Malleable OpenFOAM: Communication-Efficient Dynamic Reconfiguration for CFD

Arun Sharma
Universitat Autònoma de Barcelona
Barcelona, Spain
arunsharma14@acm.org

Javier Sánchez (advisor)
Universitat Autònoma de Barcelona
Barcelona, Spain
javier.sanchez.pujadas@uab.cat

Sergio Iserte (advisor)
Barcelona Supercomputing Center
Barcelona, Spain
sergio.iserte@bsc.es

1 Introduction and Motivation

Computational fluid dynamics consumes a large share of HPC cycles, and OpenFOAM [4] is the world's most widely used open-source CFD solver. Like almost every production MPI application it is rigid: the core count is fixed at submission and held until the job exits, however the simulation's demands change in between. Adaptive mesh refinement changes the cell count as the run proceeds, and multi-region and transient cases shift work between phases.

At exascale that rigidity becomes a limiting factor. A job sized for its heaviest phase idles cores through the rest. One sized conservatively cannot absorb capacity that frees up elsewhere. Malleability, changing a running job's allocation during execution, removes the constraint, but requires the application to rebuild its entire distributed state at the new size while the simulation is running.

We present the first malleable implementation of OpenFOAM. The solver monitors its own communication efficiency, asks for more nodes or releases those it is not using when that efficiency leaves a target band, and rebuilds its distributed state across the new rank set, with no user intervention.

2 Background and Related Work

OpenFOAM decomposes its mesh once, before the run, and each MPI rank owns one partition for the lifetime of the job. Parallel state lives on disk as one directory per rank, holding that rank's cells, fields and boundary data, and the layout is bound to the rank count at launch. A job therefore cannot resume at a different width: the directories no longer correspond to anything.

Malleability here follows a checkpoint-restart model. Dynamic Management of Resources (DMR) [3] negotiates the resource change with the batch system, stops the application and relaunches it at the new node count. It decides when to resize and by how much. What it cannot do is rebuild the application's own data. Communication efficiency (CE), the fraction of wall time spent computing rather than blocked in MPI, drives that decision: spread a fixed mesh over more ranks and each holds fewer cells while exchanging as much with its neighbours, so CE falls. TALP, part of the DLB library [2], measures CE live and feeds DMR's policy. Reconfiguring in memory instead would mean rewriting OpenFOAM's parallel layer, an invasive change that DMR and our approach both avoid.

Malleability has been demonstrated on mini-applications and purpose-built codes. To our knowledge no malleable implementation of OpenFOAM has been documented. The gap is one of difficulty. OpenFOAM is a general-purpose framework in the order of 1.5M LoC, its runtime object registry holds far more than the solution, and working out what must move at a reconfiguration is itself the hard problem. That difficulty is why the result generalises: which runtime state defines an application is what any large production code must answer to become malleable.

3 Approach and Uniqueness

OpenFOAM already ships the machinery to move cells between ranks. `fvMeshDistribute` takes a mesh and a target partition and gives every rank its new share. It is meant to be used offline, between runs. We drive it in flight, on the running solver, and that raises two problems.

The first is knowing what to hand it. OpenFOAM keeps every runtime object in a registry: solution fields, but also cached intermediates, geometric derivations and per-rank scratch state. We establish that OpenFOAM's own `AUTO_WRITE` flag, marking what the solver would write in a checkpoint, discriminates exactly the set a reconfiguration must carry. It is not an output convenience. It is the definition of solver state. Everything else is rank-local and must not travel.

The second is a failure that appears only at scale. Some cached objects are optional: a rank reads one if its file is present and recomputes it otherwise. After a reconfiguration only the ranks that held data write a stale copy, so at restart those ranks skip a collective call that the rest enter. The group falls out of step and the run deadlocks. A uniform rank set never diverges, which is why this stays hidden until the set is large enough to be inhomogeneous. Such caches have to be discarded and recomputed rather than carried across.

The mechanism sits above the solver, on the mesh and field registry every OpenFOAM solver shares, so it is independent of the physics and the case. A reconfiguration is taken between iterations, where OpenFOAM would already write a checkpoint, so the state that moves is never half updated, and it is driven from the run loop rather than from inside the solver. Policy parameters, CE target and sensitivity, the reconfiguration inhibitor, and minimum and maximum node counts, are set in the launch script rather than compiled in. Every reconfiguration also repartitions across the new rank set, so whatever imbalance the run has accumulated is corrected at the same moment. Load balancing comes free with the resize.

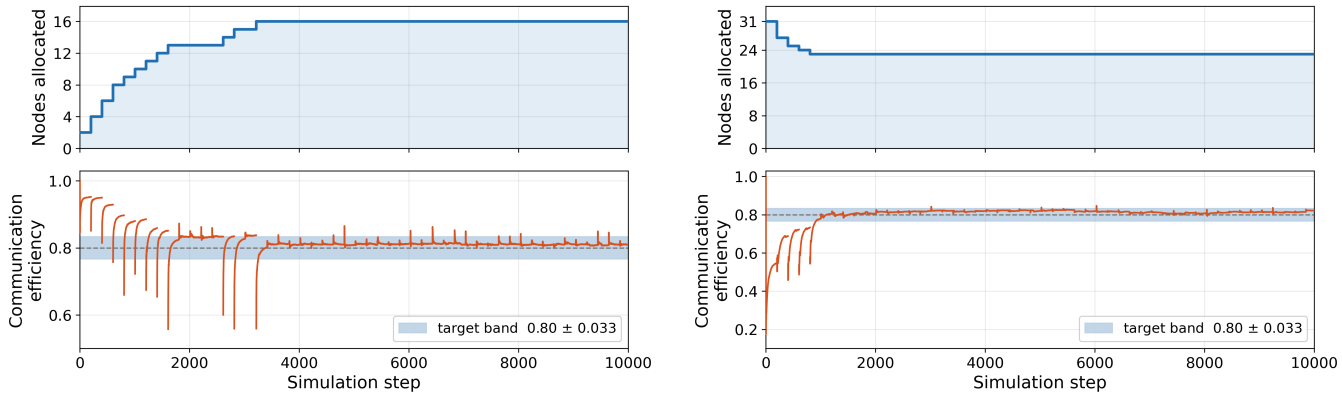


Figure 1: The controller reaches the target band from both directions. Left, from 2 nodes it grows to 16 in eleven reconfigurations. Right, from 31 nodes it shrinks to 23 in four, returning 8 nodes to the scheduler while the simulation continues.

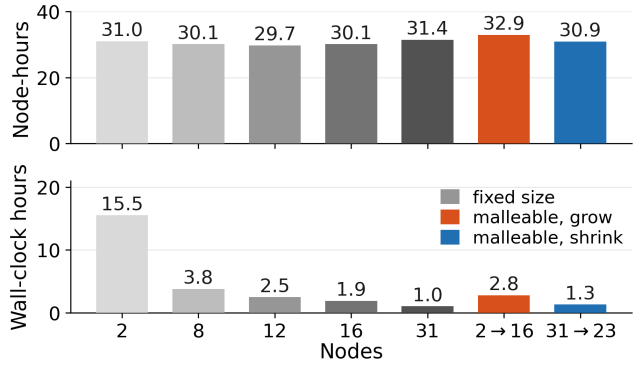


Figure 2: Seven runs, five at a size fixed in advance. Cost is nearly constant at every size. Time is not.

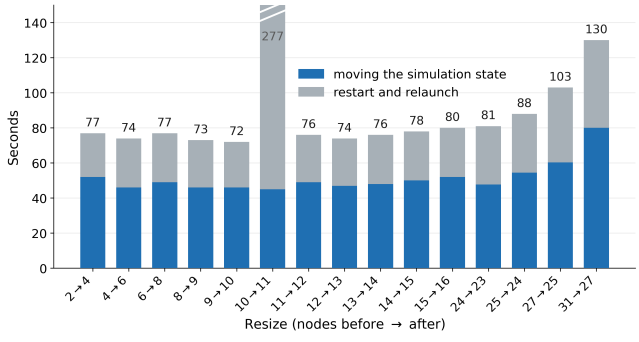


Figure 3: Where a reconfiguration's time goes, per event. Moving solver state is the larger half. The remainder is restart and relaunch, which a mechanism that avoided stopping the job would remove.

4 Results and Contributions

The same build reconfigures in both directions, repeatedly, on much smaller cases: a lid-driven cavity, a moving-mesh case, and a six-region engine with non-conformal couplings and mesh-to-mesh swaps. None of those scales to a supercomputer without expertise the case itself demands, which is why the campaign runs on the drivAerFastback automotive benchmark [1], at 92M cells, incompressible turbulent flow, on MareNostrum 5 supercomputer at 112 cores per node, CE target 0.80 (band ±0.033). Each of seven runs advances the same 10,000 SIMPLE iterations, and the allocation reaches 31 nodes, 3472 cores.

Two runs probe the controller from opposite directions (Figure 1). From an under-provisioned start at 2 nodes it reaches 16 in eleven reconfigurations. From an over-provisioned start at 31 it converges to 23 in four, returning 8 nodes to the scheduler while the simulation continues. Both arrive in the target band, at 0.810 from below and 0.813 from above.

Five fixed-size runs at 2, 8, 12, 16 and 31 nodes give the comparison (Figure 2). Cost is nearly constant across all seven runs, 29.7 to 32.9 node-hours, because this case scales near-linearly to 31 nodes, while wall time varies by a factor of fifteen. The malleable runs sit in the same cost band as every fixed size while choosing that size themselves, so what a user commits no longer depends on guessing it correctly in advance. Finding the size is paid in time rather than in resources: the growing run takes 2.76 h against 1.88 h for a run started at the 16 nodes it converged to.

Fourteen of the fifteen reconfigurations cost 72 to 130 s each, the fifteenth having waited on the resource manager. Moving solver state is 54% of the total and the rest is restart and relaunch (Figure 3). The cost does not grow with the job, and their number follows how fast the controller converges rather than run length, so the same handful serves a production run.

Force coefficients follow the same trajectory as the fixed-size runs, with no discontinuity at any reconfiguration.

Taking everything into account, this malleable implementation of OpenFOAM automatically selects the resources so users only pay for the efficient run and do not waste resources or wait more time for their results.

Acknowledgments

We thank the Accelcom@BSC research group for the opportunity and for access to MareNostrum 5, and the OpenFOAM, DMR and DLB communities for their support. AI assistance (Claude, Anthropic) was used for text editing.

References

[1] Angelina I. Heft, Thomas Indinger, and Nikolaus A. Adams. 2012. *Introduction of a New Realistic Generic Car Model for Aerodynamic Investigations*. SAE Technical Paper 2012-01-0168. SAE International. doi:10.4271/2012-01-0168

[2] Victor Lopez, Guillem Ramirez Miranda, and Marta Garcia-Gasulla. 2021. TALP: A Lightweight Tool to Unveil Parallel Efficiency of Large-scale Executions. In *Proceedings of the 2021 on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn STrategy (PERMAVOST '21)*. Association for Computing Machinery. doi:10.1145/3452412.3462753

[3] Petter Sandås, Sergio Iserte, Guillaume Houzeaux, and Antonio J. Peña. 2026. Seamless Execution of Malleable Applications in Controlled and Production HPC Environments. arXiv:2606.13266 [cs.DC]

[4] H. G. Weller, Gavin Tabor, Hrvoje Jasak, and Christer Fureby. 1998. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics* 12, 6 (1998), 620–631. doi:10.1063/1.168744