

Characterizing Compute-Communication Duality in Halo-Exchange Masking for Seismic Wave Simulation

Francisco Pegoraro Etcheverria¹, Lucas Mello Schnorr¹

¹ Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

fpetcheverria@inf.ufrgs.br, schnorr@inf.ufrgs.br

Abstract. *Stencil-based seismic codes rely on halo exchange to overlap boundary communication with computation. Convolutional Perfectly Matched Layer (CPML) absorbing boundaries introduce a duality: subdomains in the corners, edges, and faces carry heavier compute loads but fewer neighbors, while interior subdomains have more neighbors but less work to mask communication. We implement a 3D decomposition for a CPML-based elastodynamic solver and evaluate it on Grid'5000 across two clusters, three interconnects, five problem sizes, and up to 125 MPI ranks. Trace-based analysis shows masking efficiency decreases monotonically with neighbor count across hardware setups and problem sizes, and that this gap is lower for large domains as higher volume-to-surface ratios and fixed-thickness CPML dilution improve masking. Minimizing communication volume via 3D decomposition does not resolve this duality, since effective masking depends on the per-rank co-location of compute capacity and communication cost.*

Resumo. *Códigos sísmicos baseados em stencil utilizam a troca de halo para sobrepor a comunicação de fronteira ao cálculo. As camadas absorventes Convolutional Perfectly Matched Layer (CPML) introduzem uma dualidade: subdomínios nos cantos, arestas e faces possuem cargas computacionais mais pesadas, porém menos vizinhos, enquanto subdomínios interiores têm mais vizinhos, mas menos trabalho para mascarar a comunicação. Implementamos uma decomposição 3D de um solver elastodinâmico com CPML, avaliada no Grid'5000 em dois clusters, três interconexões, cinco tamanhos de problema e até 125 ranks MPI. A análise baseada em traços mostra que a eficiência de mascaramento decresce monotonicamente com o número de vizinhos, e que essa diferença é menor em domínios maiores devido à maior razão volume-superfície e à diluição da região CPML de espessura fixa. Minimizar o volume de comunicação via decomposição 3D não resolve essa dualidade, pois o mascaramento efetivo depende da co-localização, por rank, entre capacidade de computação e custo de comunicação.*

1. Introduction

Stencil-based numerical codes drive the computational demands of many modern high-performance computing (HPC) applications, particularly in seismic modeling. In these distributed applications, halo exchange is the standard mechanism utilized to resolve data dependencies at the boundaries of partitioned domains [Kjolstad and Snir 2010]. By overlapping boundary communication with local domain computation, halo exchanges can

fully mask network latency. However, achieving an optimal balance between computation and communication is difficult, as it depends on several dynamic factors, including varying computation costs, communication volume, and hardware configuration.

Ondes3D [Aochi et al. 2013] is a representative example of this class of applications: a finite-difference elastodynamic solver for 3D seismic wave propagation, built around the staggered-grid velocity-stress formulation and CPML (Convolutional Perfectly Matched Layer) absorbing boundaries described in Section 3.1. Prior scalability studies of this simulator evaluate it under a 2D horizontal domain decomposition, which creates subdomains with high surface-to-volume ratios and higher communication costs. Furthermore, because each CPML boundary cell must update 18 recursive auxiliary memory variables and perform additional computations, subdomains along the outer boundaries of the simulated volume incur higher computational cost per iteration than interior subdomains of equal size, introducing spatial load imbalance [Dupros et al. 2013], [Tesser et al. 2019]. To investigate these communication mechanics under optimal partitioning, we implement a custom solver with the same numerical formulation as Ondes3D but with a fully 3D domain decomposition. This 3D partitioning directly addresses the 2D communication bottleneck by minimizing the surface-to-volume ratio of the subdomains, significantly reducing total network volume. However, it retains the structural asymmetry and spatial compute load imbalance between subdomains in the corners, edges, and faces and interior subdomains.

From the perspective of halo exchange, the structural imbalance reveals a duality across subdomain roles. 1/ Subdomains in the corners, edges, and faces intersect three, two, or one outer CPML layers, respectively, and therefore carry a larger computational workload that helps mask communication. However, since they lie at the boundary of the global domain, they have fewer neighbors than interior subdomains and therefore communicate less. 2/ Interior subdomains perform only the regular wave equations, giving them a smaller computational load. However, because they are interior, they have neighbors on all six sides, giving them the highest communication volume with the lowest amount of computation to potentially mask it. This duality generates a problem: the heterogeneity in communication volume and computational cost means that the benefits of halo exchange are location-dependent. Even under a uniform 3D geometric partitioning (for example, a $3 \times 3 \times 3$ grid), the imbalance in data transfer and local compute makes it difficult to achieve homogeneous communication masking across all MPI ranks. The severity of this duality is also sensitive to the global problem size. Because the CPML layer typically maintains a fixed cell thickness, in smaller domains CPML boundary cells constitute a higher percentage of the total workload, exacerbating the compute imbalance at the corners. As the domain size increases, the CPML fraction decreases, and the subdomain volume-to-surface ratio rises. This reduces the relative load imbalance and gives subdomains more compute relative to communication, but interior ranks, which perform only regular updates and communicate with all six neighbors, may lack sufficient computational weight to mask their communication fully.

This paper makes the following contributions. First, we characterize parallel efficiency and halo-exchange masking effectiveness across two clusters, three interconnects, five problem sizes, and up to 125 MPI ranks, showing that the two metrics track each other but capture distinct effects. Second, we analyze how increasing problem size narrows the

Corner-Interior masking gap by diluting the fixed-thickness CPML cost and increasing the local compute-to-communication ratio. Third, we conduct a systematic trace-based analysis on Grid’5000 that isolates the compute-communication duality by topological role (Corner, Edge, Face, Interior), showing that masking efficiency decreases monotonically with neighbor count across various cluster, network, and problem-size combinations. Finally, we pinpoint that reducing total communication volume via 3D decomposition does not resolve this duality, since masking effectiveness depends on the local co-location of compute capacity and communication cost at each rank rather than on global communication volume.

The remainder of this paper covers related work (Section 2), our implementation and experimental methodology (Section 3), results and discussion (Section 4), and conclusions (Section 5).

Software and Data Availability. We endeavor to make our analysis reproducible for a better science. Companion materials are available in a public repository at <https://gitlab.com/fpetcheverria/sscad-2026-companion>, containing the experimental data and source code of the program. We also include scripts to run the experiments and generate figures.

2. Related Work

Prior work on communication/computation overlap in earthquake codes has largely sidestepped the duality we tackle. Zhou et al. [Zhou et al. 2013] developed an overlap and communication-reduction scheme for a multi-GPU AWP-ODC implementation, but their decomposition is restricted to X and Y, leaving every subdomain with at most four neighbors and no CPML-induced compute asymmetry. Their strategy works precisely because compute load is homogeneous across ranks, which is not the case in our application. Okamoto et al. [Okamoto et al. 2010] use a full 3D decomposition for a multi-GPU FDM solver, exposing the same geometric neighbor-count asymmetry we observe. However, their absorbing boundary treatment adds negligible extra computation per cell, so boundary and interior subdomains carry essentially the same load. Our work differs in that CPML introduces a genuine compute-versus-communication duality across subdomains. Michéa and Komatitsch [Michéa and Komatitsch 2010] recognized the duality we tackle but did not analyze it. In their multi-GPU scaling experiments they deliberately excluded CPML layers because the higher operation count in CPML regions would complicate scaling measurements. They identify this as an open problem, noting that unequal CPML workloads across ranks could force the simulation to synchronize on the slowest node, and they argue for a decomposition that weights CPML versus regular grid points differently. Our work directly addresses this gap. Pekkilä et al. [Pekkilä et al. 2022] formalize the compute/communication race for stencil loops via a model that classifies subdomains as compute-bound or communication-bound from the ratio of local work to halo size. Their analysis, however, assumes spatially uniform compute load and periodic boundaries; it does not consider the case central to our work, where compute load varies with subdomain position due to CPML.

Related work on Ondes3D corroborates the compute-side half of this duality. Solórzano et al. [Solórzano et al. 2020] instrumented Ondes3D across eight multicore architectures and confirmed that border and corner ranks carry heavier, more variable computational load due to CPML and Free Surface computations. Dupros

et al. [Dupros et al. 2013] first reported the underlying elastodynamics scalability issues, and Tesser et al. [Tesser et al. 2019] addressed the compute imbalance via over-decomposition and Adaptive MPI. None of these works, however, incorporate communication time into the analysis: Solórzano et al. exclude inter-rank communication from their traces. The CPML-driven compute imbalance in Ondes3D is thus well documented, but its interaction with halo-exchange communication masking remains unexamined.

3. Materials and Methods

We next detail the governing equations of Ondes3D, our hybrid implementation and optimizations, and our evaluation methodology.

3.1. Theoretical Background

Our implementation of Ondes3D solves the elastic wave equation in its first-order velocity-stress form, following the staggered-grid finite-difference formulation of [Graves 1996]. For a linear, isotropic elastic medium, momentum conservation and the linear stress-strain relation combine into a first-order hyperbolic system relating the three velocity components v_x, v_y, v_z to the six independent stress components $\tau_{xx}, \tau_{yy}, \tau_{zz}, \tau_{xy}, \tau_{xz}, \tau_{yz}$:

$$\begin{aligned}\rho \partial_t v_x &= \partial_x \tau_{xx} + \partial_y \tau_{xy} + \partial_z \tau_{xz} + f_x, \\ \rho \partial_t v_y &= \partial_x \tau_{xy} + \partial_y \tau_{yy} + \partial_z \tau_{yz} + f_y, \\ \rho \partial_t v_z &= \partial_x \tau_{xz} + \partial_y \tau_{yz} + \partial_z \tau_{zz} + f_z,\end{aligned}\tag{1}$$

$$\begin{aligned}\partial_t \tau_{xx} &= (\kappa + \frac{4}{3}\mu) \partial_x v_x + (\kappa - \frac{2}{3}\mu)(\partial_y v_y + \partial_z v_z), \\ \partial_t \tau_{yy} &= (\kappa + \frac{4}{3}\mu) \partial_y v_y + (\kappa - \frac{2}{3}\mu)(\partial_x v_x + \partial_z v_z), \\ \partial_t \tau_{zz} &= (\kappa + \frac{4}{3}\mu) \partial_z v_z + (\kappa - \frac{2}{3}\mu)(\partial_x v_x + \partial_y v_y), \\ \partial_t \tau_{xy} &= \mu (\partial_y v_x + \partial_x v_y), \quad \partial_t \tau_{xz} = \mu (\partial_z v_x + \partial_x v_z), \quad \partial_t \tau_{yz} = \mu (\partial_z v_y + \partial_y v_z),\end{aligned}\tag{2}$$

where ρ is density, μ is the shear modulus, κ is the bulk modulus, and f_x, f_y, f_z are the body-force components used to inject sources into the grid. To prevent reflections at the edges of the domain, our implementation mimics the original Ondes3D [Dupros et al. 2008] and uses the Convolutional Perfectly Matched Layer (CPML) absorbing boundary condition, as formulated by [Komatitsch and Martin 2007]. The CPML replaces standard spatial derivatives with stretched coordinate derivatives $\partial_{\tilde{x}}$ and updates the recursive auxiliary memory variable ψ_x at each time step (Equation 3), where b_x and a_x depend on the damping parameters d_x, κ_x , and α_x :

$$\partial_{\tilde{x}} = \frac{1}{\kappa_x} \partial_x + \psi_x, \quad \psi_x^n = b_x \psi_x^{n-1} + a_x \partial_x v|^{n+1/2}\tag{3}$$

3.2. Hybrid OpenMP + MPI Implementation

We implement the numerical scheme in C++ using a hybrid OpenMP + MPI model, maintaining the same grid conventions and parameters as Ondes3D. We validated numerical correctness by comparing the nine velocity and stress fields against a reference Ondes3D

execution after 5000 iterations, obtaining relative L_2 -norm errors below 3×10^{-10} for velocities and 10^{-14} for stresses. Below, we outline our domain representation, two kernel enhancements over Ondes3D, and our asynchronous halo exchange.

Domain Representation. We restructured the execution model for 3D domain decomposition and improved memory utilization from the outset. Each 3D field component (the three velocity components and six stress components) is stored as a cache-line-aligned $X \times Y \times Z$ buffer with precomputed strides to allow for fast linear indexing. The medium is represented as a vertical stack of layers rather than as a full 3D model; therefore, medium coefficients such as ρ , μ , and κ have size Z . At the start of each iteration, source moments are calculated, stresses are computed, and finally, velocities are updated. The velocity and stress update kernels constitute the core computational cost of the simulator. Because the numerical scheme relies on a 4th-order finite-difference stencil, updating a single cell requires 12 neighboring values (spanning $[-2, -1, +1, +2]$ along each spatial axis), culminating in highly memory-bound behavior [Michéa and Komatitsch 2010].

Indirect Indexing in the CPML Cells. A significant memory optimization, adapted from Ondes3D, is the indirect indexing of the 18 CPML ψ auxiliary memory variables (nine for stress and nine for velocity updates). The CPML variables are only required at the outer edges of the global domain. To conserve memory bandwidth and capacity, these variables are allocated in packed buffers sized strictly to the volume of the CPML regions. During computation, threads map subdomain (x, y, z) coordinates to these dense buffers using a precomputed indirection array, avoiding the allocation of a massive, mostly empty 3D grid for the absorbing conditions. Additionally, as CPML coefficients b_x and a_x do not vary over time and require expensive exponential evaluations, they are precomputed at initialization and stored rather than recomputed at every iteration.

Enhancements over the base Ondes3D version.

1/ Avoiding Conditional Branching. Because interior cells and CPML boundary cells require different arithmetic updates, a naive loop covering the entire domain requires conditional branching on the cell type to determine which update to apply. This branching prevents compilers from generating efficient SIMD vectorized instructions. To resolve this, we employ a region-dispatching strategy. The rank’s subdomain is decomposed into up to seven distinct geometric regions: one pure interior volume and up to six independent CPML faces. Specialized kernels for velocity and stress updates exclusively on interior cells or CPML cells are dispatched on their respective regions. This branchless design allows the compiler to fully vectorize the inner Z-axis loops, annotated with `#pragma omp simd`, maximizing instruction throughput.

2/ Optimized Memory Access Patterns. With the domain partitioned, we optimize the memory access patterns of these regions. If threads iterate over the large interior volume linearly, the stencil’s wide spatial dependencies evict neighboring cells from the CPU cache, bottlenecking the solver on memory bandwidth. To maximize cache reuse, we implemented a 2D cache-blocking strategy. We decompose the domain into 8×8 tiles along the X and Y dimensions. OpenMP threads are dynamically scheduled across these tiles (`#pragma omp parallel for collapse(2)`), ensuring the memory loaded for the 13-point stencil remains hot in the L2/L3 caches while computing the contiguous Z dimension. The CPML boundary regions, being narrow, do not require this tiling.

Asynchronous Halo Exchange and Topological Heterogeneity.

To resolve the spatial dependencies of the 13-point stencil across rank boundaries in our full 3D Cartesian partitioning grid (P_x, P_y, P_z) , each rank’s subdomain is padded with a halo region of depth 2 in all three dimensions. To mask network latency, we overlap communication with computation. As velocity updates depend on stresses and vice versa, ranks begin each stress or velocity phase by first computing the two-cell-wide boundary slabs needed by neighbors. For each active neighbor direction (up to 6), the rank posts a non-blocking `MPI_Irecv`, packs the non-contiguous boundary slabs across all updated fields (six stress or three velocity components) into a buffer, and initiates `MPI_Isend`. While transfers progress asynchronously, threads compute the large remaining interior region; once finished, the rank invokes `MPI_Waitall` and unpacks the buffers into its halo cells when they are received. However, as detailed in Section 4, this 3D decomposition exposes a compute-communication duality. Subdomains in the corners, edges, and faces contain CPML cells and carry a heavier compute load, but communicate with fewer neighbors (3, 4, and 5, respectively). Conversely, interior subdomains solve only the cheaper regular wave equations but exchange halos with all 6 neighbors, making effective masking dependent on both problem size and rank location.

3.3. Performance Evaluation Methodology

Our goal is to characterize the compute-communication duality in 3D spatial decomposition quantitatively. We aim to measure how the structural heterogeneity of subdomains, arising from CPML boundary cells along the edges of the global grid, affects the application’s ability to mask communication. By isolating these subdomains, we can evaluate the influence of problem size on this duality and overall scalability.

Workflow. We established a profiling and trace-analysis workflow to reach that goal. The application is dynamically instrumented with an in-house tracing library, Akypuera¹, which captures fine-grained timestamps for all MPI function calls without introducing significant overhead. We then parse the trace data to extract exact MPI call timings and derive the compute workload for each rank. Because the application overlaps non-blocking communication with interior domain computation, we define a rank’s total wait time as the sum of its `MPI_Waitall` durations, which captures unmasked communication and synchronization time relative to the rank’s execution span.

Performance Metrics. We use three primary metrics to interpret these execution traces and evaluate the scalability of our spatial domain decomposition: Execution Time, Parallel Efficiency, and Masking Efficiency. Execution Time (T) is the duration of the iteration loop for a run using R ranks, measured by rank 0 in each run. All ranks call `MPI_Barrier` immediately before the first iteration of the loop and once again after the last iteration; T is the elapsed time between these two barriers. Parallel Efficiency (E) measures the overhead introduced by distributing the workload across multiple ranks compared to a baseline execution. It is calculated as: $E(R) = \frac{T_1}{R \times T_R}$ where T_1 is the execution time of the single-rank configuration, R is the total number of MPI ranks, and T_R is the execution time using R ranks. Masking Efficiency (M) quantifies the fraction of each rank’s time not spent waiting on halo exchange, and is computed independently for every rank. We define the makespan for a rank (T_{span}) as the time between the rank

¹<https://github.com/schnorr/akypuera>

leaving the barrier before the first iteration, and entering the barrier after the last iteration. Within this window, we define T_{wait} as the cumulative time the rank spent blocked in `MPI_Waitall` calls, reflecting both unmasked network transfers and time spent waiting on slower neighbors. Masking efficiency is defined as: $M = 1 - \frac{T_{wait}}{T_{span}}$. A value of 100% indicates that all halo transfers completed before the rank finished its interior updates. We report M at two levels of granularity: globally, as the median across all ranks in a run, and by topological role (Corner, Edge, Face, or Interior, having [3, 4, 5, 6] neighbors respectively).

3.4. Experimental Project

Hardware. Experiments were conducted on two distinct compute clusters within the Grid’5000 testbed: the `dahu` cluster and the `paradoxe` cluster. These clusters enable a comprehensive evaluation of the program’s scalability across different hardware platforms. Table 1 summarizes the architectural specifications of the compute nodes.

Table 1. Hardware specifications of the Grid’5000 nodes used for evaluation.

Parameter	dahu	paradoxe
CPU Model	Intel Xeon Gold 6130	Intel Xeon Gold 5320
Sockets per Node	2	2
Cores per Socket	16	26
Memory	192 GiB	384 GiB
Primary Network	100 Gbps Intel Omni-Path	25 Gbps Ethernet
Secondary Network	10 Gbps Ethernet	N/A

Software. The application was compiled using GCC 13.2.0. The MPI implementation used was Intel MPI version 2021.18. To guarantee overlap between inter-node communication and computation, we enabled Intel MPI’s asynchronous progress engine, dedicating a single thread per rank to advancing communication requests without interrupting numerical computation. We assigned two MPI ranks per compute node, with one MPI rank bound to each NUMA socket. Within each socket, one physical core was dedicated to the Intel MPI asynchronous progress thread, while all remaining cores were allocated to OpenMP threads for numerical computation. To prevent migration overhead, all threads were pinned to physical cores. MPI traces were captured using the Akypuera tracing library dynamically preloaded at runtime. **Design of Experiments.** To evaluate the scalability and communication overhead of the application, we formulated a full-factorial experimental design for each cluster. The design investigates the interactions between problem size, scale (number of MPI ranks), and network fabric. We evaluate the following factors: **1/ Problem Size:** Table 2 summarizes the five seismic domain sizes tested, each approximately three times larger in volume than the last; iteration counts were scaled down accordingly. The CPML boundary layer width was fixed at 10 grid cells along the outer faces, so its proportion of the total domain shrinks as the problem grows.

2/ Number of ranks: To evaluate scaling, we vary the number of MPI ranks R . Each rank corresponds to one socket in a node. On the `dahu` cluster, configurations scaled through [1, 2, 4, 8, 16, 27, 32, 45, 48] ranks. To reach the 48-rank scale on `dahu`, our allocation unavoidably included two nodes with a reduced memory configuration (5

Table 2. Problem sizes, iteration counts and CPML boundary proportions.

Grid size (N^3)	208	300	433	624	900
Iterations	3240	1080	360	120	40
CPML cells (%)	26.16	18.70	13.23	9.31	6.52

DIMMs rather than 6 in a socket). Rather than let these nodes fall at an arbitrary rank number, we placed them at the end of the hostfile, localizing the resulting compute imbalance to the final ranks so its effect could be isolated. On the larger `paradoxe` cluster, we scaled through $[1, 2, 4, 8, 16, 27, 32, 48, 64, 100, 125]$ ranks; and **3/ Network:** On the `dahu` cluster, every configuration was executed over both the 100 Gbps Intel Omni-Path interface and the standard 10 Gbps Ethernet interface. On `paradoxe`, the 25 Gbps Ethernet interface was used. For a given number of MPI ranks R , we decompose the spatial domain into a 3D grid of subdomains (P_x, P_y, P_z) such that $P_x \times P_y \times P_z = R$. To minimize communication costs, we chose the most cube-like partitioning, minimizing $P_x + P_y + P_z$. Additionally, we performed five repetitions of each configuration to assess variability, and we randomized their execution order within an experimental session. Average values come associated with standard error bars with a 99% confidence interval, and median values come associated with the IQR of measurements to quantify variability.

4. Results and discussion

Execution Time and Parallel Efficiency. To establish a baseline for scalability, we begin by examining the application’s execution time and parallel efficiency across hardware configurations and problem sizes. Figures 1 and 2 plot average execution time and parallel efficiency, respectively, as line plots against MPI rank count on a log scale on the x-axis, and lines colored by problem size, with standard error bars indicating a 99% confidence interval. Each plot is faceted by hardware/network setup: `dahu` with 10 Gbps Ethernet, `dahu` with 100 Gbps Omni-Path, and `paradoxe` with 25 Gbps Ethernet. As illustrated in these figures, the application exhibits strong scaling characteristics up to a certain rank count, and larger problem sizes consistently scale better than smaller ones. This behavior is expected: since communication volume grows with N^2 while computation grows with

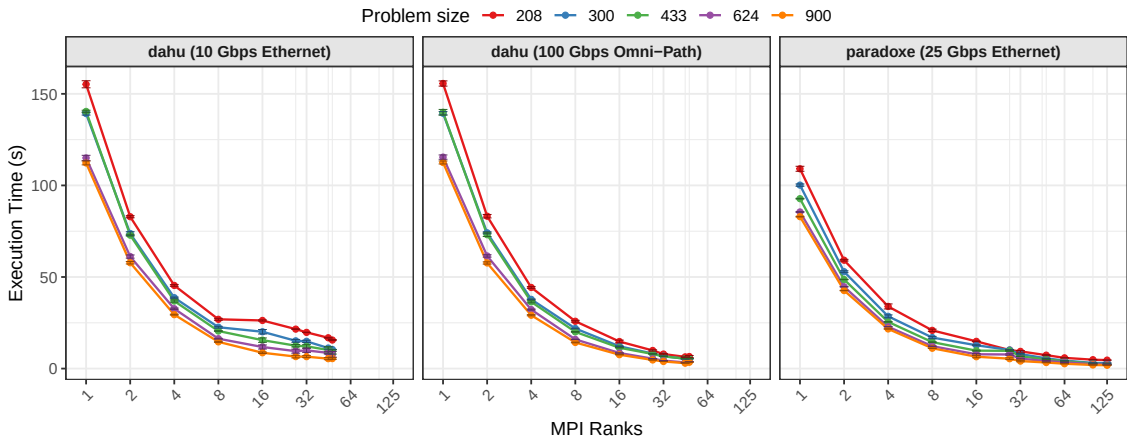


Figure 1. Execution time across setups and problem sizes.

N^3 , smaller problems have proportionally more communication, causing their efficiency to fall sooner. On the *dahu* cluster, the 100 Gbps Omni-Path interconnect yields substantially better parallel efficiency than the 10 Gbps Ethernet fabric, confirming that the application is highly sensitive to communication bandwidth and latency. Meanwhile, on the *paradoxe* cluster, we can see scaling for a much larger number of nodes, with efficiency dropping significantly to $\approx 37\%$ for the 900^3 problem at 125 ranks. *paradoxe*'s baseline execution times are lower than *dahu*'s, a consequence of its more powerful hardware, but the scaling behavior is inferior to *dahu* with 100 Gbps Omni-Path, suggesting its 25 Gbps interconnect is insufficient to keep pace with its higher compute throughput.

Two anomalies are visible in Figures 1 and 2. First, on *paradoxe* and on *dahu* with Omni-Path, parallel efficiency is lower at 27 ranks than at 32 ranks for several problem sizes, which we examine in more detail shortly. Second, execution time increases from 45 to 48 ranks on *dahu*. As noted in Section 3.4, two of the nodes had one memory module fewer than the others, lowering their memory bandwidth, and the resulting per-rank compute imbalance degraded the whole run's performance, visible as the sharp drop at the right edge of every *dahu* (100 Gbps Omni-Path) series in Figure 2.

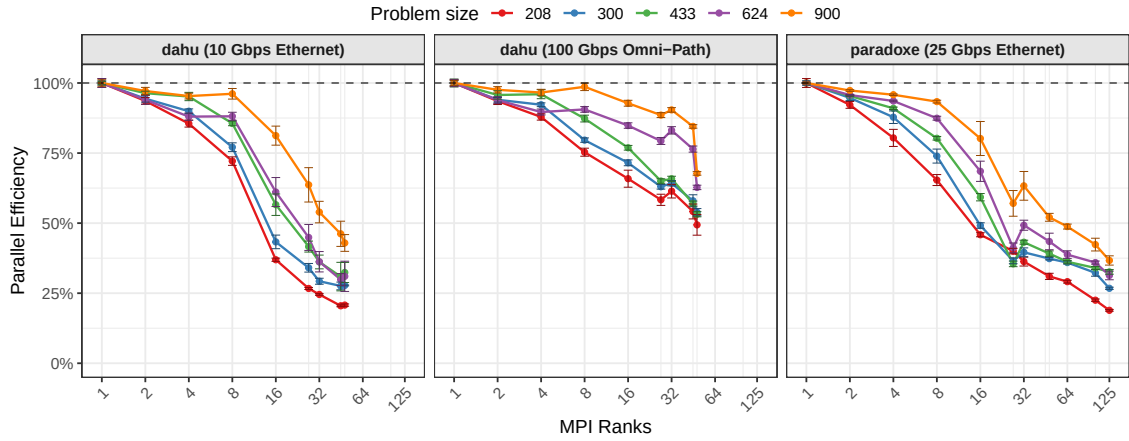


Figure 2. Parallel efficiency relative to one rank across setups and problem sizes.

This second anomaly depicted in Figure 2 exposes a structural weakness of the halo-exchange strategy. It implicitly assumes all ranks are roughly equal in computational and communication speed. Because of the tight data dependencies between neighboring subdomains, hardware heterogeneity, caused here by a memory configuration mismatch affecting only two of *dahu*'s nodes, propagates to the entire run, with every rank in the end being bounded by the slowest ones, whether directly or indirectly.

Halo-Exchange Masking Efficiency. Parallel efficiency depends on many aspects of the program, one of which is how well communication is masked by computation. To isolate this effect, we analyze the median masking efficiency for each configuration. Figure 3, which follows the same layout as Figures 1 and 2, depicts MPI ranks on a log-scale x-axis, with median masking efficiency on the y-axis and error bars indicating the IQR. Lines are colored by problem size, and facets separate the hardware setups. Masking efficiency shows a similar trend to parallel efficiency: larger problems generally have better masking, and *dahu* with Omni-Path is noticeably more favorable. At 45 ranks on *dahu* (Omni-Path), parallel efficiency ranges from 54.2% for the smallest problem size to

84.5% for the largest, while median masking efficiency remains higher (76.9% to 96.8%), indicating that unmasked `MPI_Waitall` time accounts for only part of the efficiency loss at this scale.

The same 27-vs-32-rank dip observed in Figure 2 is also visible here, most clearly for the `paradoxe` configuration, suggesting that the effect originates in communication masking rather than computation time alone. We hypothesize this stems from the specific (P_x, P_y, P_z) decomposition selected at 27 ranks: because $27 = 3^3$ is a perfect cube, there is a single interior rank with no CPML cells to compute, but the most communication to perform, therefore more time is spent idling in `MPI_Waitall`, lowering efficiency. 32 ranks, on the other hand, is more balanced, as its $4 \times 4 \times 2$ decomposition has no interior ranks. A trace analysis of the `dahu` 48-rank execution noted earlier also illustrates the duality from the opposite extreme: the two degraded-memory ranks individually reach a median masking efficiency of $\approx 100\%$ because their compute is slow enough that halo data always arrives before it is needed. Consequently, when analyzing masking efficiency, a pronounced gap where a set of ranks achieves near-perfect masking efficiency while others stall in `MPI_Waitall` often signals that those high-masking ranks are compute-bottlenecked, ultimately slowing the entire simulation.

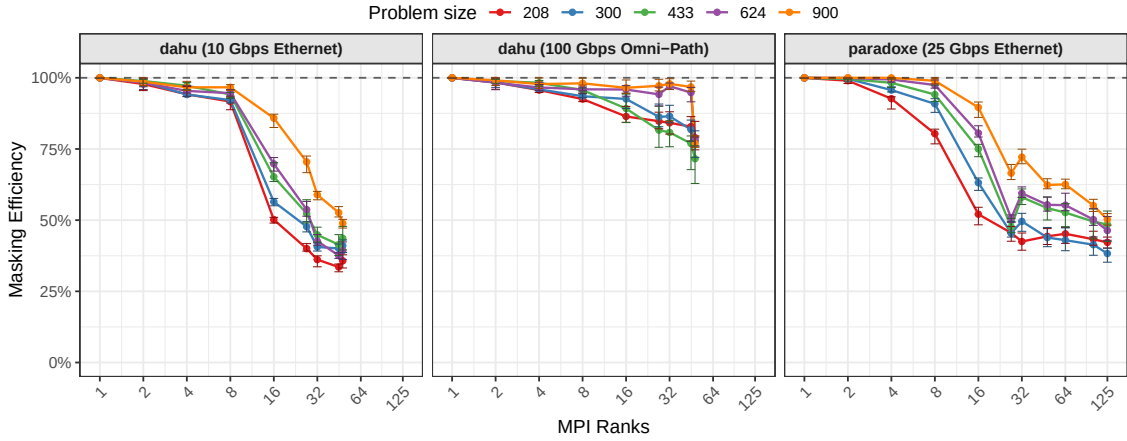


Figure 3. Median masking efficiency across setups and problem size.

To directly test the compute-communication duality hypothesized in Section 1, we classify each rank by its role (Corner, Edge, Face, or Interior) and examine their median masking efficiency independently. We focus on the `dahu` cluster at 45 ranks (both interconnects), which lets us isolate the effect of network speed on a fixed partition geometry, and on `paradoxe` at 125 ranks, its largest and geometrically richest configuration ($5 \times 5 \times 5$). To visualize this duality, Figure 4 uses box plots to show performance by grid placement, with scattered points indicating individual measurements. The x-axis categorizes the ranks into their topological roles. The y-axis measures masking efficiency, and the plot grid is faceted in two dimensions: vertically by hardware setup and rank count, and horizontally by problem size. This figure is consistent with the duality: in every cluster/network/problem-size combination examined, we can see that the median masking efficiency decreases as the number of neighbors increases. This penalty arises because interior subdomains not only have the least local computation to hide their 6-neighbor halo transfers, but may also stall in `MPI_Waitall` waiting on boundary neighbors de-

layed by heavier CPML compute. The duality thus creates an upper bound on parallel efficiency: because of the dependency chain, both unmasked network transfers at interior ranks and CPML compute delays at boundary ranks propagate across neighbors during a given iteration, ultimately inflating the total execution time.

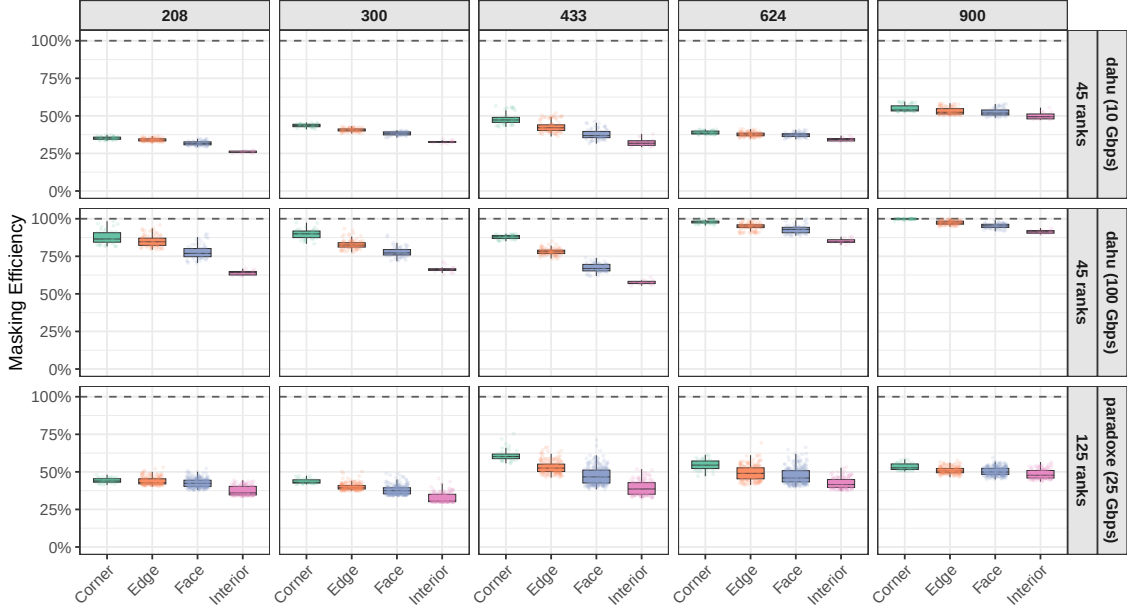


Figure 4. Masking efficiency by topological role, for dahu at 45 ranks and paradoxo at 125 ranks, across all problem sizes.

The size of the Corner-Interior gap, however, depends heavily on the network and problem size, as well as the number of ranks. Comparing dahu at 10 Gbps and 100 Gbps, we see that in the 10 Gbps setup, masking efficiency is consistently low (25.7%–51.8% for Interior ranks), and the gap between Corner and Interior ranks is clearly visible. On dahu with 100 Gbps Omni-Path, the picture changes substantially: masking efficiency is far higher overall, and the gap between Corner and Interior ranks is quite large at sizes 208^3 to 433^3 , narrowing as the problem grows to 624^3 and 900^3 . The 900^3 case is the only one to achieve 100% masking for Corner ranks, with Interior ranks achieving only 91.3%. Growing the problem from 433^3 to 900^3 raises Interior rank masking efficiency in all setups, lowering the relative imbalance between roles. This reflects both the lower proportion of CPML cells in larger domains (Section 3.4), which balances per-role compute, and the higher subdomain volume-to-surface ratio ($O(N^3)/O(N^2)$), which provides more work relative to communication volume.

This duality introduces a constraint for load-balancing strategies. A common strategy is to assign smaller subdomains to boundary ranks and larger subdomains to interior ranks, such that their compute cost is balanced. However, because interior ranks already have the highest communication costs, increasing their subdomain surface area further increases their communication overhead relative to the boundary ranks, potentially shifting the imbalance from computation to communication.

5. Conclusion

In this paper, we characterized the compute-communication duality in halo-exchange masking for a full 3D domain decomposition of a stencil-based seismic simulator. While transitioning from a 2D to a 3D geometric partitioning scheme minimizes the network communication volume by reducing the surface-to-volume ratio of the subdomains, our trace-based performance analysis reveals it does not resolve the inherent compute-communication duality caused by absorbing boundary conditions. This structural imbalance creates fundamentally unequal workloads across the topology. We demonstrated that the effectiveness of overlapping communication with computation is location-dependent. Subdomains positioned at the corners of the global grid, which are burdened with heavy CPML boundary calculations but communicate with three neighbors, more successfully mask their network latency. In contrast, interior subdomains, which process the cheaper standard wave equations but exchange halo regions with six neighbors, lack sufficient local compute to mask both their network transfers and the wait on slower, CPML-heavy neighbors. Consequently, this masking gap limits scalability, an effect especially pronounced on slower network fabrics or in smaller problems, where lower volume-to-surface ratios and a larger proportion of CPML cells exacerbate the imbalance across roles.

For future work, we plan to investigate non-uniform 3D domain decomposition techniques that balance computational weight and halo surface area, alongside weak-scaling and varying CPML thickness to decouple the effect of CPML dilution from surface-to-volume scaling. Additionally, we plan to port the core kernels to run on GPUs. This will drastically reduce the execution time of the numerical updates, shifting the application's compute-to-communication ratio, and requiring an evaluation of how effectively halo exchanges can be masked under much tighter temporal constraints.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001. This work was also partially funded by Petrobras under Agreement no. 2020/00182-5 and 2018/00263-5 with UFRGS. The authors thank PCAD/UFRGS for infrastructure access in the prior study's evaluation. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

References

- Aochi, H., Ulrich, T., Ducellier, A., Dupros, F., and Michéa, D. (2013). Finite difference simulations of seismic wave propagation for understanding earthquake physics and predicting ground motions: Advances and challenges. *Journal of Physics: Conference Series*, 454(1):012010.
- Dupros, F., Aochi, H., Ducellier, A., Komatitsch, D., and Roman, J. (2008). Exploiting intensive multithreading for the efficient simulation of 3D seismic wave propagation. In *Proceedings of the 11th IEEE International Conference on Computational Science and Engineering (CSE)*, pages 253–260.

- Dupros, F., Do, H.-T., and Aochi, H. (2013). On scalability issues of the elastodynamics equations on multicore platforms. *Procedia Computer Science*, 18:1226–1234.
- Graves, R. W. (1996). Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences. *Bulletin of the Seismological Society of America*, 86(4):1091–1106.
- Kjolstad, F. B. and Snir, M. (2010). Ghost cell pattern. In *Proceedings of the 2010 Workshop on Parallel Programming Patterns (ParaPloP)*, pages 1–9.
- Komatitsch, D. and Martin, R. (2007). An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. *Geophysics*, 72(5):SM155–SM167.
- Michéa, D. and Komatitsch, D. (2010). Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. *Geophysical Journal International*, 182(1):389–402.
- Okamoto, T., Takenaka, H., Nakamura, T., and Aoki, T. (2010). Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition. *Earth, Planets and Space*, 62(12):939–942.
- Pekkilä, J., Väisälä, M. S., Käpylä, M. J., Rheinhardt, M., and Lappi, O. (2022). Scalable communication for high-order stencil computations using CUDA-aware MPI. *Parallel Computing*, 111:102904.
- Solórzano, A. L. V., Navaux, P. O. A., and Schnorr, L. M. (2020). Temporal load imbalance on Ondes3D seismic simulator for different multicore architectures. In *Proceedings of the 2020 International Conference on High Performance Computing and Simulation (HPCS)*, pages 1–8.
- Tesser, R. K., Schnorr, L. M., Legrand, A., Heinrich, F. C., Dupros, F., and Navaux, P. O. A. (2019). Performance modeling of a geophysics application to accelerate the tuning of over-decomposition parameters through simulation. *Concurrency and Computation: Practice and Experience*, 31(11):1–21.
- Zhou, J., Cui, Y., Poyraz, E., Choi, D. J., and Guest, C. C. (2013). Multi-GPU implementation of a 3D finite difference time domain earthquake code on heterogeneous supercomputers. *Procedia Computer Science*, 18:1255–1264.