Malleable OpenFOAM

Communication-Efficient Dynamic Reconfiguration for CFD

Arun Sharma[1], Javier Sánchez (advisor)[1] Sergio Iserte (advisor)[2]

[1]Universitat Autònoma de Barcelona, Spain [2]Barcelona Supercomputing Center, Spain

The Problem

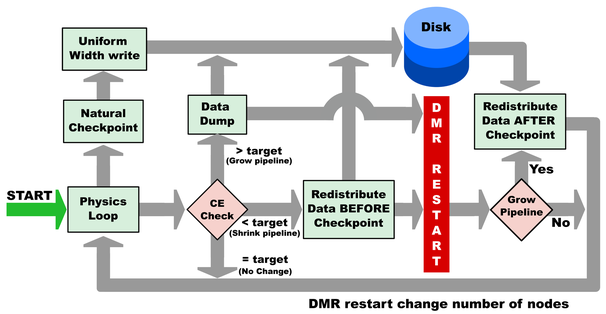
Rigid allocations waste compute. A simulation runs for days or weeks on a core count fixed at submission, whatever its demands do in between. Cores sit idle, and capacity that frees up elsewhere in the machine cannot be used.

Our Proposal

We present a **malleable implementation of OpenFOAM**, the world's most used open-source computational fluid dynamics solver: able to change its node count during execution, with no user intervention.

The decision is driven by communication efficiency (CE), the fraction of time the solver spends computing rather than waiting in MPI communication. When CE falls the job is too wide and shrinks; when CE has headroom it grows.

The Pipeline: Checkpoint-Restart



DMR restart change number of nodes

dynamic load balancing

DMR dynamic management of resources

DLB's TALP tool measures CE live. **DMR** reallocates resources and reshapes MPI processes. This work redistributes OpenFOAM's data.

Acknowledgements

We thank the Accelcom@BSC Research Group and OpenFOAM/DMR/DLB communities for their support.

References

[1] Weller et al. A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in Physics 12(6), 1998.

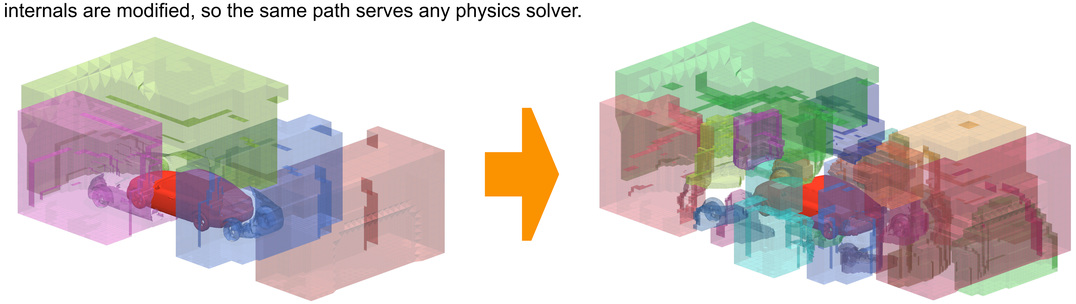
[2] Sandås et al. Seamless Execution of Malleable Applications in Controlled and Production HPC Environments. arXiv:2606.13266, 2026.

[3] López et al. TALP: A Lightweight Tool to Unveil Parallel Efficiency of Large-scale Executions. PERMAVOST '21.

[4] Heft et al. Introduction of a New Realistic Generic Car Model for Aerodynamic Investigations. SAE 2012-01-0168

Contribution

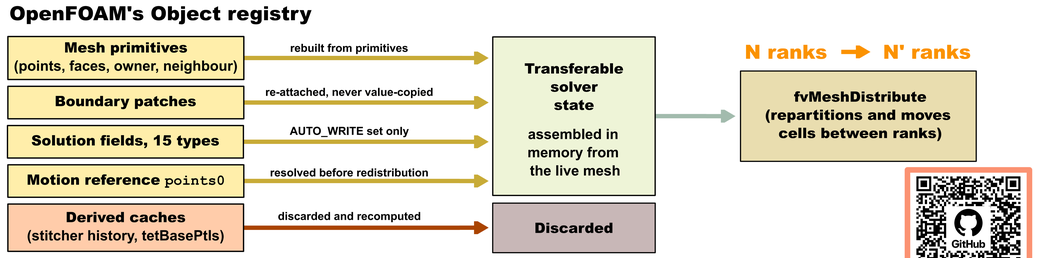
We establish what constitutes transferable solver state in OpenFOAM, and move it between rank sets in parallel. No solver internals are modified, so the same path serves any physics solver.



Each colour is one rank's subdomain: 4 ranks left, 25 right. When the rank count changes, PT-Scotch computes a new partition that minimises the shared boundary, and every cell moves from its old rank to its new one, in parallel. More ranks means smaller subdomains and more boundary to exchange, which is why CE falls.

Approach

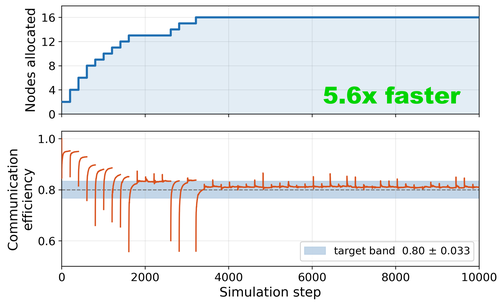
What a reconfiguration must carry, and what it must not.



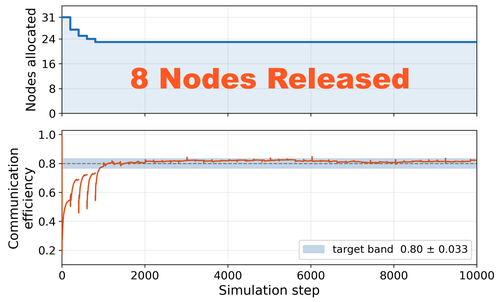
AUTO_WRITE is the invariant that defines solver state; the clone is built in memory, not re-read. Stale caches deadlock the next collective call.

GitHub Code Repository

Results



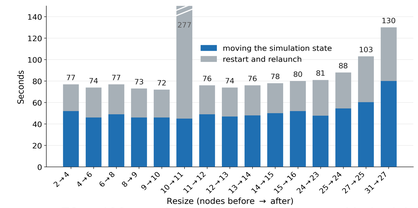
Too small a start: nodes added until efficiency reaches target. Settles at 16 after 11 resizes.



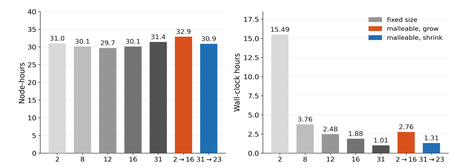
Too large a start: 8 nodes released, same target, settles at 23.

Experimental Setup

drivAerFastback, 92M cells, incompressible turbulent flow, foamRun on MareNostrum 5 supercomputer at 112 cores per node. Malleable runs start at 31 and at 2 nodes, static controls at 2, 8, 12, 16 and 31.
CE target 0.80, band ±0.033. The target is set in the launch script and is tunable per case.

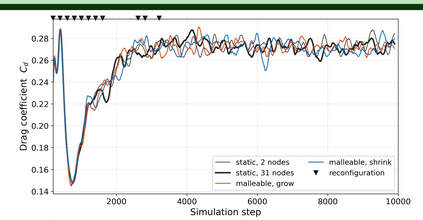


Every resize costs 72 to 130 s and the cost does not grow with the job: moving the state stays near 50 s throughout. One resize stalled in restart (axis broken). Its state move was the fastest of all.



Similar cost at every size, very different time. From 2 nodes the malleable run is 5.6x faster for 6% more node-hours; from 31 it returns 8 nodes at near zero cost.

The number of reconfigurations is set by convergence, not by run length: the same handful serves 10k steps or 500k. Here we check resizes far more often than production would, so the gain is much larger in practice.



Drag coefficient. The malleable runs follow the same trajectory as the static ones, with no discontinuity at any reconfiguration.

Conclusions

OpenFOAM's 1.5M LoC now make better use of the hardware they are given. The solver changes its resource allocation while running, redistributing its full state in parallel between ranks, with the computed results unchanged and no solver internals modified.The practical result is automatic resource selection: users pay for an efficient run instead of wasting nodes or waiting longer than they need to.