Topology-Aware Communication Optimization for CFD Simulations in OpenFOAM

Jonas Sieberer[✉1], Clemens Gößnitzer[2], Andreas Schröder[1], and Robert Elsässer[1]

[1] Universität Salzburg, 5020 Salzburg, Austria
{firstname.lastname}@plus.ac.at
[2] Large Engines Competence Center LEC, 8010 Graz, Austria
clemens.goessnitzer@lec.tugraz.at

Abstract. Parallel computational fluid dynamics (CFD) simulations are commonly executed on modern high-performance computing systems with hardware topologies, where communication costs vary significantly across memory hierarchies and network links. OpenFOAM, a widely used open-source CFD framework, provides a scalable MPI-based parallel infrastructure. However, its default process placement and communication strategies are largely topology-agnostic, which can lead to suboptimal performance on NUMA-based cluster architectures. We present a topology-aware communication optimization for OpenFOAM that targets both local and global communication in iterative linear solvers. Local communication is optimized by reassigning MPI ranks to physical cores based on the communication structure induced by the domain decomposition and the underlying hardware topology. In addition, a custom global reduction schedule is introduced that aligns collective communication with the hierarchical organization of compute nodes, NUMA domains, and NUMA nodes.
Our experimental results show a substantial redistribution of communication toward faster hardware paths, leading to better runtime with the largest improvements observed for communication-intensive solver configurations. These findings demonstrate that topology-aware communication strategies can significantly enhance the performance of OpenFOAM on modern HPC systems.

Keywords: Computational fluid dynamics · Finite volume method · OpenFOAM · Non-uniform memory access hierarchy.

1 Introduction

Computational fluid dynamics (CFD) simulations [18] are an essential tool in modern scientific and engineering research, with applications ranging from aerodynamics and weather prediction to combustion and energy systems. As simulation models grow in size and complexity, CFD applications increasingly rely on high-performance computing (HPC) systems to achieve acceptable runtimes [13]. The open-source CFD framework OpenFOAM (Open Source Field Operation and Manipulation) [21] is widely used in both academia and industry[3]. OpenFOAM employs the finite-volume method (FVM) to discretise the underlying partial differential equation, which results in large (and

[3] We used OpenFOAM-12 by "The OpenFOAM Foundation", https://openfoam.org/release/12/

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

sparse) systems of linear equations. These systems are typically solved using iterative methods on large HPC machines. Parallel execution in OpenFOAM is based on domain decomposition and MPI-based inter-process communication [19]. Although this approach makes simulations highly scalable in theory, the performance of the corresponding solvers is strongly influenced by the amount of local and global communication arising during the computation.

Modern HPC systems exhibit hierarchical hardware topologies, including compute nodes, NUMA domains, and cache-coherent core groups, where communication costs vary significantly across levels. OpenFOAM assigns MPI ranks by default sequentially without taking into account the underlying hardware structure. Thus, communication patterns are not aligned with the hardware topology, causing a substantial fraction of communication to be mapped onto slower hardware paths instead of low-latency ones.

We present a topology-aware communication optimization for OpenFOAM that targets both, local and global communication in iterative linear solvers. We improve local communication by assigning MPI ranks to physical cores based on the communication structure induced by the domain decomposition and the hardware topology. We also introduce a hardware-aware global reduction schedule to better align communication with the hierarchical organization of the system. The proposed approach is implemented as a preprocessing step and does not modify OpenFOAM's numerical algorithms. Our experimental results show that these topology-aware strategies lead to (partly significant) runtime improvements on modern HPC systems. While implemented for OpenFOAM, the methodology can possibly be extended to other applications and architectures.

Our main goal is to reduce communication overhead in OpenFOAM, which is a key bottleneck for its scalability [2]. Improving the efficiency of various other parallel applications by exploiting hardware topology has been widely studied. In particular, local [7] and global communication [8] were optimized in MPI applications through rank-to-core binding and rank reordering. In [16] a related approach was used to optimize data communication in ccNUMA systems for a parallel social network application. Further work on reducing communication in NUMA architectures includes thread and memory migration strategies [6], and the reduction of cache coherence traffic [1]. How OpenFOAM currently deals with the hardware architecture is described in Section 4.

The paper is organized as follows. Section 2 outlines FVM and its parallel execution in OpenFOAM. Section 3 describes the relevant hardware characteristics of modern HPC systems. Section 4 presents our topology-aware communication optimizations. Experimental results are given in Section 5, followed by the conclusions in Section 6.

2 The Finite-Volume-Method in OpenFOAM

This chapter presents an overview of the implementation of the finite-volume method in OpenFOAM. First, the discretisation of the computational domain and the mesh structures used are described. Then the discretisation of PDEs and their transformation into a system of linear equations is briefly explained for the sequential workflow. Subsequently, the parallel workflow in OpenFOAM is discussed, with a particular focus on the communication between processes. Special attention is given to the local and global communication steps required during the execution of iterative linear solvers.

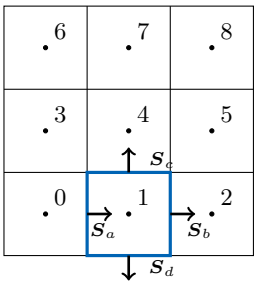
ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

2.1 Mesh Structure and Finite-Volume Discretisation

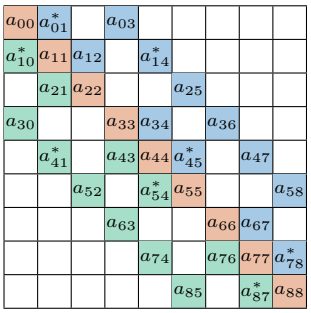
In OpenFOAM, the three-dimensional computational domain $\Omega \subset \mathbb{R}^3$ is discretized into a finite number n of non-overlapping polyhedral control volumes V_i, also called cells, that together form the discretized domain Ω_D. A control volume is bounded by a set of flat faces; the collection of cells and faces defines the computational mesh [9].

Faces shared by two control volumes are referred to as internal faces, while faces on the boundary of Ω_D are boundary faces. By convention, the cell with the smaller index is designated as the owner of a face, while the adjacent cell is referred to as the neighbor. Boundary faces are grouped into boundary patches, allowing different boundary conditions to be applied to different parts of the domain boundary. The faces belonging to these boundary patches are conventionally placed at the end of the face index range. Each face f_r is associated with a face area vector $\boldsymbol{S}_r$, which is normal to the face and points outward from the owner cell. Each control volume V_i is associated with a representative calculation point $\boldsymbol{x}_i$, typically chosen as the cell centroid [9].

OpenFOAM operates internally on three-dimensional meshes. Two-dimensional problems are represented by using a single cell in the third direction by assigning empty boundary conditions to the corresponding faces. Figure 1a illustrates a simple two-dimensional 3×3 mesh together with the associated face area vectors.



(a) 3×3 mesh over $[0, 1]^2$ with cell indices.



(b) Finite volume Matrix A for the left mesh.

Fig. 1: Structured 3×3 example mesh with marked faces $f_a, \ldots, f_d$ of cell V_1 and corresponding sparse finite-volume matrix. (Some matrix elements are marked with $*$, they represent the coefficients lying on the processor patches, see explanation in Fig. 2b)

Let the partial differential equation be defined over Ω_D. In the finite-volume method, the equation is integrated over each control volume. Applying the divergence theorem [15] one can transform volume integrals into surface integrals over the faces of the cell. The resulting face fluxes are approximated using discretisation schemes provided by OpenFOAM [4]. Internal faces give rise to coupling terms between neighbouring control volumes, while boundary faces contribute according to the prescribed boundary conditions.

Each control volume therefore leads to a linear equation containing contributions from neighbouring cells as well as source and boundary terms collected on the right-hand side. Assembling the contributions from all faces yields a system of linear equa-

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

tions of the form $A\phi = \boldsymbol{b}$, where $A \in \mathbb{R}^{n\times n}$ is a sparse matrix whose non-zero off-diagonal entries correspond to internal faces and encode the mesh connectivity.

In many practical applications, the underlying PDEs are nonlinear. In OpenFOAM, such cases are treated using iterative linearization techniques, where nonlinear terms are evaluated using previously computed solution values. This results in a sequence of linear systems of the above form, which are solved repeatedly until convergence.

Figure 1b shows the finite-volume matrix corresponding to the mesh in Figure 1a. Diagonal entries collect all self-coupling and boundary contributions, while the right-hand side vector $\boldsymbol{b}$ contains source and boundary-condition terms.

Solving this system yields the values of the unknown field at the calculation points, which are assumed to be representative for the corresponding control volumes. OpenFOAM provides a range of iterative solvers for this task, and solving these linear systems constitutes the dominant computational cost in most simulations. Parallel execution reduces the size of the local systems but introduces communication between processes, as discussed in the following sections.

2.2 Domain Decomposition and Parallel Execution

When OpenFOAM is executed in parallel, the mesh is initially generated in the same way as for a sequential simulation. For a given number of processes $n_p \geq 2$, the parallel computation begins with a domain decomposition step, in which the global mesh is partitioned into n_p submeshes. The objectives of this decomposition are to obtain a balanced distribution of cells among the processes and a minimization of inter-subdomain interfaces, as these directly determine the amount of inter-process communication.

OpenFOAM provides several decomposition methods, including simple geometric partitioning as well as graph-based approaches using third-party libraries such as SCOTCH [12] and METIS [5]. During decomposition, each control volume V_i is uniquely assigned to exactly one MPI process P_k, with $k \in \{0, \ldots, n_p - 1\}$. Each process therefore obtains a subset of cells that form its local submesh.

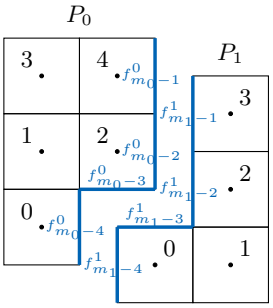
Figure 2a illustrates a simple decomposition of a 3×3 mesh into two subdomains. After decomposition, each submesh is constructed locally on its assigned process. As in the sequential case, the submesh consists of points, faces, and cells, which are assigned new local indices. Addressing structures are created to map these local indices to the corresponding global indices of the original mesh. The original boundary patches are split accordingly and distributed among the submeshes.

For each pair of adjacent subdomains, a new boundary patch is introduced, referred to as a *processor patch*. These patches represent the set of faces between neighbouring MPI processes, which are appended to the list of boundary faces of each submesh. Each processor face corresponds to an internal face of the global mesh and therefore appears on exactly two processor patches, one on each adjacent process. The processor boundary condition defines the communication interface and ensures that the required field values are exchanged between neighbouring processes during the computation.

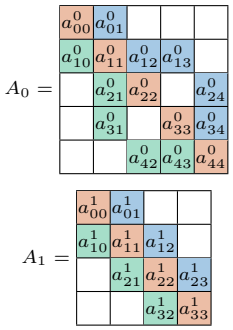
Once the submeshes have been constructed, all processes execute the same solver code in a distributed-memory parallel fashion. Apart from additional coordination and input/output tasks performed by MPI rank 0, all ranks participate symmetrically in the numerical computation and communication.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

On each process P_k, a local linear system $A_k\phi_k = \boldsymbol{b}_k$ is assembled for the corresponding submesh. The coefficients associated with internal faces and physical boundary faces are computed in the same manner as in the sequential case. For processor faces, the discretisation coefficients are determined analogously to internal faces. However, the required neighbour values are not locally available.



(a) Example decomposition into two subdomains (local cell indices).



(b) Local matrices A_0 (top) and A_1 (bottom).

Fig. 2: Simple horizontal decomposition of 3×3 mesh. The faces $f^0_{m_0-4}, \ldots, f^0_{m_0-1}$ form the new processor boundary patch of P_0 and correspond to the global coefficients $a^*_{01}, a^*_{41}, a^*_{45}, a^*_{78}$ in Figure 1b (where they are marked with $*$).

All coefficients associated with internal faces are contained in the local matrices, whereas those corresponding to processor faces are stored separately. Since processor faces do not introduce additional source terms, the local right-hand side vectors $\boldsymbol{b}_k$ and the initial solution values are direct subvectors of their global counterparts. Hence, the collection of local systems remains algebraically consistent with the global system. Figure 2b illustrates the resulting local matrix structures for our example.

To solve the global system iteratively, missing neighbor values associated with processor faces must be exchanged between processes. This neighbor communication is required, for example, in sparse matrix–vector products involving adjacent subdomains.

In addition, iterative solvers require global operations such as scalar products and norm evaluations, which involve collective communication across all processes. Although independent of processor-face data, these reductions require global synchronization and thus contribute to the overall communication cost. If a large number of processes are used, neighbor exchange and global reductions significantly influence the runtime. The structure of these communication patterns is discussed in the next section.

2.3 Local and Global Communication

Iterative linear solvers in OpenFOAM [4], including Krylov subspace methods such as the preconditioned conjugate gradient method (PCG), and multigrid solvers such as the geometric algebraic multigrid method (GAMG), repeatedly execute distributed linear algebra operations. Each iteration involves matrix–vector products, vector norms, and scalar products. In a parallel setting, these operations induce *local communication* between neighboring subdomains and *global communication* for collective reductions.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

Local Communication. Local communication refers to data exchange between adjacent subdomains that share a processor boundary patch. Consider the local operation $\boldsymbol{u}_k = A_k \boldsymbol{v}_k$, where $A_k \in \mathbb{R}^{n_k \times n_k}$ denotes the local finite-volume matrix on process P_k and $\boldsymbol{v}_k \in \mathbb{R}^{n_k}$ is the local vector, in which each entry represents the value of a cell in the corresponding submesh. While contributions associated with internal faces can be computed locally, those corresponding to processor faces require values of $\boldsymbol{v}$ owned by neighbouring processes. These values are referred to as halo values [20].

Since the vector $\boldsymbol{v}$ changes between successive executions of matrix-vector products, the required halo data must be exchanged after every iteration. For each processor patch, the relevant vector entries are packed, sent to the neighboring process, and the corresponding remote values are received in return. Thus, the communication volume depends heavily on the processor patches and the quality of the domain decomposition.

In OpenFOAM, this exchange is implemented via a two-stage update. First, communication is initiated for all processor patches, issuing send and receive operations through OpenFOAM's communication layer, which internally wraps MPI primitives. This allows local computations that do not depend on halo data to proceed during the send/receive process. Second, the solver waits for completion of the communication and incorporates the received halo values, completing the local matrix–vector product.

Global Communication. Global communication is required for operations that combine contributions from all processes, such as scalar products, vector norms, and other collective reductions. Each process computes a local partial result, which must be aggregated into a global value and redistributed to all ranks.

In OpenFOAM, reductions are handled within the Pstream communication layer. Depending on the configuration and implementation settings, OpenFOAM either delegates the operation to MPI collective routines or executes its own explicit reduction schedule based on point-to-point communication. When default OpenFOAM schedules are used, the corresponding deterministic reduction patterns are independent of the MPI implementation and operate solely on MPI rank indices. If the number of processes exceeds a certain predefined value, a tree-based reduction schedule is used, which is organized as a logical binary tree over the subdomain indices. At each level, processors are paired such that one process receives a partial result from its partner, performs a local reduction, and forwards the aggregated value to the next level. This continues until a root process accumulates the global result, which is then propagated back to all ranks.

As mentioned above, the schedule is constructed based on MPI rank indices and is independent of the hardware topology. Hence, communication steps may traverse NUMA boundaries or network links unnecessarily, although faster paths exist. This also motivates the topology-aware communication optimizations given in the next sections.

3 Hardware Topology Background

Modern high-performance computing (HPC) systems are typically organized as clusters of interconnected compute nodes [11]. Each node is a shared-memory system containing multiple processor sockets, main memory, and a network interface for inter-node communication. Although all cores within a node share a global address space, the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

physical organization of processors, caches, and memory results in a hierarchical topology with non-uniform communication and memory-access costs.

At the highest level, communication between compute nodes is performed via dedicated interconnects such as InfiniBand. Inter-node communication usually incurs the highest latency and lowest bandwidth and is therefore the most expensive communication path. In this work, only the node-local topology is considered.

Within a node, cores are organized into NUMA (non-uniform memory access) domains, typically corresponding to CPU sockets. Each NUMA domain is directly connected to a portion of main memory. Memory accesses within the same domain exhibit lower latency and higher bandwidth than accesses to remote memory attached to other sockets, which require traversal of inter-socket interconnects.

At a finer granularity, each NUMA domain consists of multiple NUMA nodes, corresponding to individual cores or small core groups, which share partially the same cache resources. Modern processors provide a hierarchical cache subsystem, typically including private L1 and L2 caches at each core and a shared last-level cache (LLC) [3], see Fig. 3. Data served from cache is substantially faster than accesses to main memory, making intra-NUMA locality the most efficient communication path within a node.

The performance of parallel applications therefore depends on how well computation, memory placement, and communication are aligned with the hardware hierarchy. In particular, NUMA-aware process placement aims to exploit fast local communication paths while minimizing costly remote and inter-node data transfers [17].

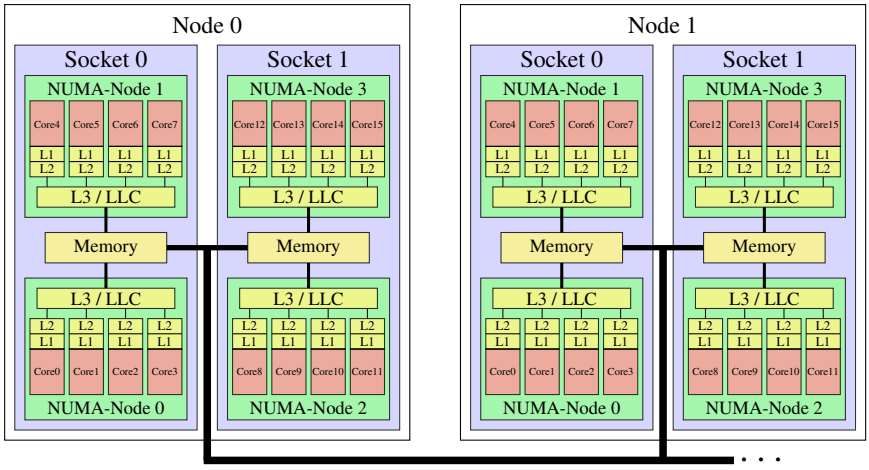


Fig. 3: Example hardware topology with two compute nodes, illustrating communication paths at different hierarchy levels. Thicker links indicate higher latency.

Depending on the relative placement of two communicating processes, four distinct types of point-to-point communication can be identified, from fastest to slowest:

- *intra-NUMA-node*, cores on same NUMA-node
- *inter-NUMA-node*, cores in same NUMA-domain, but on different NUMA-nodes
- *inter-NUMA-domain*, cores on same compute node, but on different NUMA-domains
- *inter-node*, cores on different compute nodes

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

These latency differences arise from the underlying cache hierarchy, memory controllers, and interconnects, as illustrated in Figure 3. This hierarchical structure directly motivates topology-aware communication optimizations.

3.1 Hardware Setup

The experiments were conducted on two HPC clusters with shared-memory compute nodes. The VSC-5 zen3_0512 system provides 128 cores per node across two sockets, organized into a total 16 NUMA nodes[4]. The LEC hpc2 cluster is a private in-house system with 48 cores per node distributed over four sockets, with each socket corresponding to one NUMA node. The relevant hardware and software characteristics are shown in Table 1. In VSC-5, the compute nodes are connected via several InfiniBand switches. However, we do not consider this additional network hierarchy in our paper.

To quantify the impact of different communication paths, additional point-to-point communication times were measured using the OpenFOAM communication framework. In these tests, a list of scalar values is exchanged bidirectionally between all participating processes, and the corresponding round-trip time is measured. Depending on the placement of the processes within the hardware topology, different communication times can be observed, reflecting the respective communication layers. The size of these lists corresponds, in the parallel workflow, to the number of faces associated with processor patches. Accurate knowledge of the hardware topology seems to be essential, as incorrect assumptions about the mapping between cores, NUMA nodes, and NUMA domains can significantly degrade potential performance gains.

Table 1: Hardware characteristics relevant for communication.

	VSC-5 zen3_0512	*LEC hpc2*
CPU	AMD EPYC 7713	Intel Xeon Gold 6126
Cores per node	128	48
Sockets per node	2	4
NUMA nodes per socket	8	1
Cores per NUMA node	8	12
Network	InfiniBand 200 Gbit/s	InfiniBand 100 Gbit/s
MPI	OpenMPI 4.1.4	OpenMPI 4.1.5

The measured mean communication times, summarized in Table 2, clearly reveal substantial latency differences between the communication path types. Although absolute times vary between nodes, especially on VSC-5 depending on their position, the relative cost ordering remains consistent, confirming the importance of prioritizing fast communication paths. However, for larger message sizes on VSC-5, inter-NUMA-domain communication can become slower than inter-node communication. This behavior depends on the specific hardware characteristics and on the processor-patch sizes occurring in a given simulation. Since both aspects vary across HPC systems and benchmark cases, we assume a fixed hierarchical ordering of communication paths in the

[4] See VSC-5: Vienna Scientific Cluster 5, https://docs.asc.ac.at/systems/vsc5.html

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

present work. Future work could incorporate runtime communication benchmarks to determine the effective hierarchy dynamically and adapt the optimization accordingly.

Table 2: Mean point-to-point communication time in µs.

	VSC-5				LEC hpc2		
Size	Intra-Nn	Inter-Nn	Inter-Nd	Inter-Node	Intra-Nn	Inter-Nn	Inter-Node
10	0.4	0.7	1.4	2.9	1.1	1.5	7.9
100	0.8	1.5	3.2	3.7	1.4	2.0	8.6
500	1.3	2.8	6.5	5.6	2.2	3.3	11.1
1000	3.1	5.0	11.4	8.0	3.6	5.8	11.8

4 Topology-Aware Communication Optimization

As discussed previously, the runtime of large-scale OpenFOAM simulations is dominated by iterative linear solvers, whose core operations induce substantial communication overhead. Since communication costs depend strongly on process placement within the hardware hierarchy, performance can be improved by aligning rank mapping and communication schedules with the underlying topology. In the following, we present a topology-aware optimization targeting both local and global communication.

Let P_k, $k \in 0, \ldots, n_p - 1$, denote the MPI process with rank k. By default, OpenFOAM employs a canonical, index-based rank assignment that is independent of both the mesh decomposition and the hardware topology.

For a simulation executed on N compute nodes with N_c cores each, resulting in $n_p = N \cdot N_c$ MPI processes, process P_k is placed on node $\lfloor k/N_c \rfloor$ and core $(k \bmod N_c)$. This simple bottom-up strategy does not account for NUMA nodes, NUMA domains, or inter-process communication patterns. Consequently, neighbouring subdomains with high data exchange may reside on distant hardware resources, causing avoidable inter-socket or inter-node communication.

4.1 Optimized Processor Rank-to-Core Allocation

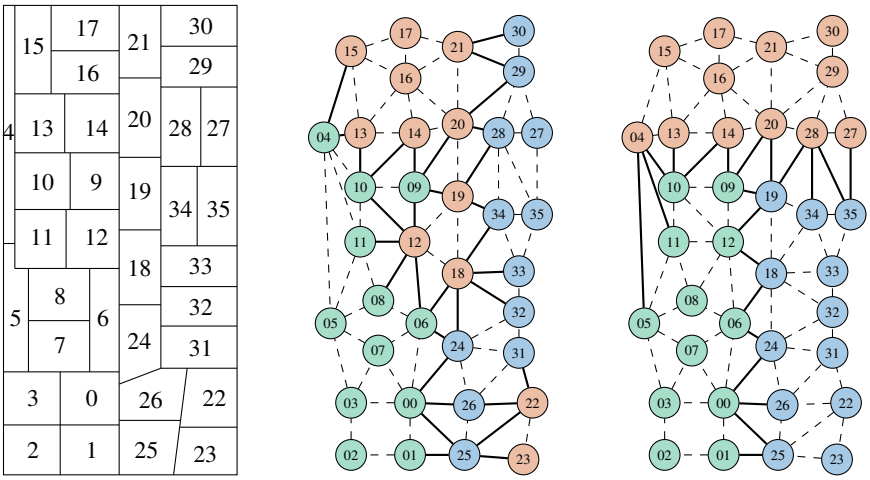
To overcome the limitations of the default OpenFOAM mapping, processes with high mutual communication are assigned to physically close cores, where proximity corresponds to faster communication paths in the hardware hierarchy. Since communication costs increase from intra-NUMA-node to inter-node transfers, the optimization minimizes communication hierarchically, beginning with the most expensive level.

Graph-based formulation The communication structure induced by the mesh decomposition is modeled as a weighted graph, where each MPI process is a vertex and edges represent neighbouring subdomains defined by processor patches. The edge weights are equal to the number of faces on the corresponding processor patch and approximate the expected communication volume.

Minimizing inter-node communication reduces to a graph partitioning problem. The graph is partitioned into N parts of (almost) equal size N_c, such that the sum of

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

edge weights between different parts is minimized. This problem is analogous to mesh decomposition and is solved using e.g. the SCOTCH graph partitioning library [12]. Since exact partition sizes cannot always be guaranteed, a lightweight post-processing step is applied to enforce balanced partitions. Figure 4 shows this approach for $n_p = 36$ subdomains on $N = 3$ nodes. Compared to the canonical OpenFOAM rank assignment, the optimized partitioning substantially reduces inter-node communication edges.



(a) Example decomposition obtained by SCOTCH

(b) Comm. graph with canonical rank-to-core assignment

(c) Comm. graph with optimized rank-to-core assignment

Fig. 4: A decomposition with two possible rank-to-core assignments shown in (b) and (c). Vertices with same color are located on the same compute node. Dashed edges show intra-node communication, while thick edges show inter-node communication.

Hierarchical partitioning. After minimizing inter-node communication the same strategy is applied recursively to lower levels of the hardware hierarchy. Each node-level partition is treated as an induced subgraph and further partitioned into NUMA domains and subsequently into NUMA nodes with appropriate partition sizes.

This yields an explicit assignment of each process to a compute node, NUMA domain, and NUMA node. For reproducibility, ranks are renumbered so that rank 0 occupies the first core of the first node, with subsequent ranks ordered by increasing NUMA-domain and NUMA-node indices, preserving the communication structure.

Rank file generation. From the optimized assignment, a rank file is generated that binds each MPI rank to a specific core. This requires a mapping of the physical cores to NUMA domains and NUMA nodes. Assuming a homogeneous cluster architecture, the mapping is identical across nodes and is stored in two arrays of length N_c. Combining these informations yields a rank file specifying a fixed target node and core slot for each rank. We implemented this procedure in a custom OpenFOAM preprocessing utility.

Optimizing only the rank-to-core assignment improves local communication but introduces a subtle drawback. The default tree-based global reduction in OpenFOAM operates solely on MPI rank indices. As a result, ranks with nearby indices communi-

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

cate at lower tree levels, whereas distant ranks interact only at higher levels. Because OpenFOAM assigns ranks canonically in index order, independent of hardware topology, this default mapping implicitly aligns well with the index-based reduction tree.

When ranks are reassigned using a custom rank file, this implicit alignment between rank indices and the reduction tree is lost. As a result, the fixed index-based reduction tree may induce additional communication over slow hardware paths. Therefore, optimizing local communication alone is insufficient and must be complemented by a topology-aware global communication schedule.

4.2 Optimized Global Communication Schedule

To overcome the issue described above, the global reductions are reorganized to follow the hardware hierarchy. That is, reductions are first performed within NUMA nodes, then within NUMA domains, and finally across compute nodes. Using the known rank-to-core mapping and the hardware topology, a custom reduction tree is constructed. For each rank, exactly one successor (send target) and a list of predecessors (receive sources) are defined, with rank 0 serving as the root. The resulting schedule is written to an OpenFOAM dictionary and loaded once at the start of the simulation. Enabling custom schedules needs only minor modifications to the OpenFOAM communication layer. These changes and the implemented utility can be found in [14].

The optimized global communication schedule can be used either in conjunction with the optimized rank assignment or independently with the default OpenFOAM rank mapping, depending on the hardware and MPI implementation. The impact of the proposed optimizations on the performance is evaluated in the following section.

5 Experimental Results

The proposed topology-aware communication optimizations were evaluated using several OpenFOAM standard cases executed on the HPC systems described in Section 3.1. Each case was run in three configurations: (i) unmodified OpenFOAM, (ii) global optimization only, and (iii) combined local and global optimization.

The benchmarks include two- and three-dimensional cases with varying mesh sizes and solver types, representative for communication-intensive workloads, see Table 3. These cases can also be found in [14]. All experiments follow a strong-scaling setup with fixed problem size and increasing node counts. The preprocessing time ranges between 10–30 s (on both, LEC hpc2 and VSC-5). Since this utility is executed once per simulation setup, its cost is negligible w.r.t. the overall runtime.

Communication analysis. Using the generated rank files and reduction schedules, the distribution of communication across hardware levels was analyzed. Figure 5 shows the relative share of local and global communication across hardware paths for the *pitzDaily* case. For local communication, the mesh-induced communication graph is used, where edge weights correspond to processor-patch sizes. Based on the rank-to-core mapping, edges are assigned to hardware levels, and the communication volume of a single halo exchange is obtained by aggregating edge weights per level and normalizing by the total volume. For global communication, the reduction tree defines the communication

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

Table 3: Overview of experimental benchmark cases and simulation parameters.

Case	System	Geometry	Cells	Solver	Timesteps
pitzDaily	VSC-5	2D	19,560,000	GAMG	300
pitzDaily	LEC hpc2	2D	11,002,500	GAMG	300
laplacianFoam	VSC-5	3D	30,000,000	CG	500
laplacianFoam	LEC hpc2	3D	11,250,000	CG	500
cavity2D	VSC-5	2D	20,250,000	GAMG	50
cavity2D	LEC hpc2	2D	9,000,000	GAMG	50
cavity3D	VSC-5	3D	20,796,875	GAMG	100
cavity3D	LEC hpc2	3D	8,998,912	GAMG	100

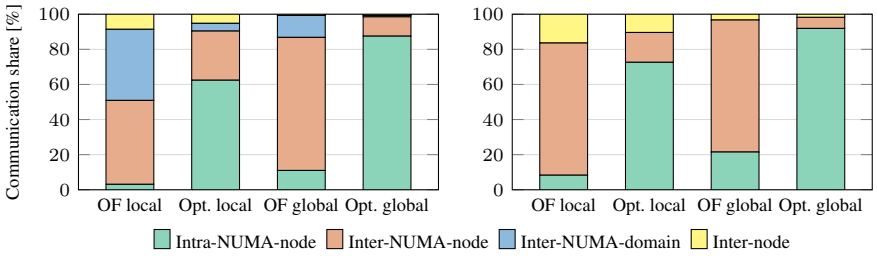


Fig. 5: Distribution of local and global communication across different hardware communication paths for *pitzDaily* on VSC-5 with six nodes (left) and LEC hpc2 with seven nodes (right) for OpenFOAM standard (OF) and the optimized versions (Opt.).

pattern, with unit weights per edge. The relative share is obtained by counting edges per hardware level. Different trees are used for the standard and optimized configurations. The optimized configuration shifts communication predominantly toward fast intra-NUMA-node paths, while communication across NUMA domains is substantially reduced. Similar trends were observed across all cases and node counts.

Runtime results. To assess the performance improvements, runtimes were measured for increasing numbers of compute nodes. For each configuration, multiple runs were performed, outliers were removed, and mean runtimes were recorded (Fig. 6 shows the scaling behavior). We decided to present the data on a log–log scale in order to accommodate a wide range of runtime values and to highlight scaling trends and relative performance differences. For the corresponding runtime decrease, comparing (i) unmodified OpenFOAM and (iii) combined local and global optimization, see Table 4.

Across all benchmarks, the combined local and global optimization consistently achieves the lowest runtime. On VSC-5, improvements are particularly pronounced, indicating that local communication dominates solver runtime on this architecture. On LEC, the difference between global-only and combined optimization is smaller, suggesting a comparatively stronger impact of global communication. On the LEC cluster, not all cases were executed up to ten nodes. Previous scaling studies showed performance degradation beyond moderate node counts for certain cases due to hardware and interconnect effects. To ensure meaningful comparisons, experiments were therefore restricted to configurations exhibiting stable scaling.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

Table 4: Relative runtime decrease (%) of the local and global optimized OpenFOAM compared to the standard version on different amount of compute nodes ($2 - 10$). A negative value means that the runtime increases. LEC denotes the cluster LEC hpc2.

Case	System	2	3	4	5	6	7	8	9	10
pitzDaily	VSC-5	6.0	16.1	12.8	23.8	23.5				
	LEC	−0.8	7.9	7.1	9.6	7.9	6.6			
laplacian	VSC-5	18.1	35.4	53.7	70.6	85.3				
	LEC	0.5	1.3	2.7	3.8	4.1	2.9	1.9	6.4	5.0
cavity2D	VSC-5	11.0	21.7	21.1	23.4	30.1				
	LEC	0.8	14.2	16.9	17.6					
cavity3D	VSC-5	3.2	4.7	6.4	6.3	8.1				
	LEC	−0.2	4.5	8.8	9.2	9.7	8.9	6.3	6.9	8.4

The observed performance differences between two- and three-dimensional cases are primarily determined by the communication characteristics of the iterative solver. In general, three-dimensional meshes involve more neighbouring cells and processor interfaces, leading to increased communication per iteration. However, the amount of communication per iteration also depends on the employed solver.

The two-dimensional cavity case exhibits larger speedups due to a higher number of iterations, resulting in increased communication, and thus greater optimization potential. The three-dimensional laplacian case setup also show significant improvements when the solver requires many iterations. Thus, the speedup depends on the mesh structure as well as solver characteristics. Although the magnitude of the speedup varies, all cases consistently show reduced runtime with topology-aware communication.

6 Conclusion and Future Work

This work demonstrates that topology-aware communication optimization can reduce the runtime of parallel OpenFOAM simulations. By optimizing process placement and reduction schedules w.r.t. the hardware hierarchy, communication in iterative linear solvers is shifted from slow inter-node and inter-socket paths to faster local links. Although we observe performance improvements across all benchmark cases, the achieved speedup depends on the hardware architecture and solver characteristics.

Future work will focus on extending the approach to dynamically changing decompositions. Possible directions include periodically recomputing rank placements and communication schedules based on updated processor patches, or incorporating communication-aware criteria directly into the mesh decomposition process.

Acknowledgments. We acknowledge the financial support of the "COMET - Competence Centers for Excellent Technologies" Program of the Austrian Federal Ministry for Innovation, Mobility and Infrastructure and the Austrian Federal Ministry of Economy, Energy and Tourism and the Provinces of Salzburg, Styria and Tyrol for the COMET Centre (K1) LEC GETS. The COMET Program is managed by the Austrian Research Promotion Agency (FFG). The computational results were partially achieved using the Austrian Scientific Computing (ASC) infrastructure. To improve the language and readability, we used GPT-5.2 of OpenAI [10].

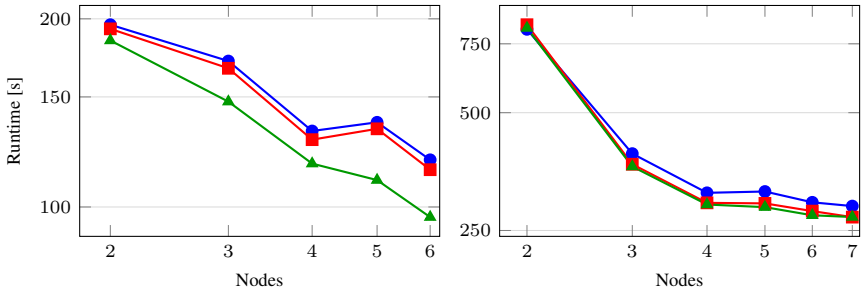
ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5

Disclosure of Interests. The authors have no competing interests.

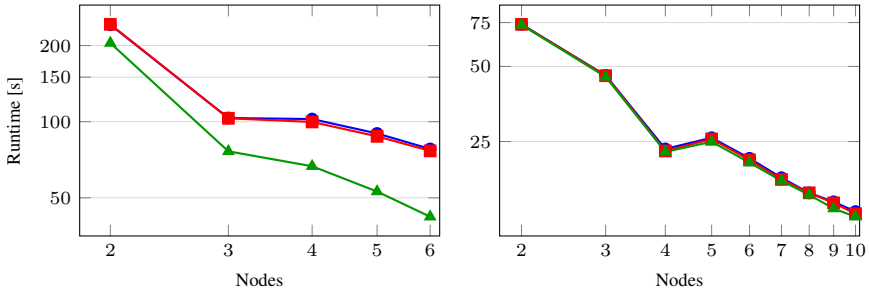
References

1. Caheny, P., Alvarez, L., Derradji, S., Valero, M., Moretó, M., Casas, M.: Reducing Cache Coherence Traffic with a NUMA-Aware Runtime Approach. IEEE Transactions on Parallel and Distributed Systems **29**(5), 1174–1187 (2018)
2. Culpo, M.: Current bottlenecks in the scalability of OpenFOAM on massively parallel clusters. PRACE white paper to appear on http://www. praceri. eu (2011)
3. García Flores, V.: Memory Hierarchies for Future HPC Architectures. Phd dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain (2017)
4. Greenshields, C., Weller, H.: Notes on Computational Fluid Dynamics: General Principles. CFD Direct Ltd, Reading, UK (2022)
5. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20**(1), 359–392 (1998)
6. Laso, R., Lorenzo, O.G., Cabaleiro, J.C., Pena, T.F., Lorenzo, J.A., Rivera, F.F.: CIMAR, NIMAR, and LMMA: Novel algorithms for thread and memory migrations in user space on NUMA systems using hardware counters. Future Gener. Comput. Syst. **129**, 18–32 (2022)
7. Mercier, G., Jeannot, E.: Improving MPI Applications Performance on Multicore Clusters with Rank Reordering. In: Recent Advances in the Message Passing Interface. pp. 39–49. Springer (2011)
8. Mirsadeghi, S.H., Afsahi, A.: Topology-Aware Rank Reordering for MPI Collectives. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1759–1768 (2016)
9. Moukalled, F., Mangani, L., Darwish, M.: The Finite Volume Method in Computational Fluid Dynamics - An Advanced Introduction with OpenFOAM and Matlab. Springer (2015)
10. OpenAI: ChatGPT-5.2 (2026), https://chatgpt.com/
11. Patterson, D.A., Hennessy, J.L.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann (2011)
12. Pellegrini, F.: Distillating knowledge about SCOTCH. In: Combinatorial Scientific Computing. Dagstuhl Seminar Proceedings (DagSemProc), vol. 9061, pp. 1–12 (2009)
13. Severance, C., Dowd, K.: High Performance Computing. OpenStax CNX (2010)
14. Sieberer, J.: Topology-Aware Communication Optimization (2026). https://doi.org/10.5281/zenodo.19368569
15. Strang, G., Herman, E.: Calculus Volume 3. OpenStax (2016), Chapter 6: Vector Calculus
16. Szustak, L., Lawenda, M., Arming, S., Bankhamer, G., Schweimer, C., Elsässer, R.: Profiling and optimization of python-based social sciences applications on HPC systems by means of task and data parallelism. Future Gener. Comput. Syst. **148**, 623–635 (2023)
17. Tan, L., Yufei, R., Dantong, Y., Shudong, J.: Analysis of numa effects in modern multicore systems for the design of high-performance data transfer applications. Future Gener. Comput. Syst. **74** (04 2017)
18. Versteeg, H.K., Malalasekera, W.: An Introduction to Computational Fluid Dynamics - The Finite Volume Method. Pearson Education, Amsterdam (2007)
19. Walker, D.W., Dongarra, J.J.: MPI: a standard message passing interface. Supercomputer **12**, 56–68 (1996)
20. Wang, H.: Algorithm Design for High-Performance CFD Solvers on Structured Grids. Phd dissertation, University of California, Irvine (2021)
21. Weller, H., Tabor, G., Jasak, H., Fureby, C.: A tensorial approach to computational continuum mechanics using object orientated techniques. Computers in Physics **12**, 620–631 (1998)

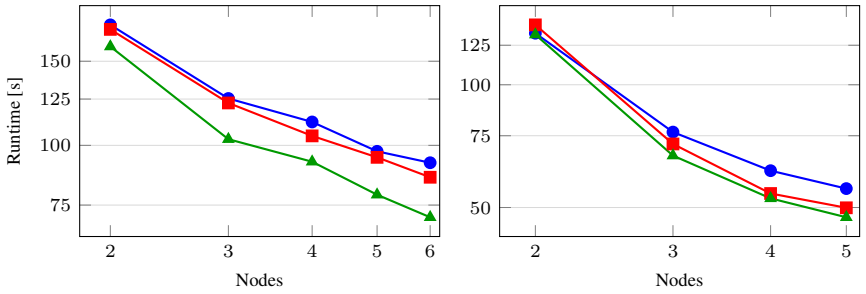
ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5



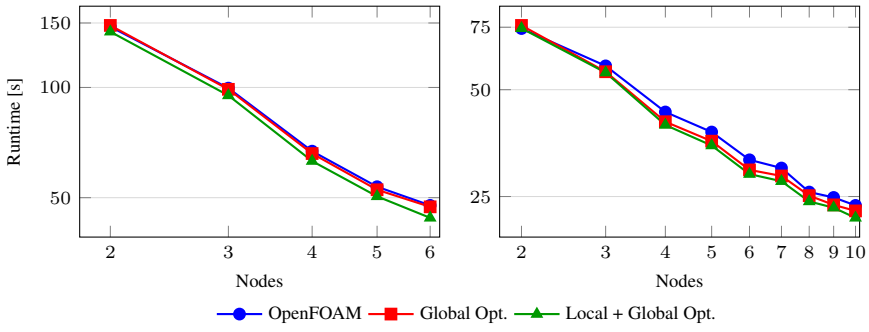
(a) Runtime of pitzDaily case on VSC-5 and LEC hpc2.



(b) Runtime of laplacianFoam case on VSC-5 and LEC hpc2.



(c) Runtime of cavity2D case on VSC-5 and LEC hpc2.



(d) Runtime of cavity3D case on VSC-5 and LEC hpc2.

Fig. 6: Runtime of all benchmark cases in the three configurations.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_5