Dear Reviewer,

We would like to sincerely thank you for your thorough and constructive review of our manuscript (An Effective Communication Topology for Performance Optimization: A Case Study of the Finite Volume WAve Modeling (FVWAM)). Your insightful comments have been invaluable in improving the quality of our work. Please find below our detailed responses to each of the comments you raised.

Sincerely,

Renbo PANG, on behalf of the co-authors

This paper describes implementation and performance benchmarks of neighborhood exchanges in the FVWAM model. It compares the performance of a standard implementation of halo exchanges based on point-to-point communication with the performance of an implementation based on MPI distributed graph topology interface and neighborhood collectives. Using the distributed graph topology interface, the authors obtained a maximum communication time speed-up of 5.63 and 40.2% reduction in the total FVWAM run time with 1,024 processes. This is an interesting and significant result, especially because the use of the *distributed graph topology interface and neighborhood collectives is not common in earth system models. I believe the article is suitable for GMD after my comments are addressed.*

Reply： Thank you very much for your positive comments!

Specific comments 1: *In section 2.2 the authors describe potential benefits of using the MPI distributed topology interface and present its ability to optimize process mappings as its main advantage. However, the benchmark results of FVWAM show that, while using this interface provides consistent speedups over the point-to-point implementation, setting the reorder flag has only minor performance impacts. What is then the main reason for the observed speedups? Can section 2.2 be expanded to discuss other potential performance benefits ?*

Reply： The original process order is determined by the partitioning scheme of METIS, as described in Lines 150-157. METIS optimizes the process ordering by placing neighboring process IDs together. The MPI implementation then allocates processes across computing nodes according to the ascending order of these process IDs. As a result, the communication performance shows only minor improvements when the reorder flag is set. However, if we were to manually or randomly arrange the process order instead of using the METIS partitioning result, we believe that the communication performance would improve significantly when the reorder flag is set.

Multiple calls to the point-to-point interface for exchanging data among different processes can lead to unfavorable side effects, such as load imbalance and unpredictable wait times during connection establishment (Torsten et al., 2002). The MPI distributed topology interface optimizes connection management (Torsten et al., 2002). This optimization is independent of whether the reorder flag is set to true or false. We have included this additional information in Section 2.2 to further clarify the benefits of using the MPI distributed topology interface.

Specific comments 2: *The presentation of the distributed graph topology workflow in section 3.1 could be improved. The first three paragraphs, related to Figure 3, describe the process of creating MPI graph topology starting from domain partitioning. Most of this material is then repeated in subsequent paragraphs, which pertain to Figure 4. If the authors' intention was to first present the workflow at a high level and then go into details specific to FVWAM this needs to be clearly stated and better organized to remove some of the repetition.*

Reply： Thank you for your valuable recommendation to improve Section 3.1! The first three paragraphs (Lines 128-148), which relate to Figure 3, have been summarized at a higher level as follows:

The workflow to create a distributed graph communication topology based on SCVT cells is shown in Figure 3. Initially, the global SCVT cells are partitioned according to the number of computing processes. A simple partitioning result of the global SCVT sea cells into three partitions is illustrated in Figure 3(a), with each partition colored green, blue, and purple, respectively.

Next, each process determines its receiving processes and cells based on the partitioning result in Figure 3(a) and the neighboring cell and process information in Figure 3(b). The red line denotes the boundary separating the cells allocated to Processes P_0 and P_1, while the orange line delineates the communication boundary for Process P_0. The cells situated between the red local cell boundary line and the orange communication boundary line comprise the receiving cells for Process P_0.

Finally, a distributed graph communication topology is created by calling the MPI interface with the sending and receiving process IDs and their respective degrees, as shown in Figure 3(c). The sending degree corresponds to the total number of sending processes, and the receiving degree represents the total number of receiving processes.

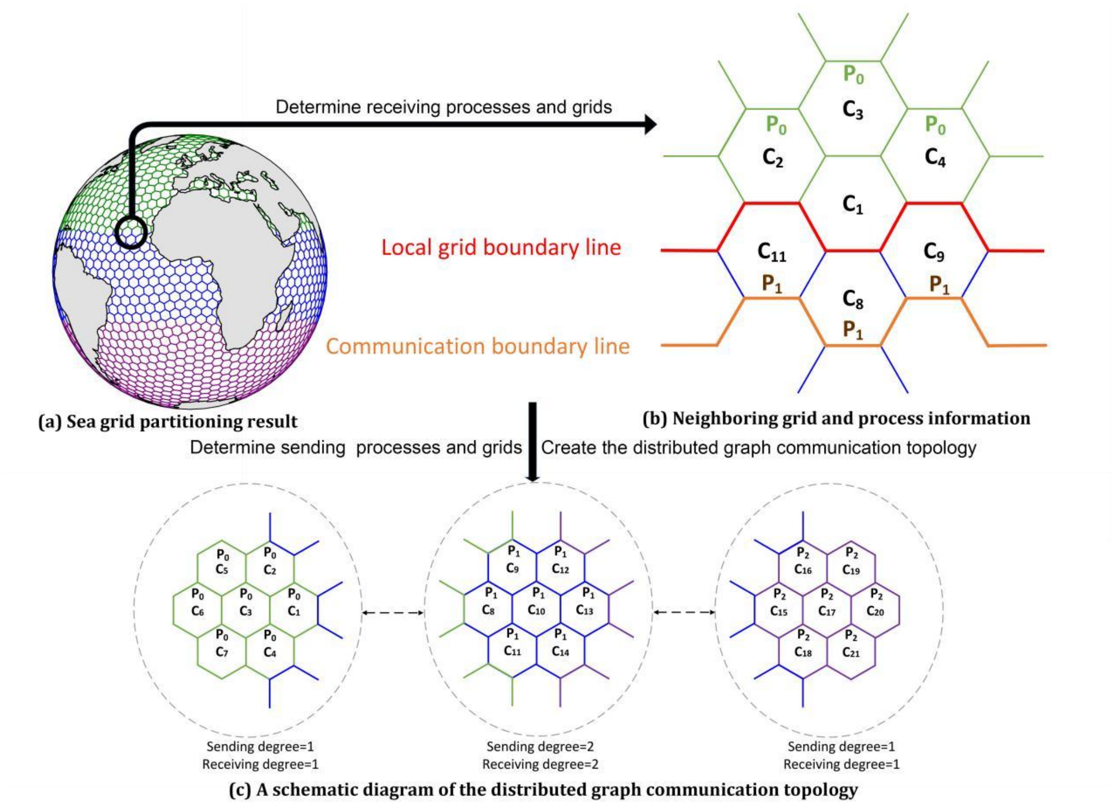
Determine receiving processes and grids
P0 C3
P0 C2
P0 C4
C1
Local grid boundary line
C11 P1
C9 P1
C8 P1
Communication boundary line
(a) Sea grid partitioning result
(b) Neighboring grid and process information
Determine sending processes and grids
Create the distributed graph communication topology
P0 C5
P0 C2
P0 C6
P0 C3
P0 C1
P0 C7
P0 C4
P1 C9
P1 C12
P1 C8
P1 C10
P1 C13
P1 C11
P1 C14
P2 C16
P2 C19
P2 C15
P2 C17
P2 C20
P2 C18
P2 C21
Sending degree=1
Receiving degree=1
Sending degree=2
Receiving degree=2
Sending degree=1
Receiving degree=1
(c) A schematic diagram of the distributed graph communication topology

Figure 3. The workflow to create a distributed graph communication topology

Specific comments 3: *Are the results of both communication mechanisms bit-for-bit identical ? How was the correctness of the implementation verified ?*

Reply: Yes, the results of both communication mechanisms are bit-for-bit identical. To verify this, we compared key variables of significant wave height, wave period, and wave direction in the output files generated by both mechanisms. No differences were observed, confirming the correctness of the implementation. We have included this verification of correctness between the point-to-point communication interface and the distributed graph topology interface in “Section 4.4 the operational products of FVWAM”.

Specific comments 4: *All of the paper performance results were obtained using Intel MPI on one computing system. I imagine that the performance of a high-level interface like the distributed graph topology can strongly depend on the quality of implementation of the underlying MPI library. At minimum, this should be discussed, but showing results using a different MPI implementation would be a great addition to the paper. Do the authors expect that their results would generalize to other platforms ?*

Reply： Thank you for your valuable recommendation to compare different communication methods across multiple MPI libraries. Due to the expiration of our rental contract for the high-performance computing system at the National Supercomputing Center of China in Jinan, we are currently unable to conduct additional experiments at the same scale (32,678 CPU cores) with different MPI implementations. However, we conducted smaller-scale tests in the West Pacific region using both Intel

MPI Library and Open MPI Library on a different platform. The results indicate that the performance of the distributed graph topology is indeed strongly dependent on the quality of the underlying MPI library implementation.

The software and hardware environment for the first set of tests is presented in Table 1.

Tab.1 Software and hardware environment

Name	Version
CPU	Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz (28 cores per node)
Memory	128GB
Hardware Architecture	X86_64
Network	Infiniband (100Gb/s)
Operating System	Red Hat Enterprise 7.6
Compiler	Ifort 17.0.3
Compilation Options	-O3
MPI	Intel(R) MPI Library 2017.3.191
NetCDF	NetCDF-Fortran 4.5.3

The cell resolution is 6-12 km, covering the region from 95° E to 145° E and 0° N to 40° N. The number of horizontal cells is 283,517, the count of the directional spectrum is 36, and the count of the frequency spectrum is 35. The time step of iterative computation in the test was 60 seconds, and the forecasting period was one hour. Each iteration involved a single neighboring communication for a 3D variable of wave action *N*. The total times of neighboring communication for *N* during the test was 60.

We performed a series of tests on the FVWAM using different numbers of computing processes, ranging from 8 to 512 (28 processes per node), to evaluate and compare the efficiency of the point-to-point communication method versus the distributed graph communication topology in the Intel MPI Library, as shown in Figure 10. For intra-node communication with 8 and 16 processes, the performance of both communication methods was similar. However, for inter-node communication, the distributed graph communication topology significantly outperformed the point-to-point method.

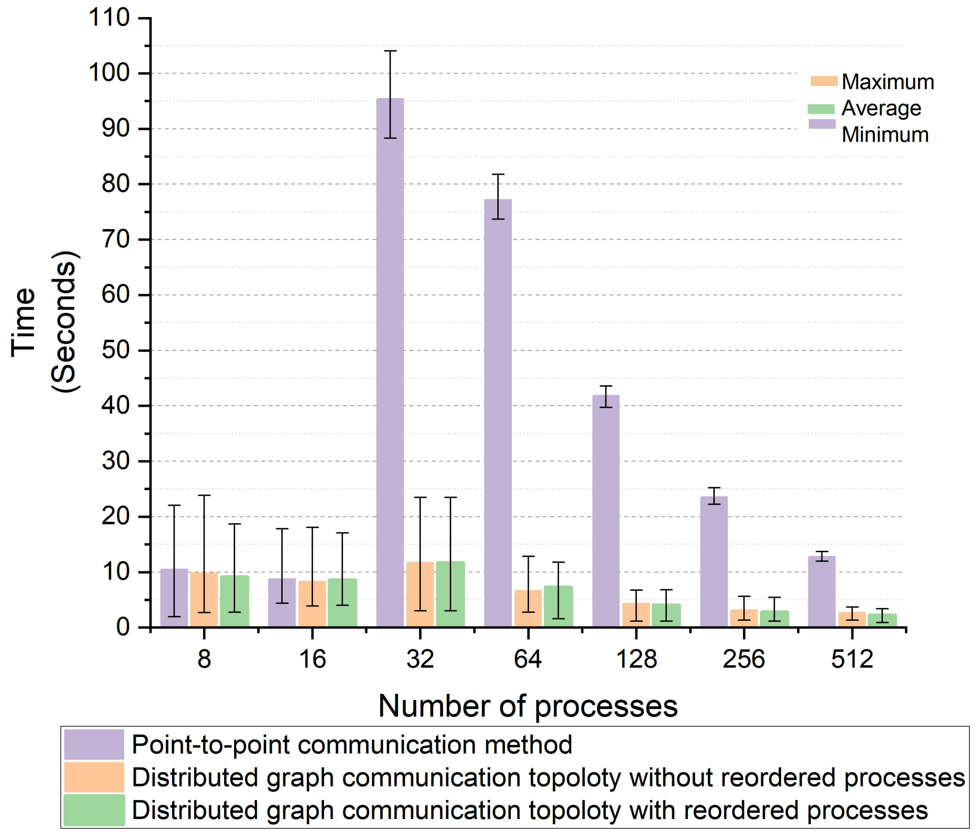

Figure 10. Time of neighborhood communication in the Intel MPI Library

The software and hardware environment for the second set of tests is presented in Table 2.

Tab.2 Software and hardware environment

Name	Version
CPU	Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz (28 cores per node)
Memory	128GB
Hardware Architecture	X86_64
Network	Infiniband (100Gb/s)
Operating System	Red Hat Enterprise 7.6
Compiler	GNU Fortran 10.2.0
Compilation Options	-O3
MPI	Open MPI 4.0.5
NetCDF	NetCDF-Fortran 4.5.3

The model configuration in this test is the same as the first test. The results of the FVWAM using different numbers of computing processes, ranging from 8 to 512 (28 processes per node), are shown in Figure 11 to evaluate and compare the efficiency of the point-to-point communication method versus the distributed graph communication topology in the Open MPI Library. The performance gap between the two methods was smaller, and there was no noticeable performance improvement in intra-node communication (with 8 or 16 processes) when using the Open MPI Library, compared to the Intel MPI Library.

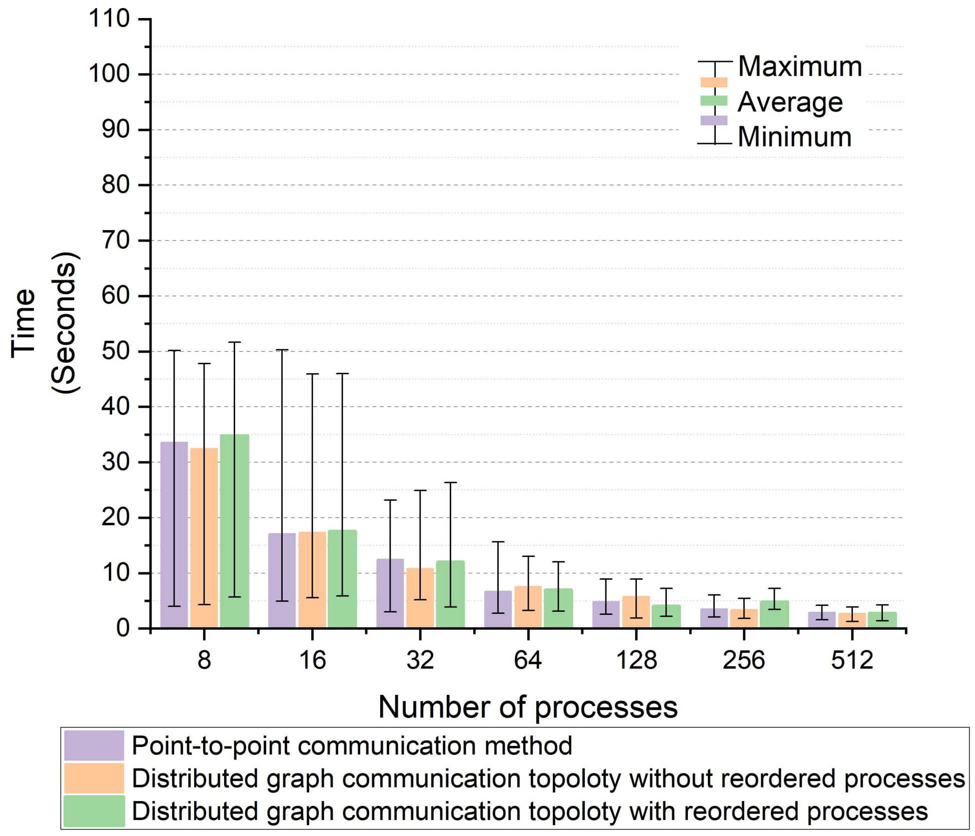

Figure 11. Time of neighborhood communication in the Open MPI Library

Minor comments 1: *Throughout the paper, the authors refer to cells and cell indices as grids and grid IDs. This terminology is very non-standard and can be confusing. I strongly suggest replacing "grids" with "cells" and either replacing "grid IDs" with "cell IDs" or adding a sentence that in this paper "grid IDs" mean cell IDs.*

Reply: We have replaced "grid" with "cell" which can be confusing , and "grid IDs" have been changed to "cell IDs" throughout the paper.

Minor comments 2: *Line 156: the variable "cellsOnCell" has already been introduced on line 134, where it is spelled "CellsonCell".*

Reply: The introduction of the variable "cellsOnCell" on line 156 has been removed.

Minor comments 3: *Line 181: "MPI_DIST_GRAPH_CREATE_ADJACENT" - why are some MPI function names written in all-caps and some are not ? I suggest using the C interface names consistently throughout the paper.*

Reply: We have replaced "MPI_DIST_GRAPH_CREATE_ADJACENT" with "MPI_Dist_graph_create_adjacent" to ensure consistency with the C interface naming convention. All MPI function names in the paper have been updated to use the correct C interface names.

Minor comments 4: *Lines 212-213: "MPI_Isend (. . .) is infrequently utilized . . . ". Can the authors back-up this claim ? All of the models I worked on used "MPI_Isend".*

Reply: "MPI_Isend (. . .) is infrequently utilized . . . " on Lines 212-213 has been removed.

Minor comments 5:*Table 1: Change "Compiling Option" to "Compilation Options".*

Reply: The term "Compiling Option" has been changed to "Compilation Options" in the paper.

Reference:

Hoefler T, Rabenseifner R, Ritzdorf H, et al. The scalable process topology interface of MPI 2.2[J]. Concurrency and Computation: Practice and Experience, 2011, 23(4): 293-310.