads*>

mpirun -genv I_MPI_PIN_DOMAIN=${OMP_NUM_THREADS}:spread -np <*ranks*> … ncpu=-${OMP_NUM_THREADS}

Open MPI Hybrid Process Pinning

export OMP_NUM_THREADS=<*threads*>

mpirun --map-by ppr:<*ranks per numa node*>:numa:PE=${OMP_NUM_THREADS} --bind-to-core -n <*ranks*> … ncpu=-${OMP_NUM_THREADS}

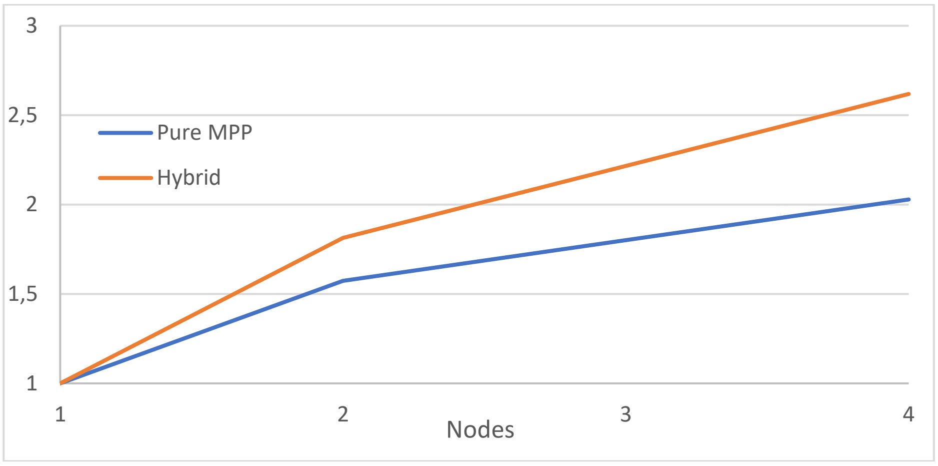


Fig.4: LS-DYNA R14.1.0 ifort190 hybrid speedup on Azure HB176-144rs_v4 – ODB10M

A further advantage of hybrid is solution consistency as core counts increase. With pure MPP, raising the number of ranks changes the domain decomposition, which can introduce small numerical differences during reductions that compound over many LS-DYNA explicit cycles [1]. In contrast, hybrid, in consistency mode, allows users to keep the same decomposition (with same number of ranks)—and therefore identical results—while scaling up by simply increasing the number of SMP threads per rank. Consistency mode is enabled on the command line by listing the number of threads as negative (i.e., ncpu=-<threads>).

2.3 Alternative Methods to Fully Subscribed Pure MPP

Having discussed undersubscription and hybrid configurations in detail, refer again to Figure 1. This figure highlights the performance gains achievable by choosing the optimal LS-DYNA binary and execution method—without requiring any changes to your LS-DYNA model or hardware.

3 Optimize Models with Domain Decomposition

In MPP and Hybrid LS-DYNA, the model is divided into subdomains, each assigned to an MPI rank. The subdomains are created using Recursive Coordinate Bisection (RCB) based on the model's dimensions and the relative cost of elements within [2]. Each rank processes the elements in its subdomain in parallel. Before proceeding to the next section of the code, MPI synchronization occurs across all processors, which can only happen once every rank has completed its work. If some ranks take longer than others, processors may sit idle, reducing overall efficiency. Optimal performance is achieved when the workload is evenly distributed across all ranks—load balance—limiting processor idle time.

3.1 Detecting Load Imbalances

A practical way to assess load balance in LS-DYNA is through the load_profile.xy and contact_profile.xy files, which are written at the end of a simulation. When loaded in LS-PrePost, these profiles show the percentage of time each processor spends on various computation categories and contact interfaces, respectively. A well-balanced decomposition of a model will display a roughly even distribution of work across all processors.

3.2 Addressing Load Imbalances

If a load imbalance is detected, LS-DYNA provides several **`*CONTROL_MPP_DECOMPOSITION`** keywords to redistribute work. Figure 5 illustrates an example load profile with poor load balance in shell processing [2]. The imbalance is caused by **`*MAT_USER`** shells, which LS-DYNA cannot reliably assign a computational cost for balanced decomposition of the model. In this case, the keyword **`*CONTROL_MPP_DECOMPOSITION_PARTSET_DISTRIBUTE`** is used to distribute the part set containing the **`*MAT_USER`** elements across all processors. This balances the load, improving performance by 35% in this case.

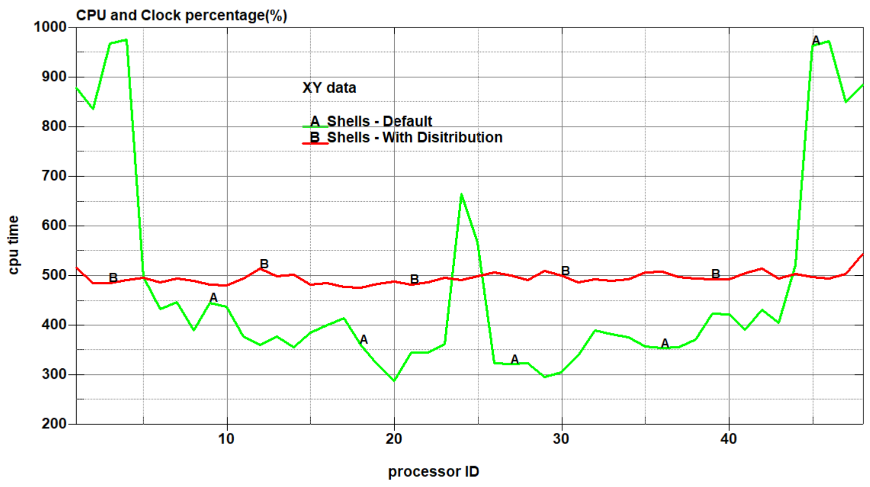


Fig.5: load_profile.xy – shell processing before and after distribution. Reproduced from [2].

Other decomposition keywords of note include **`_ARRANGE_PARTS`**, which functions similarly to **`_PARTSET_DISTRIBUTE`** but with the added control of specifying the number of ranks assigned to the distribution. This is particularly useful for large-scale runs, where assigning too many ranks could introduce unnecessary MPI overhead. **`_DISTRIBUTE_SPH_ELEMENTS`** and **`_DISTRIBUTE_ALE_ELEMENTS`** distribute SPH and ALE elements across all processors. This is advantageous because these elements are more computationally expensive than standard shell or solid elements. The **`_BAGREF`** option decomposes airbags based on their reference geometry rather than their initial folded geometry. This approach helps keep adjacent fabric elements on the same processor, improving the efficiency of particle-to-fabric contact [2]. Additional decomposition keywords are documented in the *Ansys LS-DYNA User's Manual*.

3.3 Optimizing Contact Distribution

`*CONTROL_MPP_DECOMPOSITION_CONTACT_DISTRIBUTE` is another important keyword for load balancing. It distributes elements of specified contact interfaces across all processors. MPP contact in LS-DYNA is processed sequentially—only one contact is handled at a time—which can lead to processor idle time if many small contacts exist. For optimal scaling, it is recommended to use a few large contact definitions that cover the entire model rather than numerous small contacts that only a subset of processors can process at once.

If the purpose of adding many small contact definitions is to measure forces at specific locations, a better approach is to merge them into larger contact definitions and use **`*CONTACT_FORCE_TRANSDUCER`** to capture the forces at specific locations. This method is far less expensive while still providing the required force measurements.

3.4 Defining Special Domain Decomposition

Building on the concepts above, we can make smarter choices for efficient processing with user-defined domain decomposition. Ideally, many—or all—processors should share the workload of large contact definitions. This ties directly to element processing, since elements in contact are often those experiencing high deformation. Because inelastic behaviour is more expensive to compute than elastic, evenly distributing highly deformed elements across processors is key to achieving a good load balance.

For this reason, models are often best decomposed into slices perpendicular to the impact surface or along the direction of initial velocity.

User-defined decomposition can be set up with **`*CONTROL_MPP_DECOMPOSITION_TRANSFORMATION`**, which temporarily transforms the coordinate system only for the decomposition step [1]. Figure 6 shows an example where the keyword is used to rotate coordinates about the z-axis by 30°, then scale the y-coordinates by a factor of 100 [2]. If the scaling factor exceeds the number of processors, the model is partitioned into slices spanning the full model length. The resulting transformed decomposition achieves higher performance by balancing the distribution of elements that undergo inelastic behaviour, while also enabling parallel processing of the contacts between the occupant and vehicle interior.

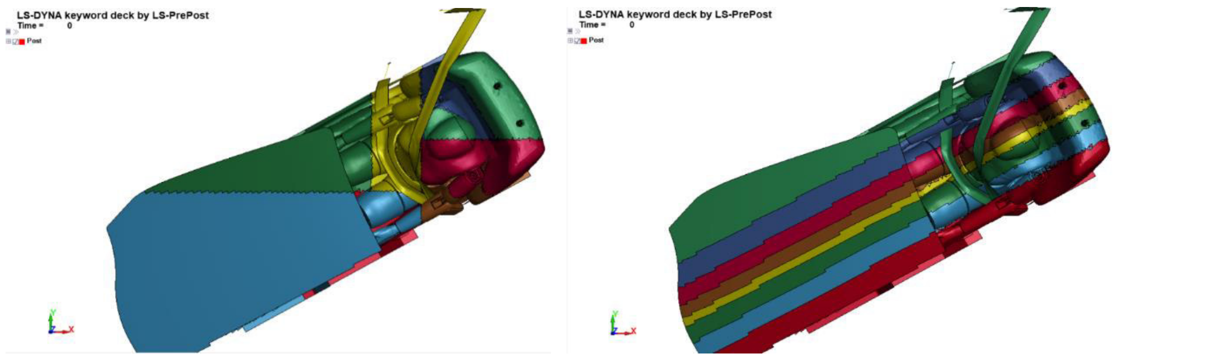

LS-DYNA keyword deck by LS-PrePost
Time = 0
Post
LS-DYNA keyword deck by LS-PrePost
Time = 0
Post

Fig.6: Interior model with occupant – default and transformed decomposition. Reproduced from [2].

3.5 Defining Special Domain Decomposition Using the Partition File.

The pfile is a powerful tool for defining custom decomposition. In fact, all **`*CONTROL_MPP_DECOMPOSITION`** keywords discussed earlier can be reproduced in the pfile—with even greater control.

Decomposition in the pfile is defined below [1]. First specify the region (area, parts, or contact interfaces) to be decomposed, apply any transformations, and finally assign the grouping—the number of processors involved in the region. The remainder of the model not defined in any region receives the remaining transformation. Multiple regions can be decomposed separately through the pfile, which is especially useful for models with angled contacts, such as in an oblique crash.

Partition File format for user-defined decomposition [1]

```
decomposition {
region { <region specifiers> <transformation> <grouping> }
region { <region specifiers> <transformation> <grouping> }
<transformation>
}
```

A proven approach is to assign a specific number of processors to a contact interface—similar to **`_CONTACT_DISTRIBUTE`**, but with the added flexibility of controlling how many processors are used. This is particularly valuable in large-scale runs, where allocating too many ranks to a small contact region can create unnecessary MPI overhead.

Further details on defining special decomposition through the pfile can be found in Appendix O of the *Ansys LS-DYNA User's Manual*.

3.6 Redecomposition

By default, LS-DYNA performs domain decomposition once at the start of the simulation. However, this static approach can lead to scaling issues in models with severe deformation or significant relative motion between parts. A decomposition that is well-suited at the beginning may become inefficient as the simulation progresses.

The keyword **`*CONTROL_MPP_DECOMPOSITION_REDECOMPOSITION`** allows LS-DYNA to periodically adjust the partitions during runtime. At user-defined intervals, the model is redecomposed based on the cost profile of parts, where costs are sampled at a specified frequency. The goal is twofold: to keep interacting parts and neighbouring elements assigned to the same processor, and to redistribute heavily deformed, expensive elements more evenly across processors.

Redecomposition is particularly beneficial for simulations involving large relative motion, such as a car wading through water or a bird strike on spinning turbine blades. In these cases, redecomposition helps maintain efficient parallel performance throughout the simulation.

4 Utilize the Latest Hardware

LS-DYNA explicit performs best on hardware with high clock speed, large L3 cache, high memory bandwidth, and low memory latency. Performance also scales with the number of cores per socket, provided that sufficient elements per core are maintained. Among x86 CPUs, the latest AMD EPYC Zen5 (9375F, 9575F, 9655, 9755) and Intel Xeon6 (6737P, 6741P, 6962P) offer strong performance characteristics for LS-DYNA. Figures 7 and 8 illustrate measured performance improvements for LS-DYNA on recent Intel and AMD architectures [3][4].

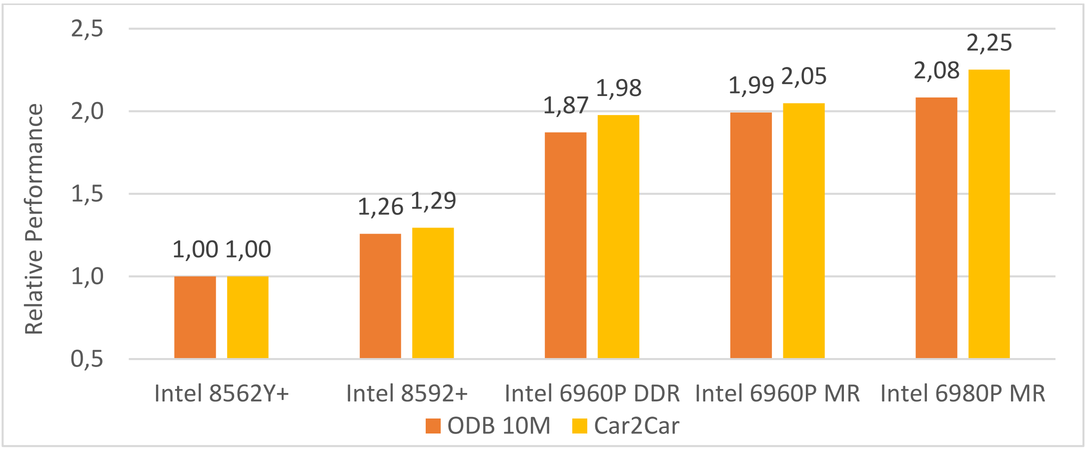


Fig.7: Intel Xeon6 (Granite Rapids) generation-to-generation 2 Socket LS-DYNA R12.1.0 normalized performance. Reproduced from [3].

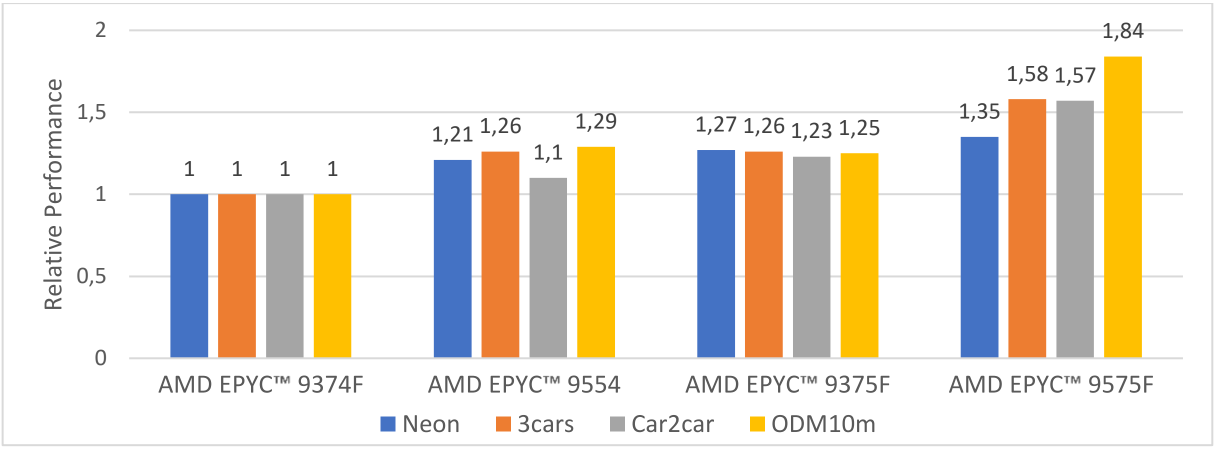
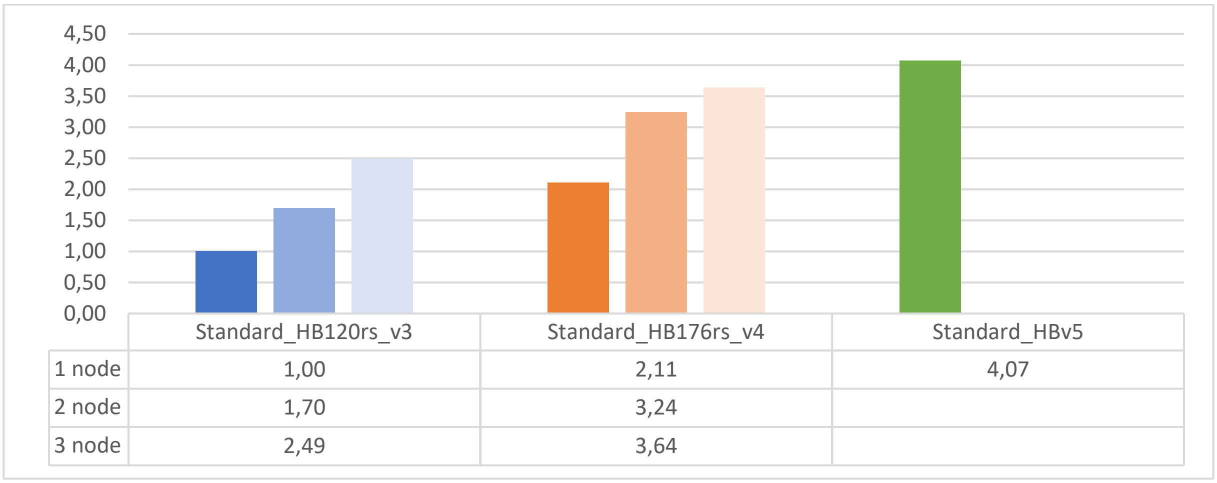


Fig.8: AMD EPYC Zen5 (Turin) generation-to-generation 2 Socket LS-DYNA R15.0.2 normalized performance. Adapted from [4].

4.1 Cloud Computing

Running LS-DYNA in the cloud allows organizations to avoid frequent on-premises hardware refresh cycles. The Microsoft Azure HB-series instances are specifically optimized for HPC and CAE workloads like LS-DYNA. As shown in Figure 9, LS-DYNA single-node performance has doubled with each new HB generation. Notably, a single HBv5 node outperforms three HBv4 nodes connected via 400 Gb/s NDR InfiniBand, demonstrating the raw performance benefits of newer hardware generations.



	Standard_HB120rs_v3	Standard_HB176rs_v4	Standard_HBv5
1 node	1,00	2,11	4,07
2 node	1,70	3,24	
3 node	2,49	3,64	

Fig.9: Microsoft Azure HBv5 generation-to-generation LS-DYNA normalized performance

4.2 ARM64

LS-DYNA also supports ARM64 architectures, which offer attractive thermal efficiency and energy savings—a key advantage for cloud and on-prem workloads where reduced power translates to lower operational costs. Importantly, this efficiency does not come at the expense of performance. As shown in Figure 10, processors like AWS Graviton4 and NVIDIA Grace-Grace deliver LS-DYNA performance comparable to prior-generation x86 CPUs [5] [6].

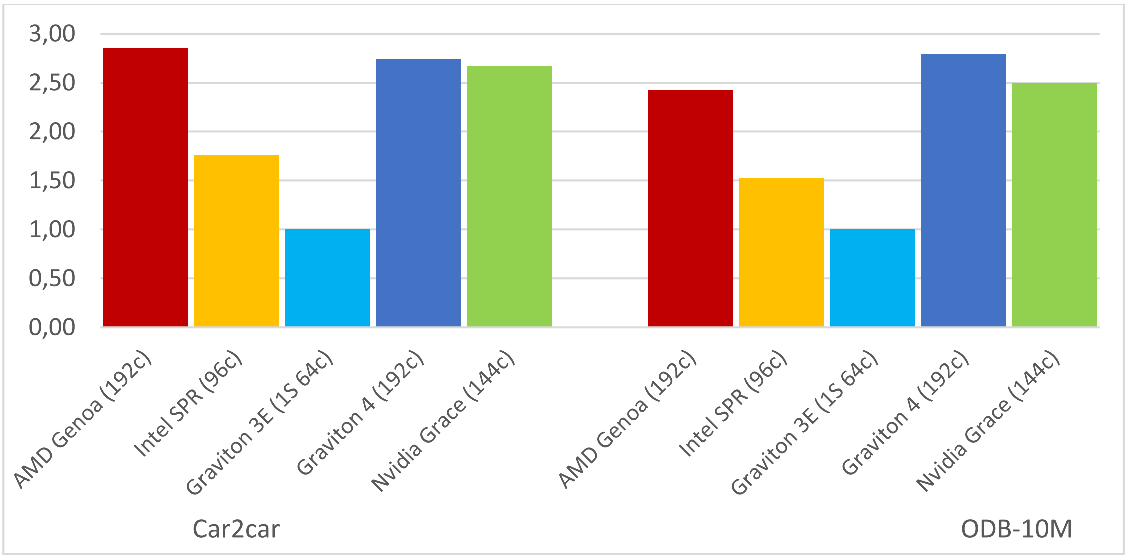


Fig.10: Prior generation x86 and ARM64 2 socket LS-DYNA R13.1.1 normalized performance. Adapted from [5].

ARM's efficient core design enables high core counts without excessive heat, reducing thermal-induced performance degradation [7]. Combined with strong Single Instruction Multiple Data (SIMD) support and high memory bandwidth, this makes ARM well-suited for scalable HPC workloads.

Nvidia Grace (publicly available in 2023) and AWS Graviton4 (mid-2024) continue to demonstrate excellent scaling relative to the latest Intel and AMD CPUs, as shown in Figure 11. System specifications for the performance benchmarks are available in Table 1. A dual-socket AMD Turin configuration consumes about 1000W for the CPUs alone, and roughly 1250W when the memory subsystem is included [8]. In contrast, a Grace-Grace system operates at approximately 500W, offering a substantial efficiency advantage [9].

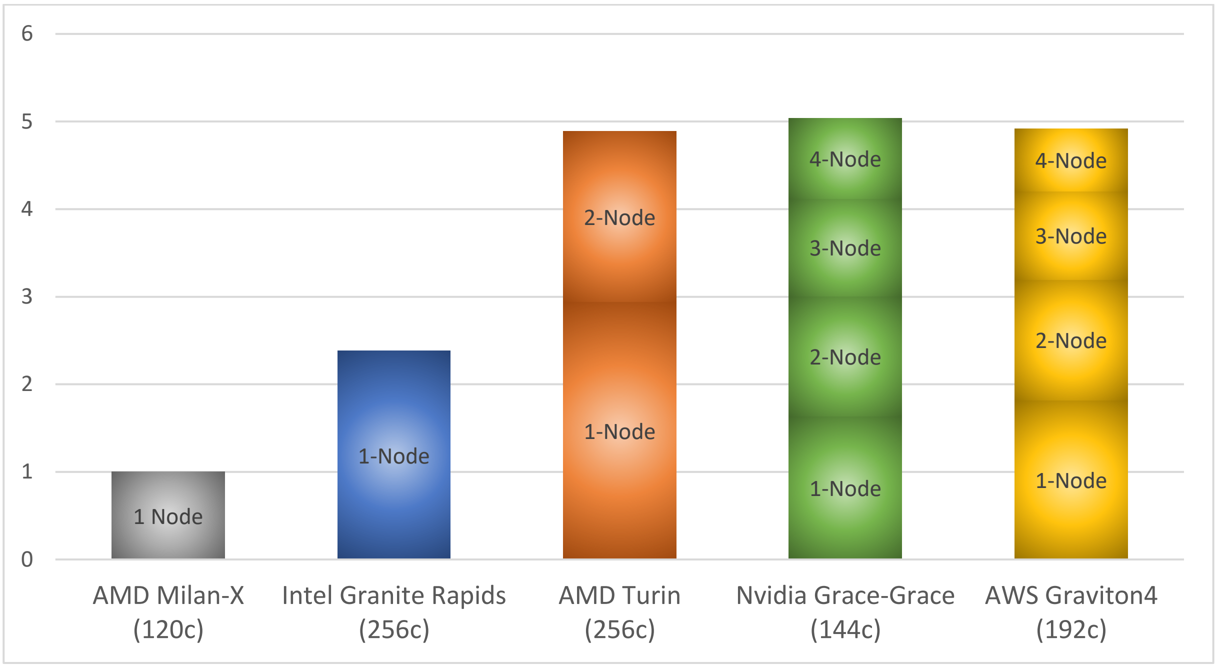


Fig.11: Latest generation x86 and ARM64 LS-DYNA Hybrid R16.1.0 normalized performance.

CPU	SKU/Instance	Cores per Node	Interconnect + rate
AMD Milan-X	EPYC 737VX	120c	
Intel Granite Rapids	Xeon 6980P	256c	
AMD Turin	EPYC 9755	256c	InfiniBand 2xNDR (200Gb/s)
Nvidia Grace-Grace	Grace CPU Superchip	144c	InfiniBand 4xNDR (400Gb/s)
AWS Graviton4	M8g.48xlarge	192c	EFA (50Gb/s)

Table 1: Latest Generation x86 and ARM64 System Specifications

5 Summary

Maximizing LS-DYNA MPP performance requires thoughtful hardware selection, efficient execution strategy, and balanced workload distribution. Selecting the optimal LS-DYNA binary for the target processor and model is a key foundation for achieving high performance. Undersubscription and hybrid configurations reduce communication overhead and improve memory and cache efficiency for increased speedup. Load balance is critical for high performance simulations. LS-DYNA's load and contact profiles provide a clear method of evaluating the load distribution. Imbalances from the inefficient distribution of contacts and expensive elements can be corrected with decomposition keywords or with the paritition file for finer control. On x86 platforms, recent hardware advances seen in Intel Xeon6, AMD EPYC Zen5, and Microsoft Azure HBv5 demonstrate impressive generation-to-generation performance gains in LS-DYNA. Meanwhile, ARM64 platforms like AWS Graviton4 and NVIDIA Grace offer energy-efficiency without compromise on performance. Together, these strategies enable high scalability, efficient resource use, sustainability, and faster results in LS-DYNA simulations.

6 Acknowledgements

The authors thank Dnyanesh Digraskar (Amazon) and Julien Santini (Ansys part of Synopsys) for their guidance.

7 Literature

References should be given in the last paragraph of your manuscript. Please use following scheme:

[1] Ansys part of Synopsys: “Ansys LS-DYNA User’s Manual”, 2025
[2] Santini, J: "Timing and Domain Decomposition", 2020, slides 11 and 23

[3] Slagter, W: "Ansys Collaborates with Lenovo to Accelerate Customer Innovation with Intel Xeon 6 Processors", 2024, slide 9

[4] Fsas Technologies: "Ansys LS-DYNA Nonlinear Dynamic Structural Analysis Solution on PRIMERGY", 2024

[5] Lecomber, D: "Ansys Fluent® and Ansys LS-DYNA demonstrates leading performance on Arm", 2024

[6] AWS Blog: “Now available: Graviton4-powered memory-optimized Amazon EC2 X8g instances, 2024

[7] Day, E: "Exploring the Potential of ARM Processors: Evaluating LS-DYNA Performance for Cloud-Based High-Performance Computing", 2023

[8] AMD: "AMD EPYC 9755", AMD product page, 2025

[9] Nvidia: "Nvidia Grace CPU Superchip Datasheet", 2024