# Analytic Modeling of Idle Waves in Parallel Programs: Communication, Cluster Topology, and Noise Impact

Ayesha Afzal<sup>1</sup>, Georg Hager<sup>1</sup>, and Gerhard Wellein<sup>1,2</sup>

 <sup>1</sup> Erlangen Regional Computing Center (RRZE), 91058 Erlangen, Germany, ayesha.afzal@fau.de, georg.hager@fau.de
 <sup>2</sup> Department of Computer Science, University of Erlangen-Nürnberg, 91058 Erlangen, Germany, gerhard.wellein@fau.de

Abstract. Most distributed-memory bulk-synchronous parallel programs in HPC assume that compute resources are available continuously and homogeneously across the allocated set of compute nodes. However, long one-off delays on individual processes can cause global disturbances, so-called idle waves, by rippling through the system. This process is mainly governed by the communication topology of the underlying parallel code. This paper makes significant contributions to the understanding of idle wave dynamics. We study the propagation mechanisms of idle waves across the ranks of MPI-parallel programs. We present a validated analytic model for their propagation velocity with respect to communication parameters and topology, with a special emphasis on sparse communication patterns. We study the interaction of idle waves with MPI collectives and show that, depending on the implementation, a collective may be transparent to the wave. Finally we analyze two mechanisms of idle wave decay: topological decay, which is rooted in differences in communication characteristics among parts of the system, and noise-induced decay, which is caused by system or application noise. We show that noise-induced decay is largely independent of noise characteristics but depends only on the overall noise power. An analytic expression for idle wave decay rate with respect to noise power is derived. For model validation we use microbenchmarks and stencil algorithms on three different supercomputing platforms.

## 1 Introduction

#### 1.1 Idle waves in barrier-free bulk-synchronous parallel programs

Parallel programs with alternating computation and communication phases and without explicit synchronization are ubiquitous in high performance computing. In theory, when running on a clean, undisturbed system and lacking any load imbalance or other irregularities, such applications should exhibit a regular lockstep pattern. In practice, however, a variety of perturbations prevent this: system and network noise, application imbalance, and delays caused by one-off events such as administrative jobs, message re-transmits, I/O, etc. Among all of these, long one-off events have the most immediate impact on the regular compute-communicate pattern. They cause periods of idleness in the process where they originated, but via inter-process dependencies they "ripple" through the system and can thus impact all other processes as well. In massively parallel programs,

delays can occur anytime, impeding the performance of the application. On the other hand, idle waves may also initiate desynchronization among processes, which is not necessarily disadvantageous since it can lead to automatic communication overlap [3].

The speed and overall characteristics of idle wave propagation have been the subject of some scrutiny [10, 12, 4, 3], but a thorough analytical understanding of their dynamics with respect to the communication topology of the underlying parallel code is still lacking. There is also no investigation so far of the interaction of idle waves with global operations such as reductions, and how the system's hardware topology and the particular characteristics of system noise impact the decay of idle waves. These topics will be covered by the present work. We restrict ourselves to *process-scalable* scenarios, i.e., where multiple MPI processes running on a hardware contention domain (such as a memory interface or a shared out-level cache) do not feel scalability loss due to hardware bottlenecks.

### 1.2 Related work

Noise has been studied for almost two decades. A large part of the work focuses on sources of noise outside of the control of the application and explores the influence of noise on collective operations [11, 6, 8]. However, it lacks coverage of pair-wise communication and the interaction of noise with idle periods, which are common in distributed-memory parallel codes. Gamell et al. [7] noted the emergence of idle periods in the context of failure recovery and failure masking of stencil codes. Markidis et al. [10] used a LogGOPS simulator [8] to study idle waves and postulated a linear wave equation to describe wave propagation.

Afzal et al. [4, 2, 3, 1] were the first to investigate the dynamics of idle waves, (de)synchronization processes, and computational wavefront formation in parallel programs with core-bound and memory-bound code, showing that nonlinear processes dominate there. Our work builds on theirs to significantly extend it for analytic modeling with further influence factors, such as communication topology, communication concurrency, system topology and noise structure.

Significant prior work exists on the characterization of noise and the influence of noise characteristics on performance of systems. Ferreira et al. [6] noted that HPC applications with collectives can often absorb substantial amounts of high-frequency noise, but tend to be affected by low-frequency noise. Agarwal et al. [5] found noise properties to matter for the scalability of collectives, comparing different distributions (exponential, heavy tail, Bernoulli). Hoefler et al. [9] used their LogGOPS-based simulator and studied both point-to-point (P2P) and collective operations. They found that application scalability is mostly determined by the noise pattern and not the noise intensity.

In the context of idle wave propagation and decay, the present work finds that the noise intensity is the main influence factor rather that its detailed statistics.

### 1.3 Contribution

This work makes the following novel contributions:

- We analytically predict the propagation velocity of idle waves in scalable code with respect to (i) communication topology, i.e., the distance and number of neighbors in point-to-point communication, and (ii) communication concurrency, i.e., how many point-to-point communications are grouped and subject to completion via MPI\_Waitall.
- The analytical model is validated with measurements on real systems and applied to microbenchmarks with synthetic communication topologies and a realistic scenario from the context of stencil codes with Cartesian domain decomposition.
- We show that not all MPI collective routines eliminate a traveling idle wave; some may even be almost transparent to it, depending on their implementation.
- We show that idle wave decay can also be initiated by the system topology via inhomogeneities in point-to-point communication characteristics between MPI processes.
- We show analytically that the decay rate (and thus the survival time until running out) of an idle wave under the influence of noise is largely independent of the particular noise characteristics and depends only on the overall noise power. This prediction is validated with experiments.

**Overview** This paper is organized as follows: Section 2 provides details about our experimental environment and methodology. In Section 3, we first introduce some important terms to categorize execution and communication in distributed-memory parallel programs and then develop and validate an analytical model of delay propagation. Section 4 covers the interaction of idle waves with collective primitives. An analysis of idle wave decay with respect to noise and system topology is conducted in Section 5. Finally, Section 6 concludes the paper and gives an outlook to future work.

# 2 Test bed and experimental methods

The three clusters listed in Table 1 were used to conduct various experiments and validate our analytical models.

Process-core affinity was enforced using the I\_MPI\_PIN\_PROCESSOR\_LIST environment variable. We ignored the simultaneous multithreading (SMT) feature and used only physical cores. The clock frequency was always fixed to the base value of the respective CPUs (or to 2.3 GHz in case of SuperMUC-NG because of the power capping mechanism). On Emmy, experiments with up to 120 nodes were conducted on a set of nodes connected to seven 36-port leaf switches in order to achieve homogeneous communication characteristics. A similar strategy was not possible on the other systems. Open-chain boundary conditions were employed unless specified otherwise. Communication delays for non-blocking calls were measured by time spent in the MPI\_Wait or MPI\_Waitall function. We used Intel Trace Analyzer and Collector (ITAC)<sup>4</sup> for timeline visualization and the C++ high-resolution Chrono clock for timing measure-

<sup>&</sup>lt;sup>3</sup> https://anleitungen.rrze.fau.de/hpc/emmy-cluster

<sup>&</sup>lt;sup>4</sup> https://software.intel.com/en-us/trace-analyzer

Systems	Emmy <sup>3</sup>	SuperMUC-NG	Hawk
Processor	Intel Xeon Ivy Bridge EP	Intel Xeon Skylake SP	AMD EPYC Rome
Processor Model	E5-2660 v2	Platinum 8174	7742
Base clock speed	2.2 GHz	3.10 GHz (2.3 GHz used*)	2.25 GHz
Physical cores per node	20	48	128
Numa domains per node	2	2	8
LLC size	25 MB	33 MB	$256 \text{ MB} = 16 \times 16 \text{ MB} / \text{CCX} (4\text{C})$
Memory per node (type)	64 GB (DDR3)	96 GB (DDR4)	$4 \text{ TB} = 16 \times 256 \text{ GB} \text{ (DDR4)}$
Node interconnect	QDR InfiniBand	Omni-Path	HDR InfiniBand
Interconnect topology	Fat-tree	Fat-tree	Enhanced 9D-Hypercube
Raw bandwidth p. lnk n. dir	$40{ m Gbits^{-1}}$	$100\mathrm{Gbits^{-1}}$	$200{ m Gbits^{-1}}$
Compiler	Intel C++ v2019.5.281	Intel C++ v2019.4.243	Intel C++ v2020.0.166
Optimization flags	-O3 -xHost	-O3 -qopt-zmm-usage=high	-O3 -xHost
SIMD	-xCORE-AVX2	-xCORE-AVX512	-mavx2
Message passing library	Intel MPI v2019u5	Intel MPI v2019u4	Intel MPI v2019u6
Operating system	CentOS Linux v7.7.1908	SESU Linux ENT. Server 12 SP3	CentOS Linux 8.1.1911
ITAC	v2019u4	v2019	v2020
	Systems Processor Processor Model Base clock speed Physical cores per node Numa domains per node LLC size Memory per node (type) Node interconnect Interconnect topology Raw bandwidth p. Ink n. dir Compiler Optimization flags SIMD Message passing library Operating system ITAC	SystemsEmmy³ProcessorIntel Xeon Ivy Bridge EPProcessor ModelE5-2660 v2Base clock speed2.2 GHzPhysical cores per node20Numa domains per node2LLC size25 MBMemory per node (type)64 GB (DDR3)Node interconnectQDR InfiniBandInterconnect topologyFat-treeRaw bandwidth p. Ink n. dir40 Gbits <sup>-1</sup> CompilerIntel C++ v2019.5.281Optimization flags-03 -xHostSIMD-xCORE-AVX2Message passing libraryIntel MPI v2019u5Operating systemCentOS Linux v7.7.1908ITACv2019u4	SystemsEmmy <sup>3</sup> SuperMUC-NGProcessorIntel Xeon Ivy Bridge EPIntel Xeon Skylake SPProcessor ModelE5-2660 v2Platinum 8174Base clock speed2.2 GHz3.10 GHz (2.3 GHz used*)Physical cores per node2048Numa domains per node22LLC size25 MB33 MBMemory per node (type)64 GB (DDR3)96 GB (DDR4)Node interconnectQDR InfiniBandOmni-PathInterconnect topologyFat-treeFat-treeRaw bandwidth p. Ink n. dir40 Gbits <sup>-1</sup> 100 Gbits <sup>-1</sup> CompilerIntel C++ v2019.5.281-03 -qopt-zmm-usage=highSIMD-xCORE-AVX2r-xCORE-AVX512Message passing libraryIntel MPI v2019u5Intel MPI v2019u5Optrating systemCentOS Linux v7.7.1908SESU Linux ENT. Server 12 SP3ITACv2019u4v2019

Table 1: Key hardware and software specifications of systems.

\* A power cap is applied on SuperMUC-NG, i.e., the CPUs run by default on a lower than maximum clock speed (2.3 GHz instead of 3.10 GHz).

ments. For tuning of the Intel MPI collectives implementations, we used the Intel MPI *autotuner*<sup>5</sup>; the configuration space is defined by I\_MPI\_ADJUST\_<opname><sup>6</sup>.

We run barrier-free bulk-synchronous MPI-parallel micro-benchmarks with configurable latency-bound communication and compute-bound workload. This results in process scalability, i.e., there is no contention on memory interfaces, shared caches, or network interfaces. The code loops over back-to-back divide instructions (vdivpd), which have low but constant throughput. The message size was set to 1024 B, which is well within the default eager limit of the MPI implementation. For more realistic workloads we chose a 3D Jacobi stencil and sparse matrix-vector multiplication (SpMV) with the High Performance Conjugate Gradient (HPCG)<sup>7</sup> matrix. Further characterization will be addressed in Section 3. One-off idle periods were generated by massively extending one computational phase via doing extra work on one MPI rank, usually rank 5.

All experiments described in this paper were conducted on all three benchmark systems. However, we show the results for all of them only if there are relevant differences.

## **3** Idle wave propagation velocity for scalable code

In this section we first categorize the execution and communication characteristics of parallel applications. Later, we investigate how they influence the idle wave velocity and construct an analytic model for the latter.

<sup>&</sup>lt;sup>5</sup> https://software.intel.com/content/www/us/en/develop/documentation/ mpi-developer-reference-linux/top/environment-variable-reference/ tuning-environment-variables/autotuning.html

<sup>&</sup>lt;sup>6</sup> https://software.intel.com/content/www/us/en/develop/documentation/ mpi-developer-reference-windows/top/environment-variable-reference/ i-mpi-adjust-family-environment-variables.html

<sup>&</sup>lt;sup>7</sup> https://www.hpcg-benchmark.org/



Fig. 1: Compact and non-compact communication topologies with bidirectional open chain characteristics.  $P_i$  sends (receives) data to (from)  $P_{i\pm 1}$  (a) till  $P_{i\pm 2}$  (b) till  $P_{i\pm 6}$  (c) till  $P_{i\pm 12}$ , (d) and  $P_{i\pm 6}$  (e) and  $P_{i\pm 12}$ .

#### 3.1 Execution characteristics

HPC workloads have a wide spectrum of requirements regarding code execution towards resources of the parallel computing platform. The most straightforward categorization is whether the workload is sensitive to certain resource bottlenecks, such as memory bandwidth. Since we restrict ourselves to scalable code here, we run the traditionally memory-bound algorithms such as stencil updates or SpMV with one MPI process per contention domain (typically a ccNUMA node). This is not a problem for the microbenchmarks since we deliberately choose an in-core workload there.

## 3.2 Categorization of communication characteristics

Here we briefly describe the different communication characteristics under investigation. We start by assuming a "P2P-homogeneous" situation where all processes (except boundary processes in case of open boundary conditions) have the same communication partners and characteristics. We will later lift this restriction and cover more general patterns.

**Communication topology** Communication topology is a consequence of the physical problem underlying the numerical method and of the algorithm (discretization, geometry). It boils down to the question "which other ranks does rank *i* communicate with?" and is characterized by a *topology matrix* (see Figure 1 for examples of *compact* and *noncompact* topologies).

In a compact topology, each process communicates with a dense, continuous array of neighbors with distances  $d = \pm 1, \pm 2, ..., \pm j$ . The topology matrix comprises a dense band around the main diagonal. In a noncompact topology, each process communicates with processes that are not arranged as a continuous block, e.g.,  $d = \pm 1, \pm j$ . In both variants, the topology matrix can be symmetric or asymmetric.

Table 2: Selected algorithms for communication concurrency in our MPI microbenchmarks. Arrows of the same color correspond to a single MPI\_Waitall call. "One distance" means that one MPI\_Waitall is responsible only for the send/recv pair of one particular communication distance, while "all distances" means that it encompasses all distances in one dimension.



 $\ddagger P_i$  send to  $P_{i+dir \times d}$ ; §  $P_i$  receive from  $P_{i-dir \times d}$ 

For example, sparse matrices emerging from numerical algorithms with high locality lead to compact communication structures, while stencil-like discretizations on Cartesian grids lead to noncompact structures with far-outlying sub-diagonals. Figures 1(a)–(c) depict symmetric cases with 4, 12, and 24 neighbors, respectively (2, 6 and 12 distinct processes per direction) for every process, while there are always four neighbors (two distinct processes per direction) for both noncompact cases in Figures 1(d)–(e).

**Communication concurrency** When a process communicates with others, it is often a deliberate choice of the developer which communications are grouped together and later finished using MPI\_Waitall ("split-waits"). However, since interprocess dependencies have an impact on idle wave propagation, such details are relevant. Of course, beyond user-defined communication concurrency, there could still be nonconcurrency "under the hood," depending on the internals of the MPI implementation.

Here we restrict ourselves to a manageable subset of options that nevertheless cover a substantial range of patterns. We assume that all P2P communication is nonblocking. Table 2 shows the four variants covered here in a 2D Cartesian setting according to the number of split-waits: *multi-wait, single-dimension* (MWSDim), *multi-wait, multi-dimension* (MWMDim), *single-wait, multi-dimension* (SWMDim), and *multi-wait, single-direction* (MWSDir). The iteration space of loops in Table 2 is defined as the outer (d) loop goes over the Cartesian dimensions (i.e., *x* and *y* here) and the inner (dir) loop goes over the two directions per dimension (i.e., positive and negative). For each direction (e.g., positive *x*), the communication is effectively a linear shift pattern; the pairing of send and receive operations per MPI\_Waitall ensures that no deadlocks will occur. The third and fourth option are corner cases with minimum and maximum number of MPI\_Waitalls.

More complex patterns Beyond the simple patterns described above, we will also cover more general *P2P inhomogeneous* communication scenarios, where subsets of

processes have different communication properties, such as in stencil codes or sparsematrix algorithms. Figure 4 shows an example with compact long-range and short-range communication, which could emerge from a sparse-matrix problem with "fat" and "skinny" regions of the matrix. Finally, we will discuss implementation alternatives of collective communication primitives.

### 3.3 Analytical model of idle wave propagation

The propagation speed of an idle wave is the speed, in ranks per second, with which it ripples through the system. Previous studies of idle wave mechanisms on silent systems [3, 4] characterized the influence of execution time, communication time, communication characteristics (e.g., uni- vs. bidirectional communication patterns and eager vs. rendezvous protocols), and the number of active multi-threaded or single-threaded MPI processes on a contended or noncontended domain. However, the scope of that work was restricted to a fixed P2P communication pattern (fourth column in Table 2 – MWSDir). Here we extend the analysis to more general patterns, which show a much richer phenomenology. We restrict ourselves to open boundary conditions across the MPI ranks. This is not a severe limitation since it only affects the survival time and not the propagation speed of the wave.

**Corner cases** Minimum idle wave speed (and thus maximum survival time) is observed with simple direct next-neighbor communication (d = 1). If  $T_{\text{exec}}$  and  $T_{\text{comm}}$  are execution and communication times of one iteration of the bulk-synchronous program, then the idle wave speed is

$$v_{\text{silent}}^{\min} = 1 \left[ \frac{\text{ranks}}{\text{iter}} \right] \times \frac{1}{T_{\text{exec}} + T_{\text{comm}}} \left[ \frac{\text{iter}}{\text{s}} \right]$$
 (1)

In this case, the wave survives until it runs into system boundaries [4], i.e., for at most as many time steps as there are MPI ranks. Barrier-like, i.e., long-distance synchronizing communication leads to maximum speed and the wave dying out quickly in a minimum of one time step. Thus, in this case,

$$v_{\text{silent}}^{\text{max}} = \alpha \left[ \frac{\text{ranks}}{\text{iter}} \right] \times \frac{1}{T_{\text{exec}} + T_{\text{comm}}} \left[ \frac{\text{iter}}{\text{s}} \right] ,$$
 (2)

where  $\alpha$  depends on the rank  $r_{inject}$  where the idle wave originated:

$$\alpha = \max \left( \texttt{MPI_Comm\_size} - r_{\texttt{inject}} - 1, r_{\texttt{inject}} - 1 \right) . \tag{3}$$

*Multi-neighbor communication* Away from the extreme cases, we have to distinguish between compact and noncompact multi-neighbor communication patterns, but the basic mechanisms are the same. The propagation speed of the idle wave can be analytically modeled as

$$v_{\text{silent}} = \kappa \cdot v_{\text{silent}}^{\min} \left[ \frac{\text{ranks}}{\text{s}} \right] ,$$
 (4)



Fig. 2: Top row: Idle wave propagation for 60 iterations in a core-bound microbenchmark for an injected delay at rank 5 (see text for details) and compact communication patterns with different numbers of communication partners: (a) two, (b) six, and (c) twelve partners per direction. The second row of panels shows the fraction of MPI ranks executing MPI library code.

Where  $\kappa$  depends on communication concurrency and topology:

$$\kappa = \begin{cases} \sum_{k=1}^{j} k = \frac{j(j+1)}{2} & \text{if compact MWSDim / MWSDir / blocking} \\ \sum_{k=1,j} k = j+1 & \text{if non-compact MWSDim / MWSDir / blocking} \\ j & \text{if MWMDim / SWMDim} \end{cases}$$
(5)

Here, *j* is the longest-distance communication partner of a rank. Modifications to these expressions may apply for complex communication topologies; we will discuss them in the validation section.

#### 3.4 Experimental validation

In this section, we first validate the analytical model via measurements using synthetic benchmarks on a real system. Thereafter, we apply the model to a 3D a stencil code with Cartesian domain decomposition. Since stencil codes are commonly memory-bound, we run a single thread per ccNUMA domain only in order to maintain resource scalability. Since the phenomenology matches across all three clusters (Table 1), we show results only for the Emmy system.

**Microbenchmarks** Figures 2 and 3 (top row) show traces of the propagation of injected one-off idle phases (extra work at at rank 5, dark blue) and its dependency on communication concurrency and communication topology, using the variants shown in Table 2. In these experiments, we used an execution phase of  $T_{\text{exec}} = 13 \text{ ms}$  (light blue) and a



Fig. 3: Idle wave propagation in a core-bound microbenchmark for an injected delay at rank 5 (see text for details) and noncompact communication patterns with two communication partners per direction at different distances on Emmy: (a)  $P_i \rightleftharpoons (P_{i\pm 1}, P_{i\pm 6})$  for 60 iterations and (b)  $P_i \rightleftharpoons (P_{i\pm 1}, P_{i\pm 12})$  for 20 iterations.

data volume of 1 KiB per message. The insets show close-ups of parts of the wave. In the second row, a quantitative timeline of the number of MPI processes executing MPI library code (i.e., waiting or communicating) is displayed. In these settings, the natural system noise is weak enough to not cause decay of the idle wave until it runs into the system boundary.

**Compact communication** In Figure 2, the observed propagation speed of the idle waves is independent of the number of split-waits, as expected. Higher speeds are observed when (i) the overall communication distance goes up, i.e., with growing number of communication partners, and (ii) the number of dimensions spanned within each MPI\_Waitall (communication concurrency). In Figure 2(a), where  $P_i \rightleftharpoons (P_{i\pm 1}, P_{i\pm 2})$ , higher speed results in (a1) with  $\kappa = \sum_{k=1}^{2} k = 3$  due to the MWSDim concurrency pattern, while in (a2) we have  $\kappa = j = 2$  for the other patterns. The data confirms the model in (4) and (5).

In Figure 2(b) and (c), the number of communication partners per direction is increased to six and twelve, respectively, with expected consequences: In (b1) we have  $\kappa = \sum_{k=1}^{6} k = 21$ , and in (b2)  $\kappa = j = 6$ . In (c1), we get  $\kappa = \sum_{k=1}^{12} k = 78$ , confirming intuitively our prediction that survival time in the high-speed limit is equal to  $T_{\text{exec}} + T_{\text{comm}}$ . Finally, in (c2) we get  $\kappa = j = 12$ .

The second row in Figure 2 shows that slower wave propagation causes a more even spread of waiting times and thus resource utilization across ranks. A rising/constant/falling slope indicates an oncoming/traveling/leaving wave. Although our particular scenarios have been designed to show no resource bottlenecks, these utilization shapes will be significant in case of memory-bound execution or bandwidth-contended communication [3]. An exploration of these mechanisms is left for future work.

*Noncompact communication* Topology matrices with noncompact characteristics (Figures 1(d)–(e)) entail a more complex phenomenology of idle wave propagation. The presence of "gaps" leads to multiple waves propagating at different speeds, with the

Fig. 4: Idle wave propagation with inhomogeneous compact communication charactersitics (60 iterations) on Emmy. (a) Topology matrix:  $P_i$  sends (receives) 1 KiB to (from)  $P_{i\pm 1}, \ldots, P_{i\pm 3}$  for processes near boundaries and to (from)  $P_{i\pm 1}, \ldots, P_{i\pm 12}$  for 40 inner processes. (b) Idle wave propagation for SWMDim concurrency.



added complication that each "hop" of a faster wave sparks local idle waves wherever it hits (see Figure 3). These secondary waves propagate and annihilate each other eventually (more specifically, after j/2 hops), and what remains is the fast wave emerging from the longest-distance communication. The speed of this residual wave is faster with (i) a larger number of split-waits, (ii) a smaller number of communication dimensions spanned by each MPI\_Waitall, and evidently (iii) a larger longest communication distance j.

With respect to communication concurrency, there is a fundamental difference between multiple split-waits and one wait-for-all in non-compact communication. The "zig-zag" pattern emerging from the two different propagation speeds prevails in case of SWMDim (one wait-for-all) but dies out for MWSDim and MWMDim after a couple of iterations. This decay is entirely a consequence of the communication concurrency and has nothing to do with the other mechanisms of idle wave decay, such as noise and communication inhomogeneity (see Section 5). The propagation of the "envelope wave" is untouched by this effect.

This phenomenon is shown in Figure 3(a1, b1, a2, b2), where the zig-zag pattern dissolves eventually, and the residual wave exhibits (a1)  $\kappa = \sum_{k=1,6} k = 7$ , (a2)  $\kappa = j = 6$ , (b1)  $\kappa = \sum_{k=1,6} k = 13$ , and (b2)  $\kappa = j = 12$ . The number of time steps required for the zig-zag to even out depends on the propagation speed. In case of a single MPI\_Waital1, however (a3, b3), the pattern prevails. The envelope travels with (a3)  $\kappa = j = 6$  and (b3)  $\kappa = j = 12$ .

The results from these microbenchmarks show that our model is able to describe the basic phenomenology of idle wave propagation on a silent system in the parameter space under consideration. In the following we cover some more general patterns.

**Inhomogeneous communication** From the basic propagation model and its validation on simple communication patterns we can now advance to more complex scenarios. In Figure 4, we use a compact topology matrix that is "fatter" for the middle 40 processes, mimicking an inhomogeneous situation that may, e.g., emerge with some sparse matrix problems (Figure 4(a)). Since the idle wave speed emerges from local properties of the topology matrix, we expect a "refraction effect," where the wave travels faster within

Fig. 5: Idle wave propagation within a double-precision 3D Jacobi algorithm with Cartesian domain decomposition and bidirectional halo exchange (15 iterations) at a problem size of  $1200^3$ and two different process grids (120 processes on Emmy) with open boundary conditions. Top row: topology matrices color-coded with communication volume. Bottom row: timelines of idle wave progression. Orange color shows idleness in MPI\_Wait, while pink color indicates waiting time in MPI\_Send. See text for communication grouping. Singlemessage communication volumes are (a) 576 kB, 480 kB, 384 kB and (b) 960 kB, 576 kB, 192 kB per dimension.



the fat region of the matrix. Indeed, this is exactly what is observed (see Figure 4(b)), and the quantitative model of propagation speed holds for the different regions: We have  $\kappa = 12$  in the middle and  $\kappa = 3$  elsewhere.

Blocking communication and eager vs. rendezvous mode Instead of grouped nonblocking point-to-point calls, a popular choice is MPI\_Sendrecv for a pair of in- and outgoing messages along the same direction. This is identical to the MWSDir case in Table 2, so the phenomenology shown in Figures 2 (a1, b1, c1) and Figures 3 (a1,b1) applies. Similarly one can employ a MPI\_Irecv/MPI\_Send/MPI\_Wait sequence within the innermost loop. In all these cases, the wave propagation speed doubles in rendezvous mode, where synchronization between sender and receiver is implied. However, the difference between eager and rendezvous mode does not impact the other variants beyond MWSDir.

Stencil smoother with halo exchange Figure 5 shows an idle wave experiment with a double-precision Jacobi smoother using Cartesian domain decomposition and two different process grids ( $4 \times 5 \times 6$  vs.  $2 \times 6 \times 10$ ; inner dimension goes first). Here we used MWSDir concurrency via MPI\_Irecv/MPI\_Send/MPI\_Wait per direction. The message sizes are such that the rendezvous mode applies. As expected from the model, the longest-distance communication determines the overall wave speed, i.e., it is lower in case (b) where the topology matrix is narrower.

The communication topology is more intricate here than in the microbenchmark studies covered so far. It turns out that all connections apart from the longest-distance one can be summarized by averaging over their respective distances and taking the largest smaller integer (floor function) when calculating the  $\kappa$  factor. For the case in

Fig. 6: Idle wave propagation in sparse matrix-vector multiplication (SpMV) using the HPCG matrix with a problem size of 16<sup>3</sup> per process and bidirectional halo exchange (15 iterations) on Emmy and three different process grids (a)-(c). Top row: topology matrices with color-coded communication volumes. Bottom row: Timelines of idle wave progression. Message sizes are 8B, 128 B, and 2.05 kB per dimension (symmetry across main diagonals).



Figure 5(a), this leads to  $\kappa = 2 + 20 = 22$ , so the propagation speed is  $22 \times 2 = 44$  times larger than  $v_{\text{silent}}^{\text{min}}$ . For Figure 5(b), we have  $\kappa = 0 + 12 = 12$  and thus 24 times  $v_{\text{silent}}^{\text{min}}$ . Both predictions are confirmed by the data after the initial slow, short-distance waves have died out.

**SpMVM with halo exchange** The High Performance Conjugate Gradient (HPCG) benchmark is popular for ranking supercomputers beyond the ubiquitous LINPACK. Here we choose to discuss idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system using a 27-point stencil in 3D. Communication is largely symmetric, except for boundaries. The number of communication partners varies between 7 (corners) and 26 (interior processes), and MWSDir concurrency applies just like in the stencil example. The per-process problem size is small enough for eager mode, but communication time is a relevant contribution to the overall runtime.

Figure 6 shows idle wave propagation through three different process grids with  $2 \times 4 \times 5 = 40$ ,  $4 \times 3 \times 5 = 60$ , and  $4 \times 5 \times 5 = 100$  ranks, respectively (inner dimension goes first). The decomposition is indicated in the captions of Figures 6(a)–(c). In case (a) we get  $\kappa = 8$ , for (b) we get  $\kappa = 12$ , and for (c) we get  $\kappa = 24$ .

### **4** Idle waves interacting with MPI collectives

Few MPI programs use point-to-point communications only. Concerning idle wave propagation, the question arises which collective routines may be transparent to a traveling wave. In practice, the elimination or the survival of the wave may be desirable



Fig. 7: Transparency of collective routines for idle waves on Emmy. (a) Default Intel MPI implementation of MPI\_Allreduce / MPI\_Alltoall / MPI\_Allgather / MPI\_Scatter / MPI\_Bcast / MPI\_Barrier / I\_MPI\_ADJUST\_REDUCE=1 / any collective with I\_MPI\_TUNING\_AUTO\_SYNC=1, (b) default MPI\_Reduce or with I\_MPI\_ADJUST\_REDUCE=8-11, (c) default MPI\_Gather / MPI\_Reduce with I\_MPI\_ADJUST\_REDUCE=2,4-7, (d) MPI\_Reduce with I\_MPI\_ADJUST\_REDUCE=3. Collective calls are injected at rank 5 in the 20th iteration and the root (where applicable) is rank 0. The message size is 1024 B, and MPI\_SUM is used for all operations. Green color indicates the time spent by MPI processes in the collective routines.

depending on the context; for instance, it was shown that idle waves can lead to automatic communication-computation overlap in desynchronized bottleneck-bound programs [3].

The effects we discuss here are certainly heavily dependent on the details of the MPI implementation, the communication buffer size, and possibly other parameters, so it is impossible to give a comprehensive overview. We thus restrict ourselves to Intel MPI on one of the three benchmark systems (Emmy). The results are summarized in Figure 7 and discussed below.

**Globally synchronizing primitives** Examples of necessarily synchronizing collectives are MPI\_Allreduce, MPI\_Alltoall, MPI\_Allgather, MPI\_Barrier, etc. These destroy propagating idle waves completely (see Figure 7(a)). The default Intel implementations of MPI\_Scatter and MPI\_Bcast are also synchronizing. If the *autotuner mode* is enabled by setting I\_MPI\_TUNING\_AUTO\_SYNC=1 (disabled by default), an internal barrier is called on every tuning iteration. This, of course, completely eradicates an idle wave on *any* collective call.

**Global non-synchronizing primitives** Figure 7(b) shows an idle wave colliding with the default Intel implementation of MPI\_Reduce. Reductions are not necessarily synchronizing, and indeed the idle wave can pass the collective, which appears like a global, compact communication block through which the wave travels with maximum speed (see the discussion of inhomogeneous communication above).

If the survival of idle waves is desirable, one option is to avoid synchronizing collectives if the performance implications are noncritical. In Figure 7(c), we show that the default MPI\_Gather implementation is completely transparent to the wave.

*Implementation variants* MPI implementations usually provide tuning knobs to optimize the internal implementation of collectives in order to better adapt it to the application. The process of finding the optimal parameter settings can also be automated [13]. With Intel MPI, the I\_MPI\_ADJUST\_<opname> environment variable can be set to a value that selects a particular implementation variant for the <opname> collective. Eleven documented settings are available in case of MPI\_Reduce. Figure 7(c), although it depicts a gather operation, is also applicable to MPI\_Reduce with I\_MPI\_ADJUST\_REDUCE set to 2 or a value between 4 and 7. Finally, Figure 7(d) illustrates how the interaction of the idle wave with MPI\_Reduce changes for I\_MPI\_ADJUST\_REDUCE set to 3 (topology-aware Shumilin's algorithm).

Another option is to override the default shared-memory node-level implementation of collectives and substitute it with a standard point-to-point variant. For instance, setting I\_MPI\_COLL\_INTRANODE=pt2pt (insted of the default shm) modifies the reduction behavior from Figure 7(b) to Figure 7(c).

# 5 Idle wave decay

The decay of traveling idle waves is a well-known phenomenon [10], and the underlying microscopic mechanism via interaction with short idle periods ("noise") is well understood [4]. There are, however, two questions that have not been addressed so far: (i) Does the system topology lead to idle wave decay also for resource-scalable parallel programs?, and (ii) Which characteristics of the system noise have an impact on the decay rate of the idle wave? Here answer both.

#### 5.1 Topological decay

It has been shown that the system topology, specifically a memory bandwidth bottleneck, can cause idle wave decay without the presence of system noise [3]. For the resource-scalable codes considered here this mechanism does not apply, but there is more to system topology than memory bottlenecks. The three benchmark systems we use here have quite different features in this respect, even within a single node: Hawk has 16 cores ( $4 \times 4$  CCX) per ccNUMA domain, 4 ccNUMA domains per socket, and 2 sockets per node. SuperMUC-NG has 24 cores per ccNUMA domain, 1 ccNUMA domain per socket, and 2 sockets per node. Emmy has 10 cores per ccNUMA domain, 1 ccNUMA domain per socket, and 2 sockets per node. The inherent topological boundaries cause communication inhomogeneities, which create structured noise as small variations in

14



Fig. 8: Topological idle wave decay on the benchmark systems running one process per core (scalable workload) using nonblocking MPI distance-1 communication topology (i.e.,  $P_i \rightleftharpoons P_{i\pm 1}$ ) for 120 iterations. We chose  $T_{\text{exec}} = 2.7 \text{ ms}$  (white color) and injected extra work of 58 ms (blue color) at rank 0. The message size was 1 MB. (a) 12 domains (sockets), 120 processes (b) 5 domains (sockets), 120 processes, (c) 30 domains (CCX), 120 processes. Topological boundaries exist at every 10, 24, and 4 cores on Emmy, SuperMUC-NG and Hawk, respectively.

communication time (intranode vs. internode) propagate and interact with the idle wave to cause visible kinks. This is demonstrated in Figure 8 for the three benchmark clusters, running one MPI process per ccNUMA domain. For 120 iterations, we measured an average decay rate of  $149 \,\mu\text{s}/\text{rank}$  on SuperMUC-NG,  $203 \,\mu\text{s}/\text{rank}$  on Hawk, and  $346 \,\mu\text{s}/\text{rank}$  on Emmy. Although one might expect Hawk to show the strongest topology effects due to its intricate node structure, it is not only the number of hierarchy levels but also the actual communication inhomogeneity that determines the decay effect. In Figure 8, all 128 processes were run on a single node of Hawk, so the internode boundary is missing there.

In order to substantiate the claim that this decay emerges from system topology and communication inhomogeneities, we repeated all experiments with *round-robin placement* of MPI ranks across nodes. In this way, node-level differences in communication characteristics are all but eliminated since all interprocess boundaries are internode boundaries. Indeed, the decay observed with standard placement vanishes under these conditions.



Fig. 9: Experiment comparing the average decay rate of an idle wave (initial duration 4850 ms) for two different noise characteristics (top vs. bottom). In both cases, the integrated noise power is 9.1% of the total area below the idle wave, i.e., 13 s of 142 s, but the distribution of the fine-grained noise is different. However, the overall average decay rate is the same (480 ms/rank), as is the wave survival time (34 s).

Fig. 10: Decay rate (min/max/median at sixteen crossprocess transitions) of an idle period in s/rank, comparing three different noise patterns (see [4]) on the InfiniBand Emmy (18 processes, one per node, single leaf switch). The x-axis shows integrated noise power with respect to overall integrated runtime of 142 s.



#### 5.2 Noise-induced decay

For the purpose of this work, we define "noise" as any (per-process) deviation from a fixed, repeatable, lockstep-type compute-communicate pattern. In this sense, strong one-off delays are also noise, but in this section we specifically consider noise that is considerably more fine grained. One of the unsolved questions in previous work about idle wave decay, specifically with resource-scalable code, is whether the detailed statistical properties of the fine-grained noise or just the integrated noise power impact the rate of decay. In order to exert full control over all noise characteristics, we conduct experiments with artificial noise injections that are orders of magnitude stronger than natural noise. Due to the fundamental scale invariance of these mechanisms, the conclusions must also hold for realistic scenarios. How idle waves interact with each other in a nonlinear way has been analyzed in previous work [4]; noise-induced decay is just a variant of this process. Noise "eats away" at the trailing edge of the wave, so a small idle period (i.e., a part of the noise) of duration  $T_{\text{noise}}$  that collides with the idle wave shortens the latter by an amount of exactly  $T_{\text{noise}}$ . This process is cumulative, which leads to the immediate conclusion that multiple interactions  $\{T_{\text{noise}}^i\}$  diminish the idle wave by  $\eta = \sum_i T_{\text{noise}}^i$ . Noise statistics is of minor importance for the average decay rate. It will only impact the "smoothness" of the decay. Figure 9 illustrates this fact by comparing the decay of the same idle wave under two widely different noise characteristics with identical integrated "noise power"  $\eta$ . Although the details of the decay are different, the survival time and hence the average decay rate of the wave is the same in both cases. This holds as long as the noise is fine-grained enough to not annihilate the idle wave in one fell swoop at an early stage. Note that previous research [9, 5, 6] only studied the influence of noise statistics on application and global operations scalability. Our observable is idle wave decay rate, which is largely robust against noise statistics.

**Experimental validation** To better validate this hypothesis, we measured the decay rate of an idle wave under three different noise characteristics with the same noise power. Figure 10 shows results for 18 processes (one per node) on one leaf switch of Emmy to rule out topological effects. Apart from this detail, the setting is similar to Figure 9. The microscopic shape of the decay is influenced by the statistics: Shot noise, i.e., random but strong, sparse noise injections of a single duration, lead to discontinuous decay and strong variations in decay rate (diamonds in Figure 10). On the other hand, exponential (squares) and uniform (circles) noise characteristics, where noise injections show a whole spectrum of durations, and the variation in decay rates is much weaker. The median of measured decay rates, however, only depends on the noise power.

## 6 Summary and future work

We have presented an analytical model of idle wave propagation speed based on communication topology and concurrency characteristics of resource-scalable MPI programs. The model was validated against simple microbenchmarks, a 3D stencil smoother, and sparse matrix-vector multiplication with the HPCG matrix. We have also shown that MPI collective routines can be transparent to idle waves depending on the type and implementation of the collective, which extends the relevance of idle wave phenomena beyond bulk-synchronous algorithms without collective communication. In light of the fact that the presence of idle waves is not necessarily detrimental for performance, this result can be quite relevant to the performance analysis of highly scalable codes. Furthermore, we have uncovered the relevance of system topology for idle wave decay: The presence of inhomogeneous communication characteristics emerging from the hierarchical structure of modern compute nodes leads to fine-grained noise that causes the decay of idle waves. Finally, we have shown that it is the noise power, and not its detailed statistical properties, that govern the noise-induced decay rate. All these findings contribute significantly to the understanding of the idle wave phenomenon on multicore clusters. Future work will include the extension of the analysis to programs that are not resource scalable, i.e., that are limited by node-level or network-level bottlenecks. There is also the open question which wave and noise phenomena can be described by effective models that abstract away from the details of the cluster hardware. Finally, we will develop a capable MPI simulation tool that can take node-level characteristics into account and will allow for more extensive experimental studies and architectural exploration.

## Acknowledgments

This work was supported by KONWIHR, the Bavarian Competence Network for Scientific High Performance Computing in Bavaria, under project name "OMI4papps," and by the BMBF under projects "Metacca" and "SeASiTe." We are indebted to LRZ Garching and to HLRS Stuttgart for granting CPU hours on their "SuperMUC-NG" and "Hawk" systems.

## References

- [1] A. Afzal, G. Hager, and G. Wellein. An analytic performance model for overlapping execution of memory-bound loop kernels on multicore CPUs. In *arXiv*, 2020. arXiv: 2011.00243 [cs.DC]. Submitted.
- [2] A. Afzal, G. Hager, and G. Wellein. Delay flow mechanisms on clusters. URL: https://hpc.fau.de/files/2019/09/EuroMPI2019\_AHW-Poster.pdf. Poster at EuroMPI 2019, September 10–13, 2019, Zurich, Switzerland.
- [3] A. Afzal, G. Hager, and G. Wellein. Desynchronization and wave pattern formation in MPI-parallel and hybrid memory-bound programs. In P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, editors, *High Performance Computing*, pages 391–411, Cham. Springer International Publishing, 2020. ISBN: 978-3-030-50743-5. DOI: 10.1007/978-3-030-50743-5\_20.
- [4] A. Afzal, G. Hager, and G. Wellein. Propagation and decay of injected one-off delays on clusters: A case study. In 2019 IEEE International Conference on Cluster Computing, CLUSTER 2019, Albuquerque, NM, USA, September 23-26, 2019, pages 1–10, 2019. DOI: 10.1109/CLUSTER.2019.8890995.
- [5] S. Agarwal, R. Garg, and N. K. Vishnoi. The impact of noise on the scaling of collectives: A theoretical approach. In *International Conference on High-Performance Computing*, pages 280–289. Springer, 2005.
- [6] K. B. Ferreira, P. Bridges, and R. Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008* ACM/IEEE conference on Supercomputing, page 19. IEEE Press, 2008.
- [7] M. Gamell et al. Local recovery and failure masking for stencil-based applications at extreme scales. In SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–12, Nov. 2015. DOI: 10.1145/2807591.2807672.

- [8] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim Simulating Large-Scale Applications in the LogGOPS Model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604, Chicago, Illinois. ACM, June 2010. ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851564.
- [9] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the* 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–11. IEEE Computer Society, 2010.
- S. Markidis et al. Idle waves in high-performance computing. *Physical Review E*, 91(1):013306, 2015. DOI: 10.1103/PhysRevE.91.013306.
- [11] A. Nataraj et al. The ghost in the machine: observing the effects of kernel operation on parallel application performance. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, 2007.
- [12] I. B. Peng et al. Idle period propagation in message-passing applications. In High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on, pages 937–944. IEEE, 2016.
- [13] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. Automatically tuned collective communications. In SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, pages 3–3. IEEE, 2000.