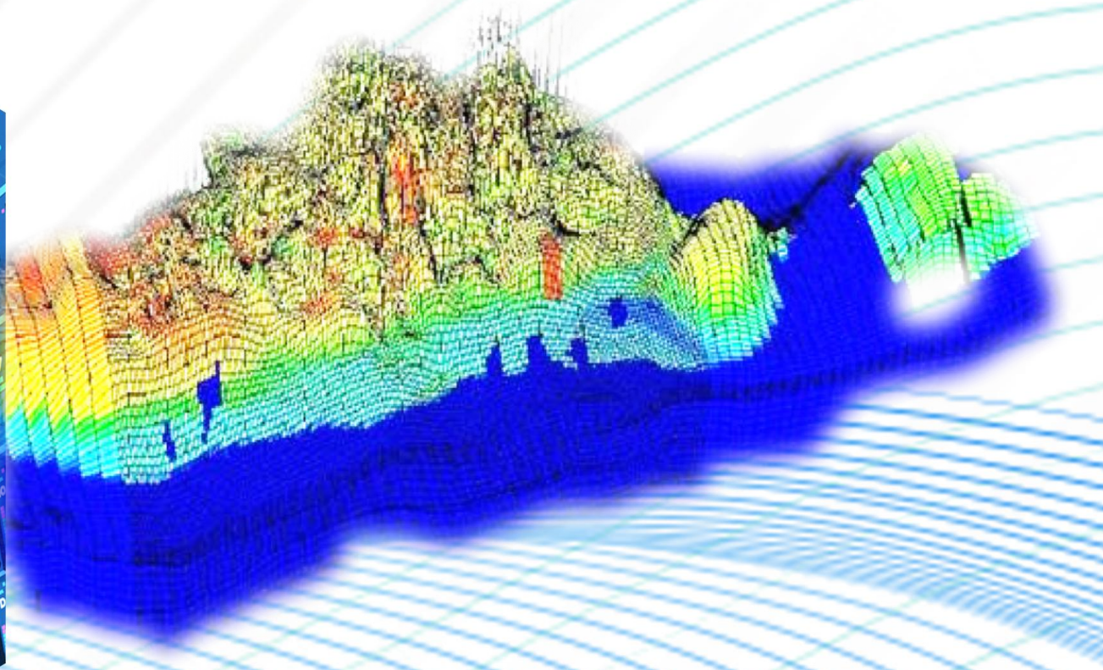
MPI Communication Optimization of Massively Parallel Applications

Fatimah Al-Ruwai & Majdi Baddourah
Saudi Aramco, Dhahran 31311, Saudi Arabia

ارامكو السعودية
Saudi Aramco
HPC Asia 2018



Introduction

In reservoir simulation, high resolution models have become the norm to model and capture the detailed characteristics of fluids flow in hydrocarbon bearing reservoir. This leads to enhanced understanding of the physics and accuracy of the simulation results. Such detailed reservoir models require huge compute capacity to run efficiently with acceptable runtime. Our simulation models run on thousands of cores for several hours. In order to address such challenge, Saudi Aramco has been acquiring and utilizing the best in HPC clustering technologies for its parallel reservoir simulation studies. As shown in Figure 1, the demand for simulation studies is increasing in Saudi Aramco as huge detailed simulation models are being built to gain insight into oil and gas reservoirs and to forecast reservoir performance in an accurate manner. Figure 2 shows how reservoir simulation computing resources are also increasing rapidly to meet the growing simulation demands in Saudi Aramco over the past few years.

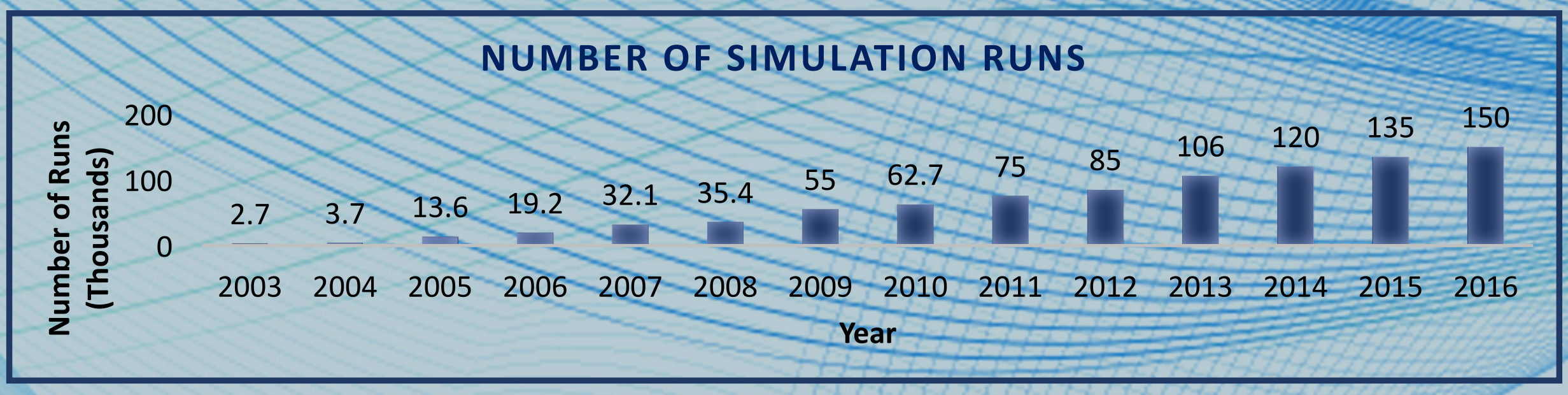


Figure 1 Saudi Aramco Growth in Number of Simulation Jobs

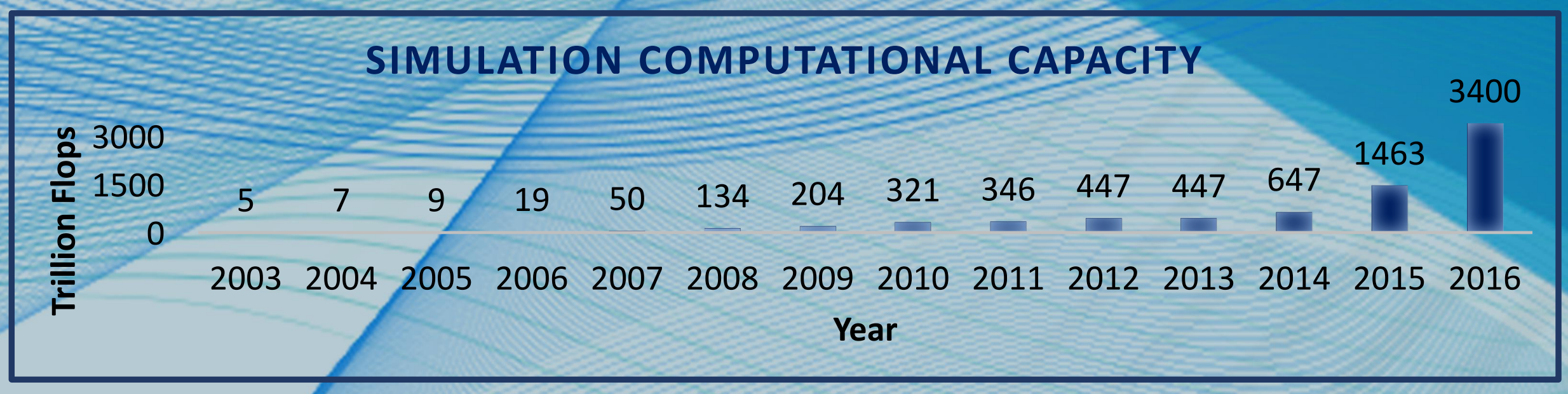
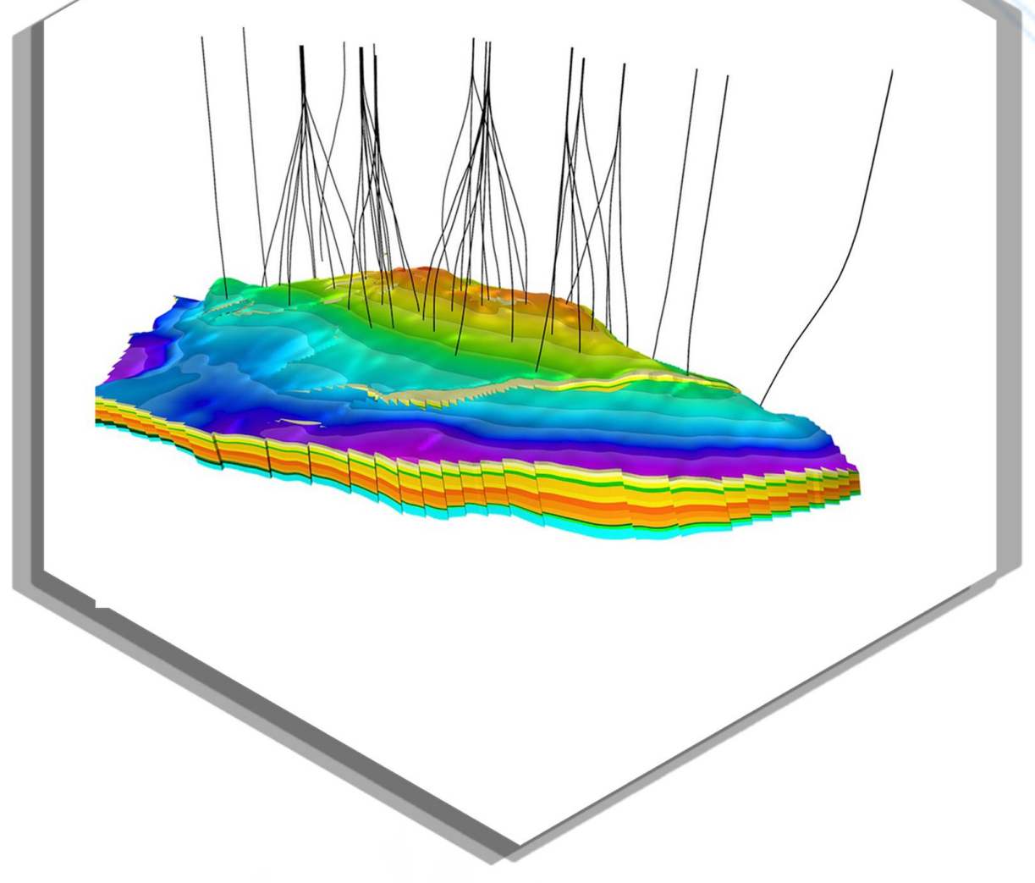


Figure 2 Saudi Aramco Growth in Simulation Computational Capacity

Besides that, Saudi Aramco has developed an in-house parallel reservoir simulator which has the capability of simulating huge Saudi Aramco reservoir models. Massively parallel scientific and engineering applications exhibit MPI communication overhead that can be abridged to improve the applications runtime and scalability. The work presented will provide the process used in identifying communication hotspots by profiling and analyzing various simulation models, and the procedure on how the optimization was undertaken to overcome communication bottlenecks.



Methods

Profiling:

The optimization process began by establishing performance baseline of different simulation models with various model sizes and runtimes. Software hotspot profiling and analysis was then performed over the selected cases using Intel® VTune™ profiler which was used to analyze, profile and help with decisions on MPI communication fine-tuning. Intel MPI was the main stream in our research area.Table 1 shows the profiling and analysis results which indicates that at least 30% of the simulator's time is spent on MPI communication. Over the four different cases, the trends indicate that MPI communication time can reach up to 44% with of the wall-clock time.

Table 1: MPI communication Profiling and Analysis for Four (4) Simulation Models

Case Name	No. of Cores	Wall-Clock Time	MPI Time	MPI Time (%)
CASE 1	500	3 H 28 M 25 S	1 H 8 M 18 S	32.78%
CASE 2	500	1 H 17 M 52 S	29 M 44 S	38.20%
CASE 3	490	3 H 12 M 33 S	1 H 25 M 52 S	44.59%
CASE 4	500	6 H 9 M 31 S	2 H 4 M 40 S	33.74%

Based on the above analysis, Intel® MPS profiler was used to perform further investigation and conduct in depth profiling of the MPI communication. It was observed that MPI Barrier collective is one of the major bottlenecks as it is the case with many other MPI applications exist in the industry. Table 2 depicts the impact of the MPI Barrier calls on the communication; the overhead of the calls varies from 14% to 44% of the communication time. The data also shows that the number of MPI Barrier calls can reach to more than one billion calls (varies from one model to another).

Table 2: Illustration of MPI Barrier Calls for Four (4) Simulation Models

Case Name	No. of Cores	Time (sec)	Time (%)	Calls
CASE 1	500	303303.55	14.80	421,693,313
CASE 2	500	260468.63	29.19	761,836,748
CASE 3	490	1125990.74	44.60	286,070,005
CASE 4	500	1315744.11	35.18	1,514,059,234

First Optimization Technique:

Thus, the main objective is to reduce the MPI Barrier calls as much as possible without changing the simulation results. The methodology of removing the MPI synchronization calls was by identifying unnecessary calls within the code. Unnecessary calls are usually those calls that were added for debugging purposes. So, whenever developers have a new portion of code that has any MPI collective, they tend to place an MPI Barrier call in place of that collective just to test their progress within the code. If results are successful, they add the MPI collective just after the MPI Barrier without removing the Barrier call. So, it is a development habit that exists in many applications nowadays.

Second Optimization Technique:

After the first optimization which is related to removing MPI Barrier unnecessary calls, MPS profiler was used again to ensure proper removal of MPI Barriers. MPS has shown almost no MPI Barrier calls in the majority of the test cases. Therefore, another optimization technique was addressed to see if MPI communication can be further improved by using Intel MPI collective algorithms. Intel has different number of built in algorithms for each collective operation. So, four highly used collective operations by the reservoir simulator were tested against the different algorithms by using Intel balance benchmark. Below figures show the best algorithm for each collective operation with respect to the number of Bytes when tested using Intel balance benchmark. So, this benchmark gives an indication of the best algorithm selection for each number of byte to be used in the reservoir simulator application. All algorithms were executed for each collective operation in the same set of nodes with a total of 500 cores; this is to ensure fair comparison between all algorithms when having exact nodes in all runs.

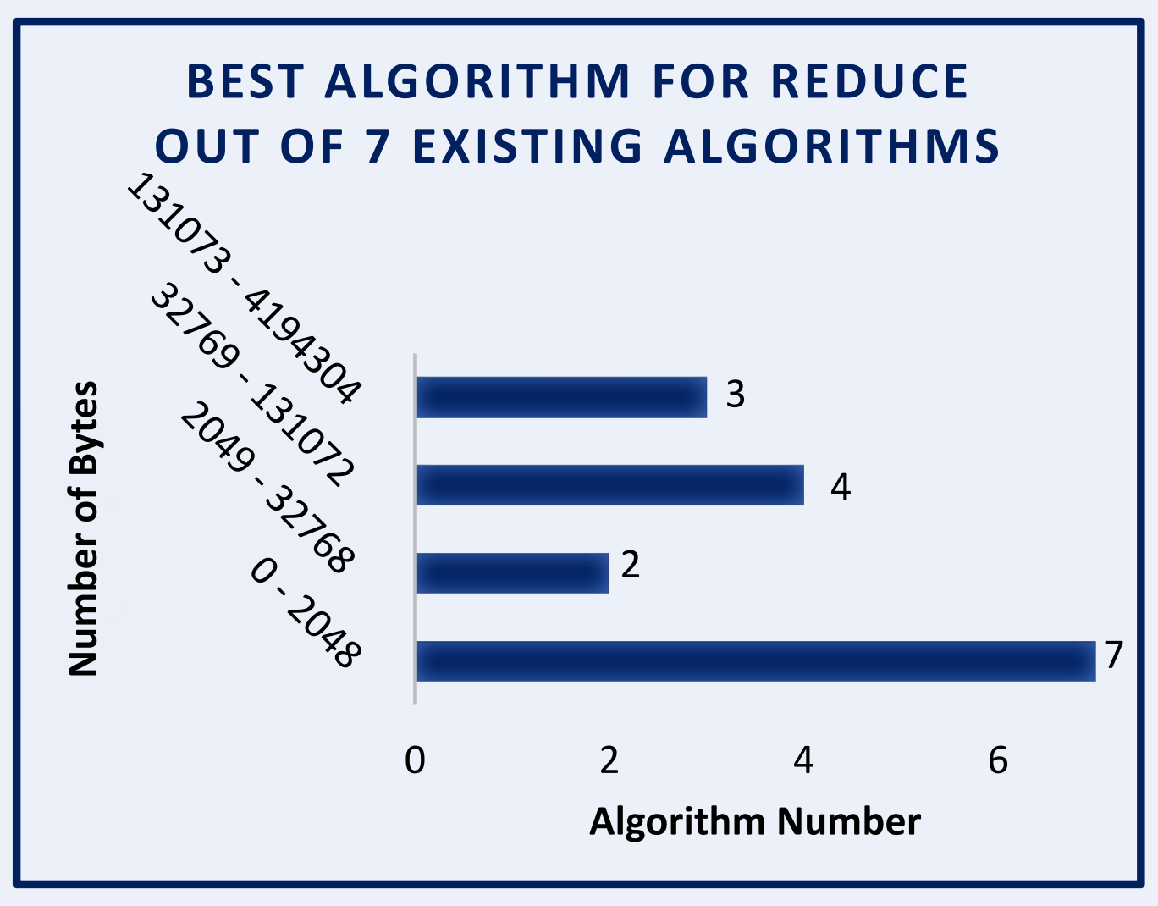


Figure 3 Best Algorithm Selection for Reduce

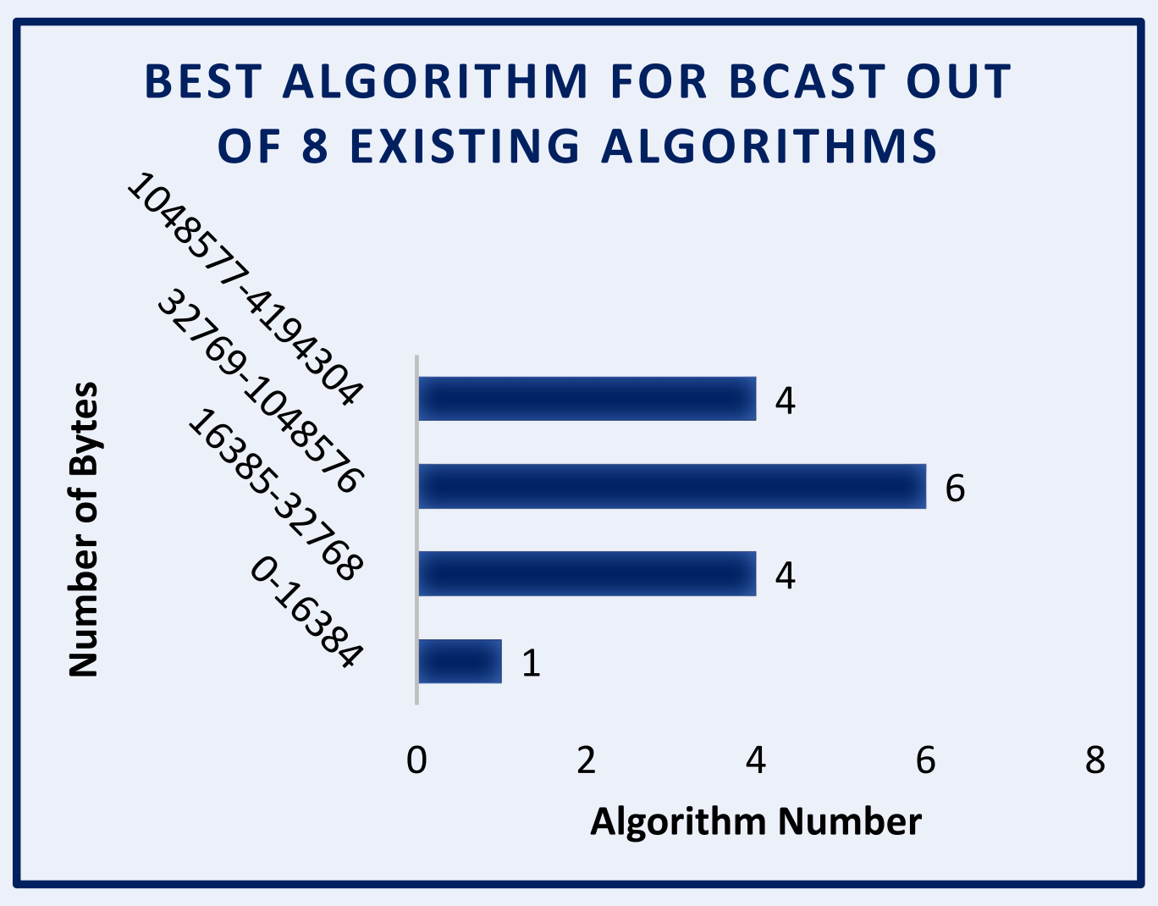


Figure 4 Best Algorithm Selection for Bcast

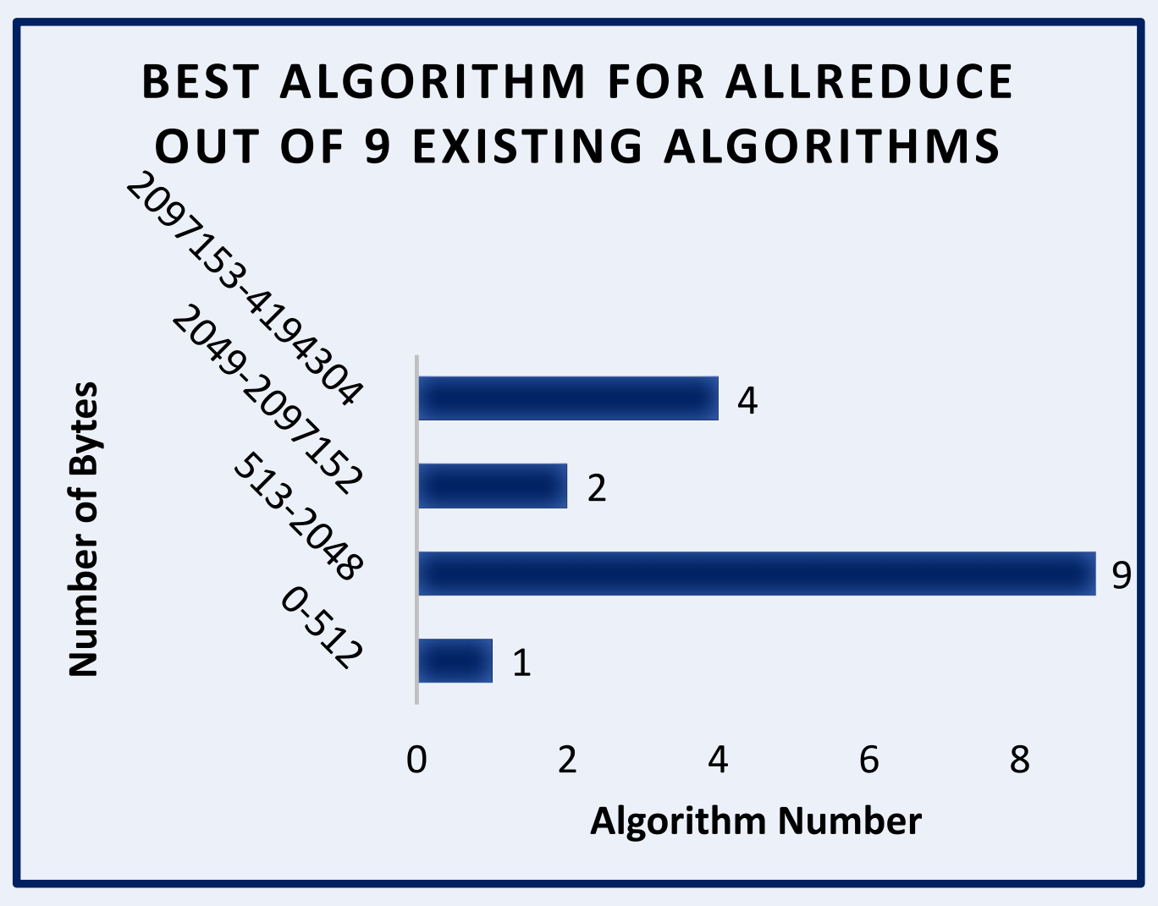


Figure 5 Best Algorithm Selection for Allreduce

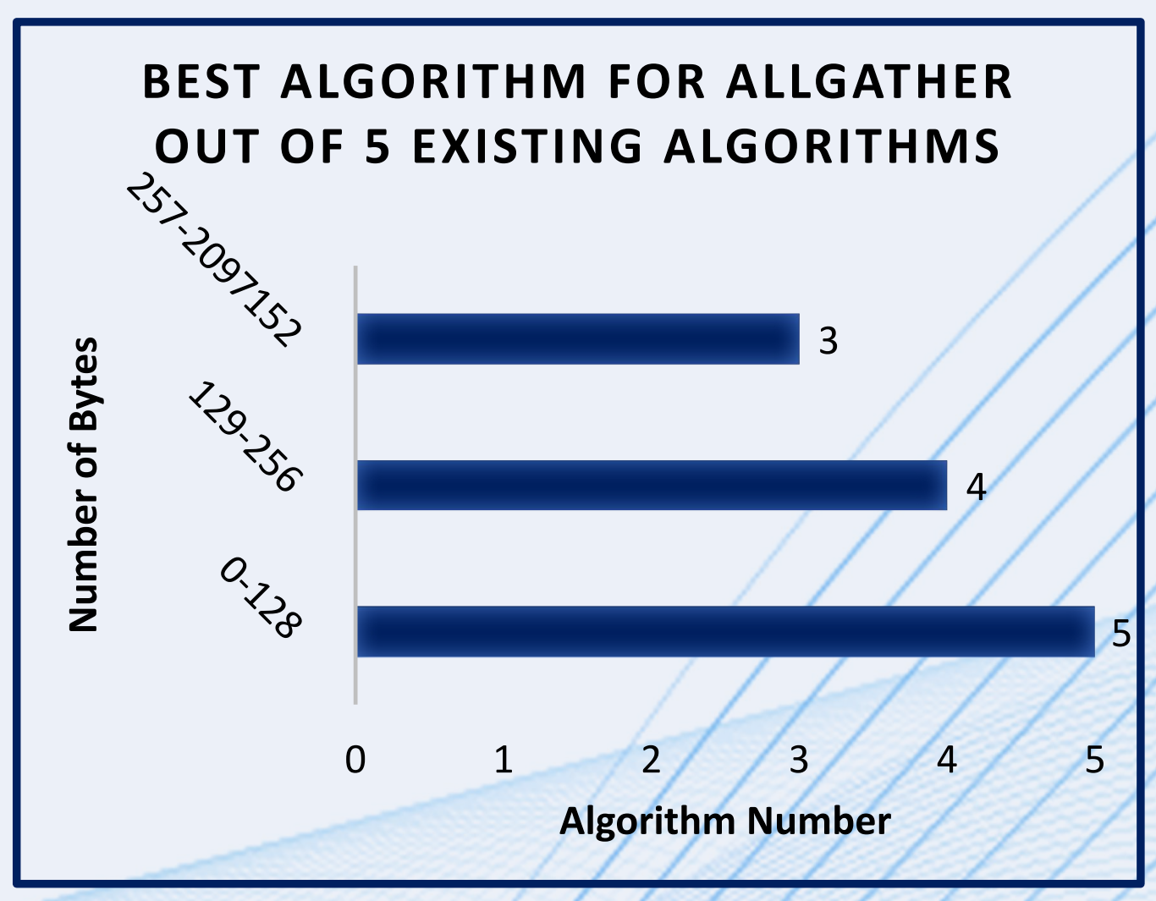


Figure 6 Best Algorithm Selection for Allgather

Results

First Optimization Outcomes:

Toward an optimal optimization, unnecessary MPI Barrier calls were targeted for removal. Majority of MPI Barrier calls were removed within several Fortran files. The percentage usage of MPI Barrier calls before optimization varies from 14.80% to 44.60%. After optimization communication overhead coming from MPI Barrier calls was reduce to almost 0% in all simulation cases. Table 3 shows detailed information of the non-optimized case verses the optimized one.

Table 3: Illustration of MPI Barrier Calls in the Original vs. Optimized Cases

Case Name	Executable	No. of Cores	Time (sec)	Time (%)	Calls
CASE 1	Original	500	303303.55	14.80	421693313
	Optimized		0	0	0
CASE 2	Original	500	260468.63	29.19	761836748
	Optimized		0	0	0
CASE 3	Original	490	1125990.74	44.60	286070005
	Optimized		0	0	0
CASE 4	Original	500	1315744.11	35.18	1514059234
	Optimized		0	0	0

Such optimization has shown dramatic speed-up. Figure 5 shows the speed-up of each case after the first optimization with respect to the wall clock time. It declares a gain of 2% to 4% speedup in the overall run time. Figure 6 illustrates the speed-up gained with respect to the MPI time. It demonstrates a speedup of 5% to 8% in the communication time.

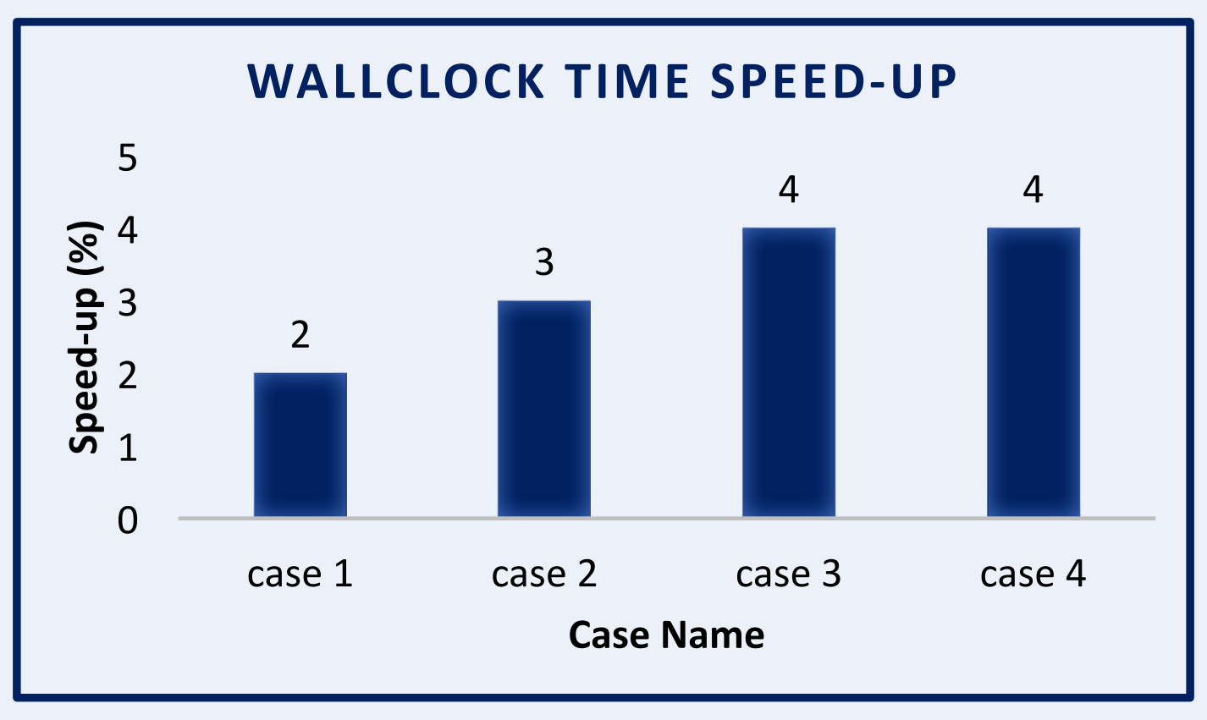


Figure 7 Illustration of Overall Run Time Speed-up

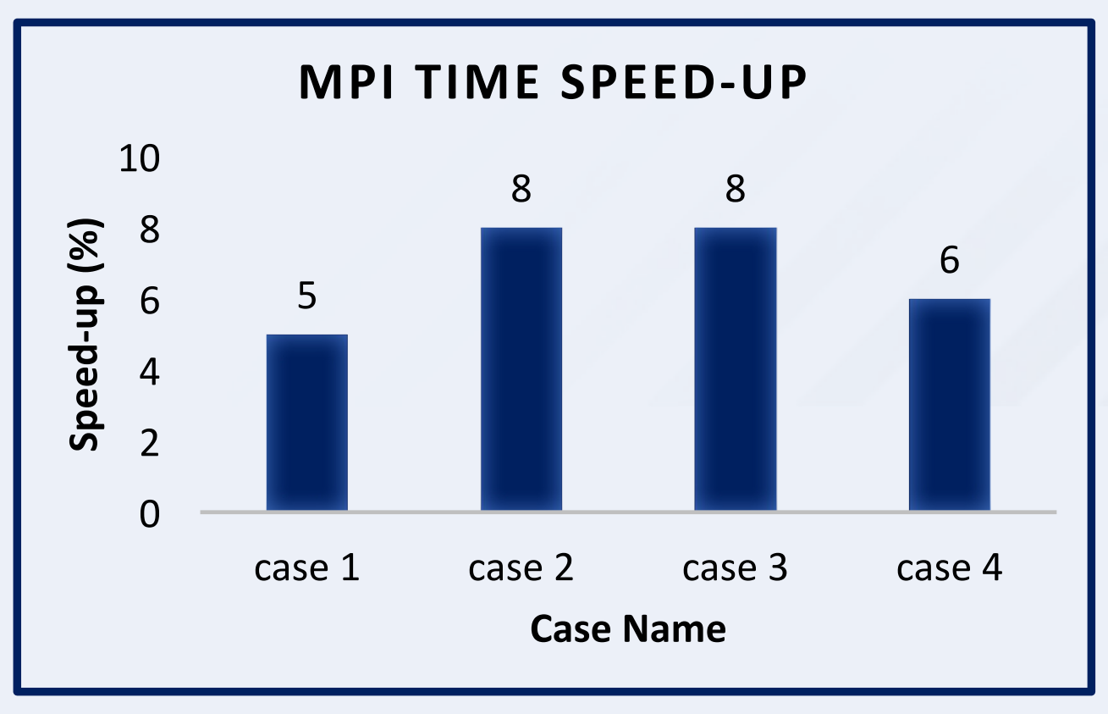


Figure 8 Illustration of MPI Time Speed-up

This significant optimization has shown a clearer image of the main hotspots of MPI communication where others have been increased in terms of time. Figure 9 shows that before optimization MPI Barrier is the main hotspot among other communication routines. However, Figure 10 shows that MPI Barrier is no longer the major hotspot and MPI Allreduce is the one. Besides that, these two figures illustrate that the time spent on the MPI Barrier in the original case was distributed among other MPI routines in the optimized case. For example, MPI Allreduce has increased to 12% in the optimized case compared to the original one. So, these figures illustrates the major MPI collectives bottlenecks after removing unnecessary MPI Barrier calls.

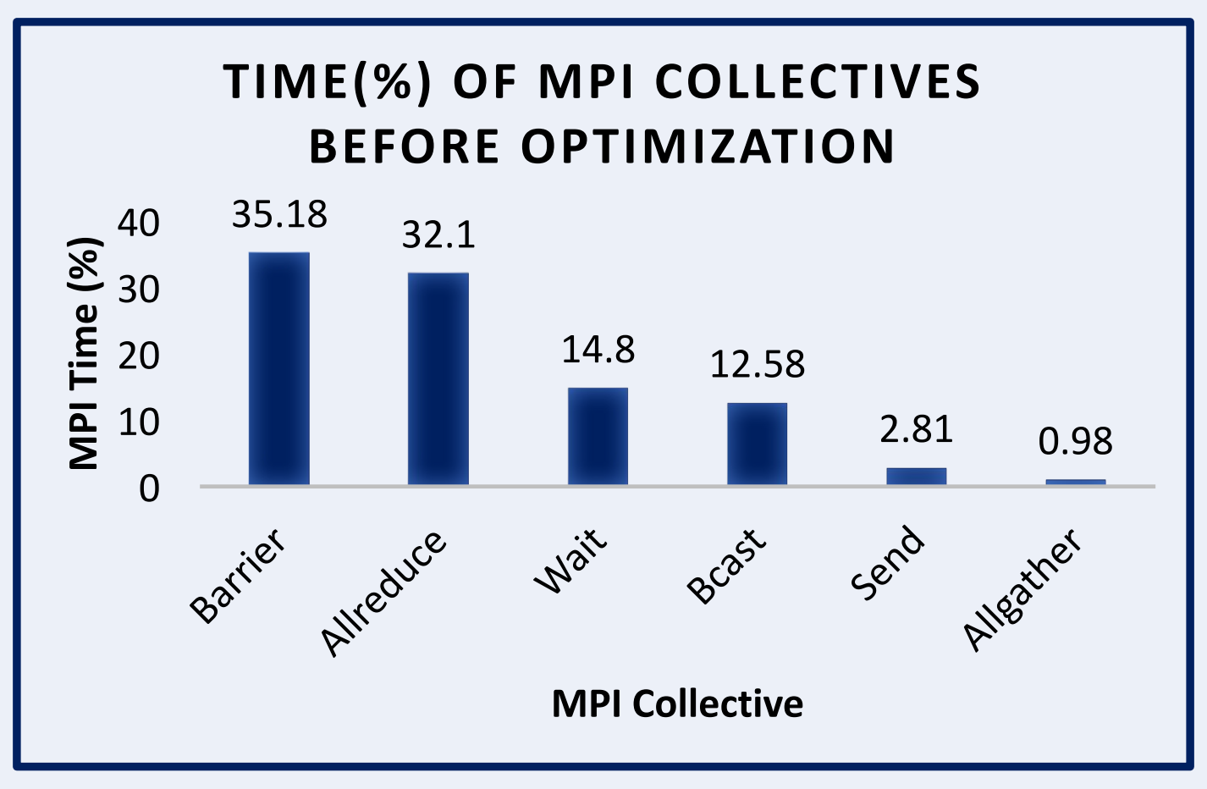


Figure 9 MPI Time Distribution on Original Case

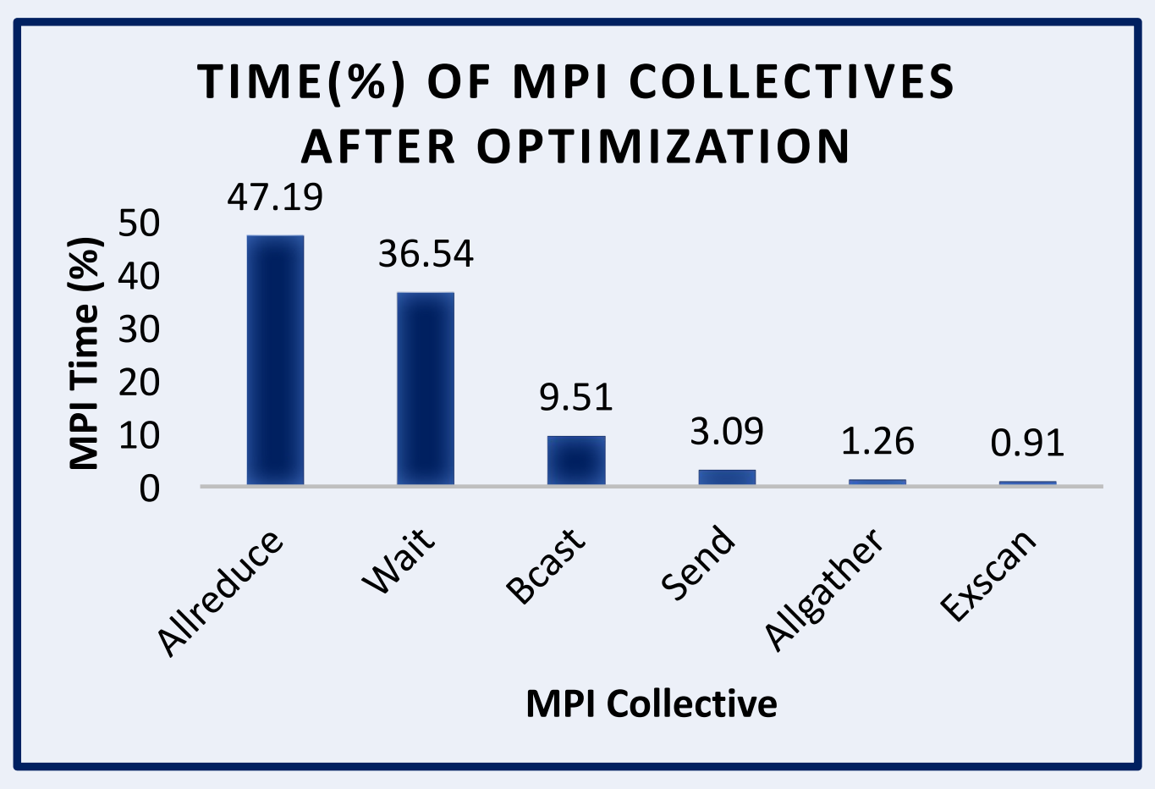


Figure 10 MPI Time Distribution on Optimized Case

Second Optimization Outcomes:

On top of the first optimization, four highly used collective operations within the simulator application were optimized by selecting the best algorithm for each message size. The targeted MPI collectives here are MPI Allreduce, Allgather, Reduce, and Bcast. Figure 11 illustrates the speed-up that was gained out of this optimization.

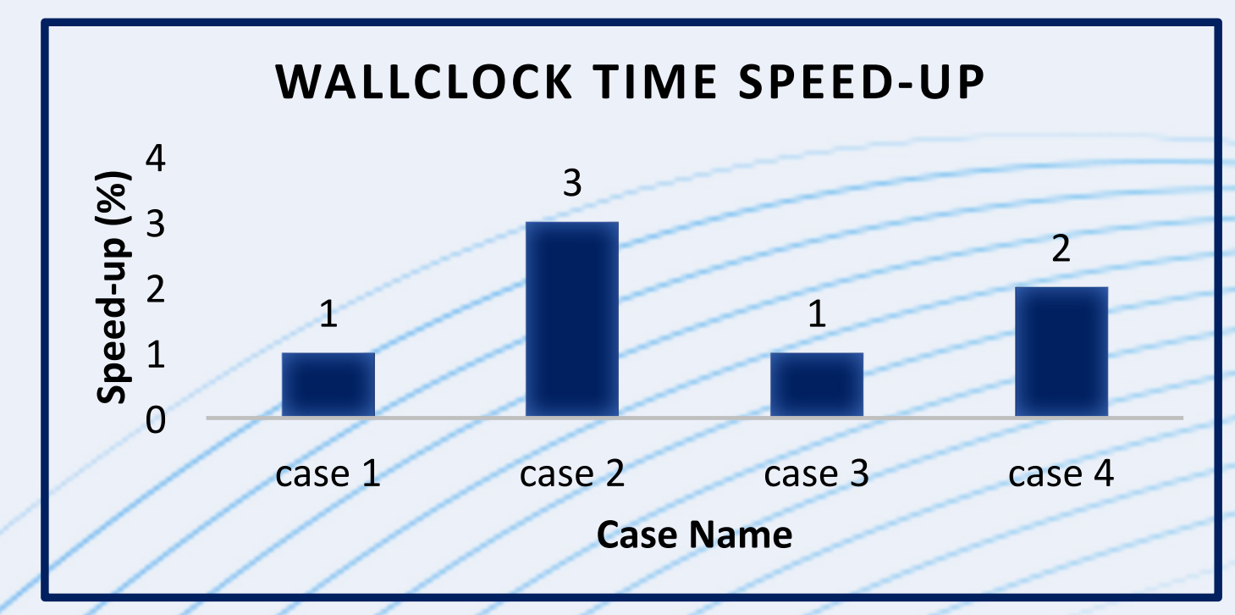


Figure 11 Illustration of Overall Run Time Speed-up

Cumulative Optimization Speedup:

Based on both optimizations that were implemented here, a cumulative speed-up was reported by making further runs that consider both optimizations. Figure 12 illustrates the speedup that has been gained after the two levels of optimization. This cumulative speedup has shown a gain that varies between 3% and 6%.

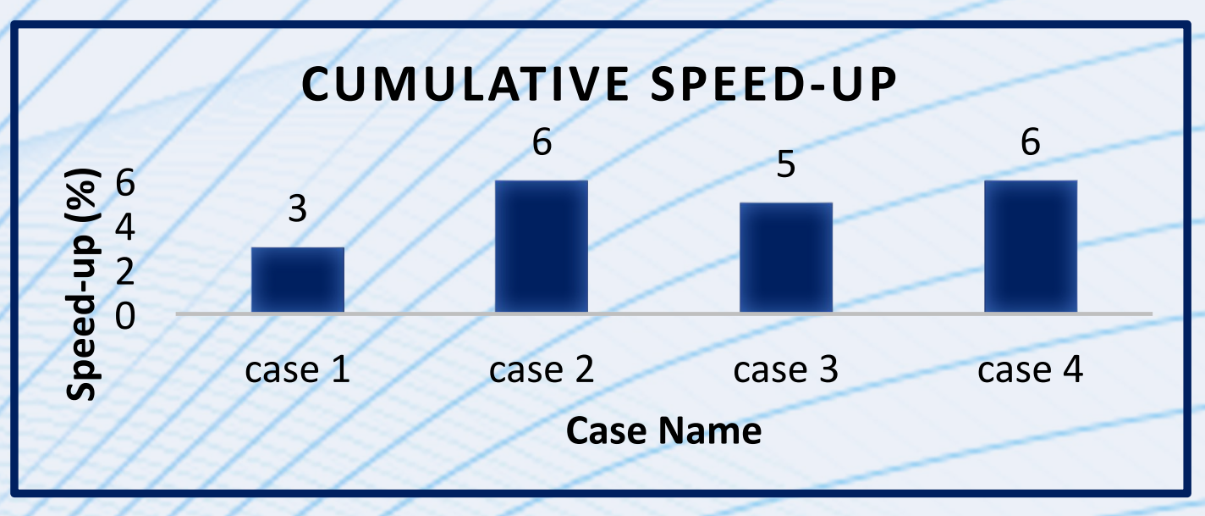
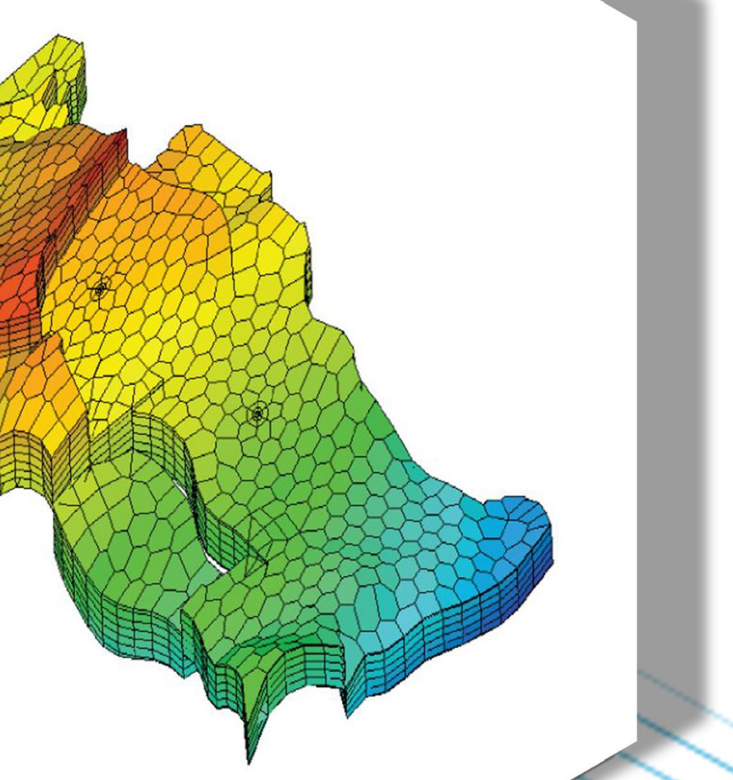


Figure 12: Cumulative Speedup after both Optimizations.



Conclusions

Simulation models in the order of one billion cells pose a challenge on runtime. MPI communication optimization is one of the main approaches to speed up the simulation runs and prepare the software for Exascale computing. MPI Barrier reduction has shown runtime improvements without any changes in the simulation results. That was attained by profiling the source code to identify communication bottlenecks, quantify the communication overhead, and optimizing the MPI Barrier calls.. Another MPI communication optimization technique was considered by evaluating various MPI collectives, Allreduce, Allgather, Reduce, and Bcast. Such optimization has shown a considerable speedup. The cumulative speedup out of both optimizations goes upto 6% which will definitely save a lot of reservoir simulation computing resources.

References

1. Kini, S.P., Liu, J., Wu, J., Wyckoff, P., and Panda, D.K. (2003). Fast and Scalable Barrier Using RDMA and Multicast Mechanisms for Infiniband0Based Clusters, Proceedings of Euro PVM/MPI Conference, (2003).
2. Intel® https://software.intel.com/en-us/articles/intel-mpi-library-collective-optimization-on-intel-xeon-phi
3. MPICH https://www.mpich.org/