A Layered Communication Optimization Method Based on OpenFOAM

Zhipeng Lin*, Xinhai Xu†, Wenjing Yang*, Hao Li*, Yongquan Feng*, Yongjun Zhang†
*State Key Laboratory of High Performance Computing
College of Computer, National University of Defense Technology, Changsha, China
†National Innovation Institute of Defense Technology, Beijing, China
{linzhipeng13, xuxinhai, wenjing.yang, lihao, yjzhang}@nudt.edu.cn, yqfeng0418@163.com

***Abstract*—Since Computational Fluid Dynamics(*CFD*) has emerged as an interdisciplinary science between different subjects, numerous CFD software are increasingly developed on typically layered frameworks. In the literature, only a limited number of optimizations are based on the framework. This paper reveals a general picture of multiphase flow simulation in OpenFOAM and proposes a layered communication optimization method. There are three layers: physical modeling layer, numerical solving layer and parallel computing supporting layer. In these three layers, we respectively eliminate redundant communications in a key module called Multidimensional Universal Limiter for Explicit Solution(*MULES*), decrease global communications by rearranging preconditioned conjugate gradient(*PCG*) solver and add non-blocking reduction call interface. Experiments on high-performance computing platforms show that our optimization dramatically enhances the strong scalability of OpenFOAM, increasing the optimal process number by up to three times in both damBreak flow case and complex 3D head-form case. With a higher degree of parallelism, the shortest execution time, therefore, decreases by 38.16% in the former case and by up to 63.87% in the latter one.**

***Index Terms*—CFD, OpenFOAM, communication optimization, PCG, MULES**

I. Introduction

Computational Fluid Dynamics(*CFD*), which rose in the 1960s, is a discipline growing up with the high-speed advancement of computers. In the course of CFD study, the researchers use the computer to solve governing equations in Fluids by means of the numerical method and then get the simulation results of fluid motion. Therefore, CFD is a typical interdisciplinary subject between Mathematics, Fluid Mechanics, and Computer Science [1].

Many typical CFD software is established on a layered framework for multi-domain users, like OpenFOAM, SOLVCON and Peano [2]–[4], for CFD covers a wide range of disciplines. There are three types of users in the framework: the parallel computing designers at the bottom of the framework(computing supporting layer), the numerical algorithm developers at the intermediate layer(numerical solving layer), and the physical modeling users at the top layer(physical modeling layer). The layered framework shields implementation details for users at different levels. This has brought application developers convenience but also limits the performance of the software. In multiphase flow simulation, for example, the top users can concentrate on their project without consideration of the underlying communication details when implementing a multiphase flow solver, but at the same time ignore the impact of communication on the parallel performance.

OpenFOAM [2] is an object-oriented open source CFD software based on C++. It is extensively used in a diversity of realms such as Fluid Mechanics, Molecular Dynamics, and Rigid Body Dynamics. With a characteristic of the layered framework, OpenFOAM offers the top-level users an abstract interface to describe partial differential equations(*PDE*), and numerical users a middle layer for numerical discrete method and calculation, which is built on the finite volume method(*FVM*). In addition, OpenFOAM supplies a wide range of applications, like solvers, utilities, and limiters, which are straightforward to operate and user-friendly to redevelop for researchers.

There are various researches based on OpenFOAM. Studies [5]–[7] summarize that communication is the main bottleneck in large-scale computing of OpenFOAM. Currently, most of the optimization methods are from a single-layer designer's point of view. For instance, the work in researches [5], [8]–[10] is respectively based on GPU acceleration, FPGA acceleration, compiler optimization, and multi-threaded hybridization to optimize CFD software parallel performance. These optimizations utilize plenty of subtle and profound computer technologies to enhance performance and bring out partly hardware potential, however, none of which take the layered framework into consideration. These methods retain some feeble design and lead to deficient exploitation of the cluster. Thus there is an urgent demand for optimizing OpenFOAM software featured by layered framework from an angle of integrating different levels.

In this paper, we propose a layered communication optimization method based on the layered framework of OpenFOAM software. We first present an analysis of

This work was funded through the National Key Research and Development Program of China (No. 2016YFB0201301) and Science Challenge Project (No. JCKY2016212A502, TZ2016002)

communication bottleneck on multiphase flow simulation. With this, we then present our method which optimizes communication at different layers. Experiments show that the optimized OpenFOAM significantly improve the scalability of multiphase flow simulation.

The main contributions of this paper are:

- We analyze the communication mechanism of MULES limiter in OpenFOAM to find redundant communication and design optMULES which eliminate redundant communication in MULES.
- We design PiPePCG and RePiPePCG solvers to reduce and hide global communication. To the best of our knowledge, this is the first study on PiPePCG in OpenFOAM.
- We evaluate our method based on three typical cases in multiphase flow simulation. Tests show that our method dramatically increases the strong scalability by 3 times and cut execution time down by up to 63.87% in test cases.

II. Communication Bottleneck Analysis

A. Key Modules in Multiphase Flow Simulation

Multiphase flow simulation is a representative application in CFD. Its application areas include various fields of science and technology, such as aerospace, shipbuilding, and oil exploration. Therefore, we focus on multiphase flow. The framework of multiphase flow solver in OpenFOAM is illustrated in Fig. 1. At physical modeling layer, there is not just the building of equations, but also the interface correction of volume fraction [11], which is accomplished by MULES limiter. MULES is composed of local correcting and synchronization. At the numerical solution layer, some iterative solvers, such as the PCG solver will solve the linear system obtained by FVM method. At the computing supporting layer, the communication library, like Psteam, provides parallel support for different modules.

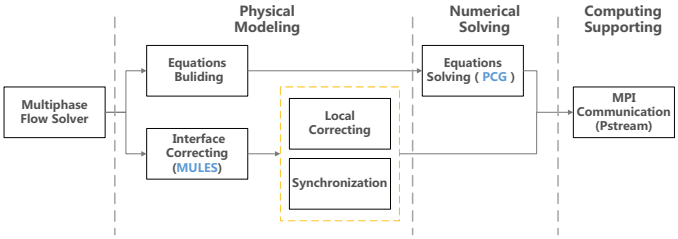


Fig. 1: The framework of Multiphase Flow Solver

Both MULES and PCG are key modules in multiphase flow simulation. To help analyze bottleneck and optimize performance, it is necessary to detect the percentages of cost in different modules. Integrated Performance Monitoring(*IPM*) is a lightweight profiling infrastructure for parallel codes [12]. We use IPM toolkit to track and analyze the damBreak case and the result is shown in Fig. 2, from which we can find that the overhead of MULES and PCG increase rapidly, as the degree of parallelism going up. In the 768 degrees of parallelism, MULES and PCG cost about 69.8% and 20.8% of the total execution time. Thus, it is clear that MULES and PCG is the hotspot of multiphase flow simulation.

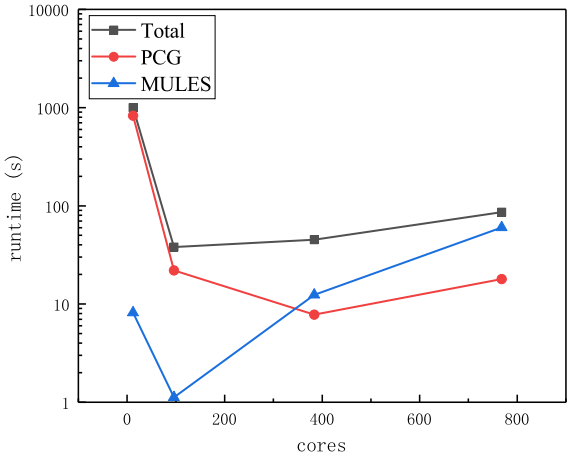


Fig. 2: Cost of MULES and PCG(damBreak)

B. Communication Bottleneck in MULES and PCG

The percentages of key MPI communication is illustrated in Fig. 3. We can figure out that the MPI_Probe() and MPI_Allreduce() costs up to 59.6% and 16.5% of the total execution time with 768 cores. Therefore, MPI_Probe() and MPI_Allreduce() is the communication bottleneck of MULES and PCG.

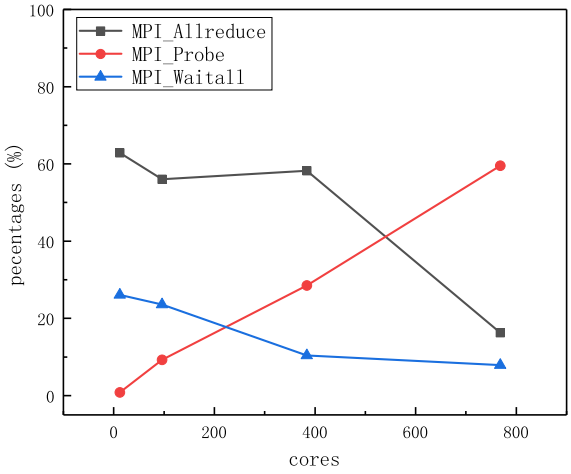


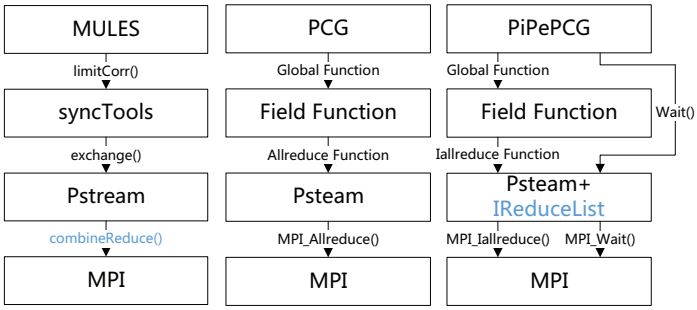
Fig. 3: Percentages of key MPI communication(damBreak)

We analyze the framework of MULES and find that MPI_Probe() is called by a function called *combineReduce()*. As demonstrated in Fig. 4(a), MULES finishes correction by limitCorr(), which will call exchange() in syncTools for synchronization. In order to get the size of sync message, exchange() will call the combineReduce() in Pstream.

As for the bottleneck of PCG, as early as the 1980s and 1990s, Duff I [13] and Dt Sturler [14] have proved that the global communication, like MPI_Allreduce(), caused by dot product operation is the main bottleneck of the development of parallel PCG method from the perspective of experimental test analysis and performance model analysis. During the iteration of PCG, two dot products are required for each step, and each dot product operation requires a global communication, which is a major bottleneck in large scale computing.

C. Redundant Communications in MULES

In multiphase flow simulation, if the grid does not change, like interFoam, interPhaseChangeFoam, driftFluxFoam and many other static solvers, the topology and boundary types between the neighboring processes remain constant, which means the size of sync message keeps the same. In other words, the execution of *combineReduce()* is redundant. By eliminating this redundant communication, we can dramatically reduce communication cost and increase the parallel scalability of MULES.



(a) MULES (b) PCG (c) PiPePCG

Fig. 4: Call Graphs

III. Layered Optimization approach

A. Communication Optimization in MULES

To eliminate the redundant communications in MULES, we develop a new exchange() function that eliminates redundant communication for multiphase flow solvers and keeps the input and output unchanged. In this way, we get optMULES.

As shown in Algorithm 1, the new exchange function add a process to judge whether the function is called by MULES or not, memory the size of sync message and determine whether the size has been recorded or not. There are three new variables: $mulesFlag$, $memFlag$, $memMessage$. Both $mulesFlag$ and $memFlag$ are false at initialization stage. The $mulesFlag$ changes to true just in MULES, while keeps false in other functional modules. Thus the exchange() function is executed normally in other modules. For MULES, if it is the first time the exchange() is called, the result is recorded in $memMessage$ and the $memFlag$ is set to be true, otherwise, the $memMessage$ is assigned to $SizesMessage$. From Algorithm 1, we can find that the new exchange function executes two more judgment sentences than the original exchange function, while executes the combineReduce() function only once, almost completely eliminating redundant combineReduce() communications and leading to the improvement of parallel scalability.

Algorithm 1 New *exchange* Function

Input: the same as original one
Output: the same as original one
1: process before *combineReduce*()
2: // $SizesMessage$ = CombineReduce()
3: /*replace the *combineReduce*() with following judging process*/
4: **if** $mulesFlag$ and $memFlag$ **then**
5: $SizesMessage = memMessage$
6: **else if** $mulesFlag$ and !$memFlag$ **then**
7: $memMessage$ = CombineReduce()
8: $SizesMessage = memMessage$
9: $memFlag$ = true
10: **else**
11: $SizesMessage$ = CombineReduce()
12: **end if**
13: process after *combineReduce*()
14: **return** $OutputSet$

B. Communication Optimization in PCG

In order to settle the bottleneck of PCG global communication, we convert two separated data-dependent dot products into several consecutive dot products with no data dependency to reduce a synchronization point and overlap the non-blocking reduce communication with matrix-vector multiply. To keep mathematically equivalent to the original PCG solver, we design a new Pipelined Preconditioned Conjugate Gradient(PiPePCG) solver in Algorithm 2, which takes both scalability and precision into consideration. Divided into initialization and update, the PiPePCG algorithm in OpenFOAM has a different initialization from classic PiPePCG [15].

By further analysis of the PiPePCG solver, we find that the PiPePCG solver can further overlap the communication by rearrangement. Since the updates of p and x have no data dependency on updates of other vectors, We can update p and x after the execution of MPI_Iallreduce(). By this arrangement, we obtain a RePiPePCG solver.

C. Communication Optimization in Pstream

The existing Pstream library in OpenFOAM supports blocking reduce communication. However, the PiPePCG and RePiPePCG solver use non-blocking MPI_Iallreduce(). Hence, we add some non-blocking reduction calling interfaces to the Pstream.

Based on the call graph of global communication in the original PCG shown in Fig. 4(b), we design new

Algorithm 2 PiPePCG in OpenFOAM

Input: A: $n \times n$ matrix, x_0: initial guess, b: rhs,M: $prec$
Output: x: approximate solution

$r_0 \leftarrow b - Ax_0 \quad u_0 \leftarrow Mr_0 \quad w_0 \leftarrow Au_0$
$\delta_0 \leftarrow (w_0, u_0) \quad \gamma_0 \leftarrow (r_0, u_0)$
$R_0 \leftarrow \sum |r_0^{(i)}| \quad X \leftarrow \sum |x_0^{(i)}| \quad N \leftarrow n$
MPI_Iallreduce on δ_0, γ_0, R_0, X, N
$a \leftarrow (X/N) * A * \vec{1}$
$norm \leftarrow |Ax_0 - a|_1 + |b - a|_1$
MPI_Iallreduec on $norm$
$m_0 \leftarrow Mw_0 \quad n_0 \leftarrow Am_0$
$residual = R_0/norm$

for $j = 1, residual > tolerance, j++$ **do**
  if j > 1 **then**
    $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$
    $\alpha_j \leftarrow \gamma_{j-1}/(\delta_{j-1} - \beta_j * \gamma_{j-1}/\alpha_{j-1})$
  else
    $\beta_j \leftarrow 0 \quad \alpha_j \leftarrow \gamma_{j-1}/\delta_{j-1}$
  end if
  $z_{j-1} \leftarrow n_{j-1} + \beta_j * z_{j-1} \quad q_{j-1} \leftarrow m_{j-1} + \beta_j * q_{j-1}$
  $s_{j-1} \leftarrow w_{j-1} + \beta_j * s_{j-1} \quad p_{j-1} \leftarrow u_{j-1} + \beta_j * p_{j-1}$
  $x_j \leftarrow x_{j-1} + \alpha_j * p_{j-1} \quad r_j \leftarrow r_{j-1} - \alpha_j * s_{j-1}$
  $u_j \leftarrow u_{j-1} - \alpha_j * q_{j-1} \quad w_j \leftarrow w_{j-1} - \alpha_j * z_{j-1}$
  $\delta_j \leftarrow (w_j, u_j) \quad \gamma_j \leftarrow (r_j, u_j)$
  $R_j \leftarrow \sum |r_j^{(i)}|$
  MPI_Iallreduce on δ_j, γ_j, R_j
  $m_j \leftarrow M * w_j \quad n_j \leftarrow A * m_j$
  $residual \leftarrow R_j/norm$
end for
return x_j

calling interfaces in Pstream. In MPI3, prior to using the message, the process needs to execute MPI_Wait() to ensure that the communication is complete [16] after the execution of non-blocking communication operations. Based on the above mechanism, we design the mechanism of non-blocking global communication in PiPePCG and RePiPecG solvers as the flow illustrated in Fig. 4(c). The key design is a static list that records the handles of all unfinished MPI_Iallreduce() request, through which, we can manage the resource for non-blocking reduce communication.

IV. IMPLEMENTATION

Our implementation consists of around 1, 367 lines of C++ code. The key c++ file we modified is listed in Fig. 5. Our code is built on OpenFOAM and does not require any configuration of the computing environment or require additional hardware or toolkits. There is a key implementation in non-blocking allreduce. Although MPI3 has implemented non-blocking global communication, its effect may be not ideal [17]. Therefore, We modify the RePiPePCG solver by dividing the update of p and x into several blocks, after which, is the execution of MPI_Test(). By executing MPI_Test(), program control is switched from OpenFOAM to the MPI core, and the MPI makes progress on global communication. In this way, the benefit of the rearrangement of vector update can be measured quantitatively. As for the PiPePCG solver, we find that matrix-vectors multiply, for instance, $A * m_j$, involving a large number of MPI communications, which can promote non-blocking communication without the addition of MPI_Test().

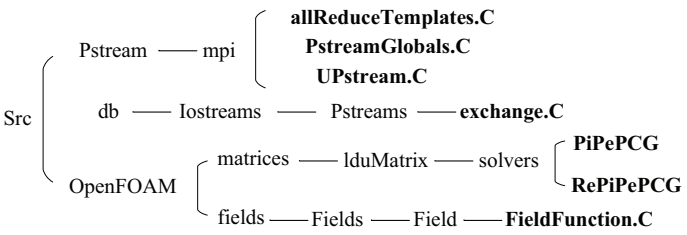


Fig. 5: Structure of Modified Files

V. EXPERIMENTS

A. Platform and Test Cases

We conduct the experiment on an HPC cluster located in the State Key Laboratory of High Performance Computing in China [18]. The cluster consists of 412 high-performance computing nodes, connected by an InfiniBand interconnect network with a total bandwidth of $40Gb/s$. Each node contains two hexacore $2.1GHz$ Intel Xeon E5-2620 CPUs and $16GB$ memory. The operating system on the cluster is Red Hat Enterprise Linux Server 6.5, based on which, the OpenFOAM-2.3.1 is installed. Linked to MVAPICH2-2.3b and slurm 14.03.10, OpenFOAM is on a performance configuration of non-blocking communication.

The cases chosen here depend primarily on the solution patterns that are expected to be run on OpenFOAM. We have tested three typical cases that represent and span a wide-range of case design space in OpenFOAM:

- *cavity* [19]: 2D case without MULES to test the performance of PiPePCG and RePiPePCG.
- *damBreak* [20]: 2D case to verify the superiority of overall communications.
- *Head-form* [21]: 3D case to demonstrate the performance of solving complex 3D multiphase flow problem.

B. Methodology

We follow a four-step methodology to demonstrate the performance of optimized OpenFOAM:

- To verify the correctness of the optimization method, we compared the error of head-form case in different solvers.
- To analyze the parallel scalability of optimization OpenFOAM, we conducted a strong and weak scaling tests based on the cavity and damBreak cases;

- To evaluate the effect of the optimized OpenFOAM on 3D simulation, we tested and analyzed the optimized OpenFOAM based on the 3d head-form case;
- To analyze the effect of communication optimization on MULES and PCG, we tested and detected the MPI communication cost with the lightweight performance tool IPM.

C. Verification Tests

Fig. 6 shows the error of final surface normal pressure field data in the head-form case. The error is relative to the field data from original OpenFOAM. It is clear that the test results using our optimization is approximately equal to the result without optimization.

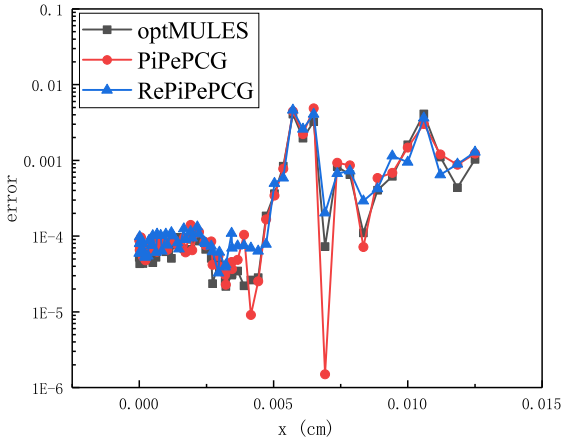


Fig. 6: The Error of Final Filed Data

D. Strong Scaling Tests

First, we performed strong scaling tests on the cavity case and the damBreak case with different parallel cores ranging from 12 to 1536(12, 24, 48, 96, 192, 384, 768, 1152, 1536). The results are shown in TABLE I. We can figure out that, with the optimal core number, the shortest execution time of the optimized OpenFOAM is about 15.39% less than the original one for the cavity case and 38.16% for the damBreak case. For parallel scalability, the optimized OpenFOAM get shortest run time for the damBreak case at 768 cores while the origin one reaches the inflection point at 192 cores which means the strong parallel scalability of optimized OpenFOAM in solving the damBreak case is greatly improved, about three times higher than that of the original OpenFOAM. The cavity case, which excludes the MULES and utilizes our optimization partly, gets the nearly identical parallel scalability for different PCG solver.

The performance of three PCG solvers in solving the cavity case is shown in Fig. 7. We can discover that the performance of non-blocking PCG solvers outperforms blocking PCG algorithm at large core number($>$ 384). When the core number rise up from 384 to 1536, the computation cost per core decreases slightly but the cost of global communication rises rapidly to more than 90%, which implies our rearrangement and communication hiding work well. Furthermore, in order to eliminate synchronization and create a pipeline, variant PCG algorithm need extra computation during the vector update, limiting the speedups at small cores. When the computation overhead is low($<$ 384), they cannot fully overlap communication with extra computation and have a poor performance of speedup. In addition, the cavity case does not use optMULES and thus performs not as good as damBreak at small cores. PiPePCG, rather than the RePiPePCG, scales best due to the impact of MPI_Test(). We will compare this two PCG solver in Section V. G.

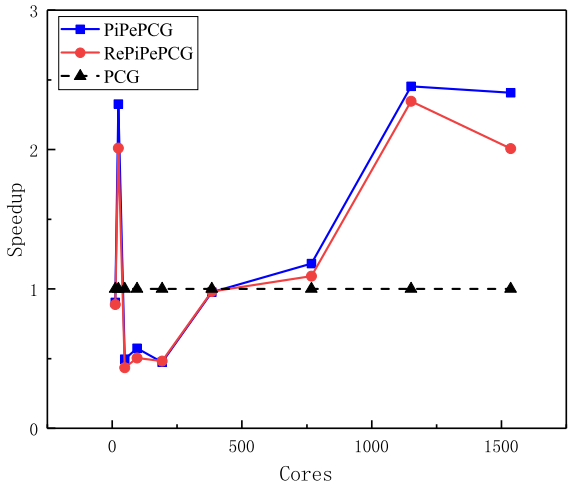


Fig. 7: Strong Scaling Speedups Relative to PCG(cavity)

The result of strong scaling tests for solving damBreak case with original OpenFOAM($OF-ori$) and different optimized OpenFOAM($optMULES+PCG$, $optMULES+PiPePCG$, $optMULES+RePiPePCG$) are shown in Fig. 8. Due to the elimination of redundant communications in MULES, three optimized OpenFOAM have a tremendous increase in strong scaling speedup and the knee point of run time increased dramatically from 192 to 768. This increment implies the strong scalability of OpenFOAM in multiphase flow simulation has a tremendous improvement. With fewer cores ($<$96), the optimization approach does not achieve ideal results because there is less global communication in OpenFOAM when running on small scale cores, which causes a failure to make up for additional computation. The optimization approach worked well when the original OpenFOAM reached its inflection point of execution time.

From the comparison of distinct combinations, we can find that the run time of the three optimal combinations declines in turn, and $optMULES+PiPePCG$ is the most effective optimization combination.

E. Weak Scaling Tests

Next, we analyze parallel performance using weak scaling tests with grid scale changing from 62.5k to

TABLE I: Results of Strong Scaling Tests

case	*solution*	*Runtime With Different Cores Numbers (sec)*								
		12	**24**	**48**	**96**	**192**	**384**	**768**	**1152**	**1536**
cavity	**PCG**	141921.47	62009.74	22886.86	8440.26	1684.50	1036.79	***806.14***	85990.41	88625.36
	PiPePCG	157132.26	26658.99	46154.35	14703.69	3555.23	1062.62	***682.04***	35060.98	36814.02
	RePiPePCG	159703.21	30858.64	52615.95	16762.2066	3490.59	1056.41	***738.37***	36638.72	44176.82
damBreak	**OF-ori**	999.75	383.47	122.25	37.99	***24.16***	45.19	83.31	245.57	664.27
	optMULES+PCG	1000.89	385.19	123.65	40.17	21.01	18.98	***17.98***	46.87	120.55
	optMULES+PiPePCG	1291.725	565.533	227.826	64.368	22.33	16.77	***14.94***	53.77	113.25
	optMULES+RePiPePCG	1420.89	622.08	253.14	71.52	25.94	17.19	***15.98***	49.68	122.04

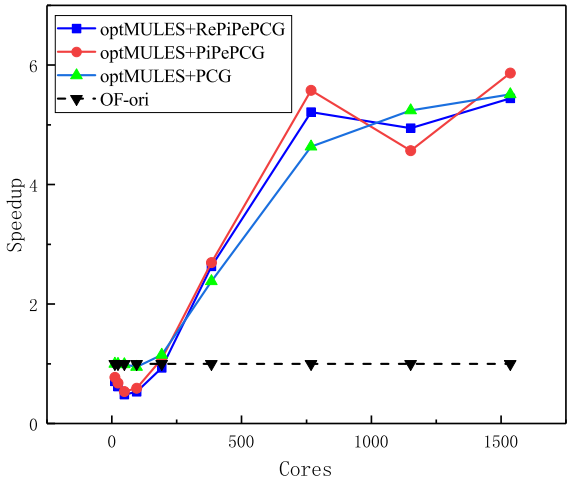


Fig. 8: Strong Scaling Speedups Relative to OF-ori(damBreak)

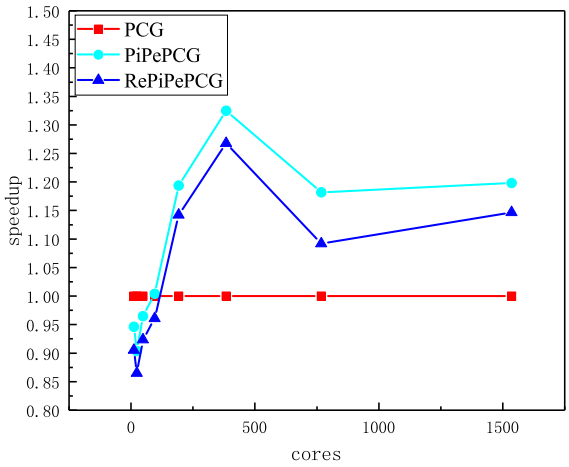


Fig. 9: Weak Scaling Speedups Relative to PCG(cavity)

8000k(62.5k, 125k, 250k, 500k, 1000k, 2000k, 4000k, 8000k) in cavity case and 127,2k to 16329.6k(127.2k, 254.4k, 508.8k, 1017.6k, 2041.2k, 4082.4k, 8164.8k, 16329.6k) in damBreak case. The number of cores increases from 12 to 1536(12,24,48,96,192,384,768,1536), through which we can set the mesh size on each process constant.

TABLE II records the run time of cavity and damBreak cases on different optimization methods. As shown in TABLE II, in the cavity case, PiPePCG and RePiPePCG produce relative speedups of 1.324x and 1.267x with 384 cores. Tests on the damBreak case, using higher work per core, show optMULES + PiPePCG performing best with up to 4.274x speedups

Fig. 9 and Fig. 10 present the weak scalability of optimized OpenFOAM in the cavity and damBreak cases. We make two key observation from these two graphs. First, our method, when increasing the number of cores, can effectively reduce run time and communication cost. Especially in the damBreak case, our optimized OpenFOAM get a distinct speedup line. Second, differences between speedup of two cases have a sharp contrast(see Fig. 10) and point out that eliminating redundant communication in MULES is more effective than rearranging PCG algorithm for improving weak scalability.

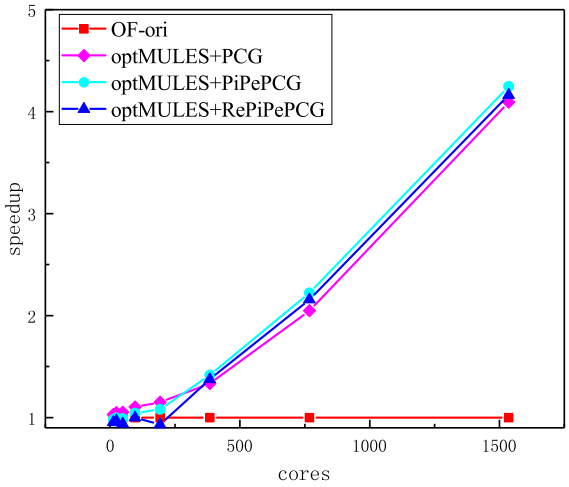


Fig. 10: Weak Scaling Speedups Relative to OF-ori(damBreak)

F. A Test on the 3D Case

Finally, based on the previous tests, we use the *optMULES + PiPePCG* solver as the optimization method for OpenFOAM and test the optimization effect in 3D multiphase flow simulation based on the head-form case. The model solver we used for this problem is interPhaseChangeFoam, a solver for two in-compressible, isothermal immiscible fluids with phase-change, with the volume of fluid phase-fraction based interface capturing

TABLE II: Results of Weak Scaling Tests

case	*solution*	*Runtime With Different Cores Numbers (sec)*							
		12	*24*	*48*	*96*	*192*	*384*	*768*	*1536*
cavity	**OF-ori**	123.32	182.43	262.48	388.86	630.90	718.82	806.14	1036.79
	PiPePCG	130.36	201.82	272.01	387.34	528.5	542.57	682.04	865.31
	RePiPePCG	136.22	210.9	284.25	404.77	552.28	566.98	738.37	904.24
damBreak	**OF-ori**	7.77	9.29	12.41	16.55	24.16	48.6	124.3	640.8
	optMULES+PCG	7.55	8.87	11.78	14.98	21.01	36.41	60.66	156.57
	optMULES+PiPePCG	7.92	9.31	12.52	15.87	22.33	34.28	55.94	150.88
	optMULES+RePiPePCG	8.15	9.58	13.21	16.54	25.94	35.3	57.61	153.96

approach [2].

Fig. 11 displays the lines of run time for solving the head-form case with original OpenFOAM and optimized OpenFOAM, using different cores. we can analyze that the turning point of run time line is improved from 96(OF-ori) to 384(OF-opt), about three times higher than that of the original one, and there is a 63.87% decrease in the execution time of the optimized OpenFOAM, compared with the original OpenFOAM respectively. From the graph, we can see that optimized OpenFOAM dramatically improves the performance of strong scalability and reduces execution time, which indicates that the OpenFOAM communication optimization method combining the characteristics of CFD layered framework has obvious optimization effect in simulating the three-dimensional complex multiphase flow.

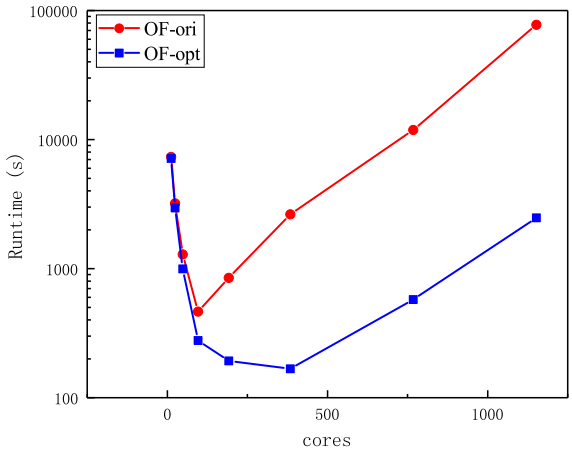


Fig. 11: Strong Scaling Run Time in head-form

G. MPI Communication Tests

To evaluate the communication optimization in detail, we probe the MPI communication cost in damBreak case. The percentages of key MPI operation are manifested in Fig. 12. Because the new MULES has eliminated the redundant communication, there is little percentage of time spent in executing MPI_Probe(), almost 0%. Benefited from this design, the parallel performance in solving multiphase flow problem has a great improvement. Compared with Fig. 3, we can find the dip in MPI_Allreduce(), the rise in MPI_Wait() and the appearance of MPI_Iallreduce() which is caused by the augmentation of non-blocking reduction in the implementation of PiPePCG. According to our statistics, the PiPePCG can bring an up to almost 20% decrease in solving the linear system.

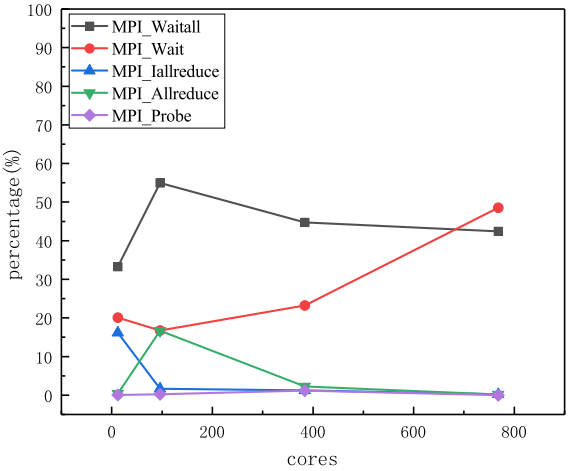


Fig. 12: Percentages of Key MPI Communication(opt)

The Fig. 13 profiles the average MPI communication cost of RePiPePCG and PiPePCG in solving cavity case at 768 cores. We find that in the RePiPePCG, approximately 8.4% of the total execution time (82.02s) is executing MPI_Test(). After subtracting this MPI operation time, we obtain an estimate of the best execution time of RePiPePCG, which will drop to 656.35s and is potentially 3.97% shorter than that of PiPePCG. This result suggests a 3 to 4% potential speedup in RePiPePCG.

VI. Related Work

A. Performance bottlenecks of OpenFOAM

OpenFOAM has been extensively tested and carefully analyzed on various high-performance computing platforms, including Tianhe No.2 [6], [18], [22], [23]. In these experiments, the focus on parallel performance bottlenecks for OpenFOAM was highlighted. Culpo et al. [6] pointed out that the solving process of linear systems has an important influence on the application of OpenFOAM on large-scale clusters. Rivera et al. [24] conducted an in-depth analysis of the communication process in OpenFOAM with the performance tool IPM and found that a large amount of collective communication is a major performance bottleneck of OpenFOAM.

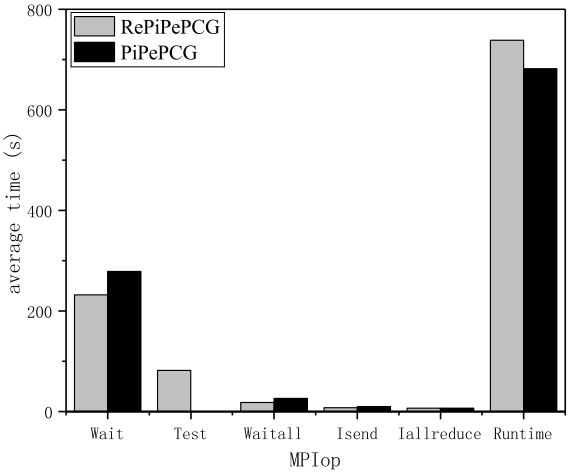


Fig. 13: Comparison of Average MPI Cost(cavity,768)

B. Summary of Optimization in OpenFOAM

The current OpenFOAM optimization can be divided into two main categories: one is through the hybrid OpenMP and MPI communications, compiler optimization, FPGA acceleration, GPU acceleration and vector instruction set and other underlying technology to accelerate the computing and communications of OpenFOAM [5], [8]–[10], and the other is to optimize the libraries of OpenFOAM [18], [25]. These optimizations get impressive effectiveness, however, they retain some feeble design in MPI communication and lead to deficient exploitation of the cluster

VII. Conclusion and Future Work

In this paper, we propose a layered communication optimization method based on the layered framework of CFD software. Extensive experiments in multiphase flow problem show that we can eliminate redundant communications in MULES and rearrange PCG to produce a higher performance of scalability and execution time.

In the future, we plan to design a dynamic elimination mechanism in MULES for multiphase flow simulation with dynamic grids. At the same time, we will optimize the GAMG, smoothsolver and other solvers in OpenFOAM.

References

[1] L. S. Sørensen, "An introduction to computational fluid dynamics: The finite volume method," 1999.

[2] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, "Openfoam: A c++ library for complex physics simulations," in *International workshop on coupled methods in numerical dynamics*, vol. 1000. IUC Dubrovnik, Croatia, 2007, pp. 1–20.

[3] Y.-Y. Chen, D. Bilyeu, L. Yang, and S.-T. J. Yu, "Solvcon: a python-based cfd software framework for hybrid parallelization," in *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2011, p. 1065.

[4] H.-J. Bungartz, M. Mehl, T. Neckel, and T. Weinzierl, "The pde framework peano applied to fluid dynamics: an efficient implementation of a parallel multiscale fluid dynamics solver on octree-like adaptive cartesian grids," *Computational Mechanics*, vol. 46, no. 1, pp. 103–114, 2010.

[5] A. AlOnazi, D. Keyes, A. Lastovetsky, and V. Rychkov, "Design and optimization of openfoam-based cfd applications for hybrid and heterogeneous hpc platforms," *arXiv preprint arXiv:1505.07630*, 2015.

[6] M. Culpo, "Current bottlenecks in the scalability of openfoam on massively parallel clusters," *PRACE white paper to appear on http://www. praceri. eu*, 2011.

[7] V. Starikovičius, R. Čiegis, and A. Bugajev, "On efficiency analysis of the openfoam-based parallel solver for simulation of heat transfer in and around the electrical power cables," *Informatica*, vol. 27, no. 1, pp. 161–178, 2016.

[8] M. Taouil, "A hardware accelerator for the openfoam sparse matrix-vector product," *Delft University of Technology, Delft, The Netherlands*, 2009.

[9] X. G. Ren, "Optimize openfoam from the compiler perspective," in *Applied Mechanics and Materials*, vol. 687. Trans Tech Publ, 2014, pp. 3183–3186.

[10] P. Dagnaa and J. Hertzerb, "Evaluation of multi-threaded openfoam hybridization for massively parallel architectures," *PRACE WP98, Aug*, vol. 20, 2013.

[11] G. Tryggvason, R. Scardovelli, and S. Zaleski, *Direct numerical simulations of gas–liquid multiphase flows*. Cambridge University Press, 2011.

[12] D. Skinner, "Performance monitoring of parallel scientific applications," Ernest Orlando Lawrence Berkeley NationalLaboratory, Berkeley, CA (US), Tech. Rep., 2005.

[13] I. S. Duff and G. A. Meurant, "The effect of ordering on preconditioned conjugate gradients," *BIT Numerical Mathematics*, vol. 29, no. 4, pp. 635–657, 1989.

[14] E. de Sturler, "A performance model for krylov subspace methods on mesh-based parallel computers," *Parallel Computing*, vol. 22, no. 1, pp. 57–74, 1996.

[15] P. Ghysels and W. Vanroose, "Hiding global synchronization latency in the preconditioned conjugate gradient algorithm," *Parallel Computing*, vol. 40, no. 7, pp. 224–238, 2014.

[16] W. Gropp and E. Lusk, "Users guide for mpich, a portable implementation of mpi," Argonne National Lab., IL (United States), Tech. Rep., 1996.

[17] P. R. Eller and W. Gropp, "Scalable non-blocking preconditioned conjugate gradient methods," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 18.

[18] H. Li, X. Xu, M. Wang, C. Li, X. Ren, and X. Yang, "Insertion of petsc in the openfoam framework," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 2, no. 3, p. 16, 2017.

[19] O. Botella and R. Peyret, "Benchmark spectral results on the lid-driven cavity flow," *Computers & Fluids*, vol. 27, no. 4, pp. 421–433, 1998.

[20] A. Z. Zhainakov and A. Kurbanaliev, "Verification of the open package openfoam on dam break problems," *Thermophysics and Aeromechanics*, vol. 20, no. 4, pp. 451–461, 2013.

[21] M.-R. Pendar and E. Roohi, "Investigation of cavitation around 3d hemispherical head-form body and conical cavitators using different turbulence and cavitation models," *Ocean Engineering*, vol. 112, pp. 287–306, 2016.

[22] M. Manguoglu, "A general sparse sparse linear system solver and its application in openfoam," *Partnership for Advanced Computing in Europe*, 2012.

[23] M. Wang, Y. Tang, X. Guo, and X. Ren, "Performance analysis of the graph-partitioning algorithms used in openfoam," in *Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 99–104.

[24] O. Rivera and K. Furlinger, "Parallel aspects of openfoam with large eddy simulations," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. IEEE, 2011, pp. 389–396.

[25] S. Ito, S. Ohshima, and T. Katagiri, "Ssg-at: An autotuning method of sparse matrix-vector multiplicataion for semi-structured grids–an adaptation to openfoam," in *Embedded Multicore Socs (MCSoC), 2012 IEEE 6th International Symposium on*. IEEE, 2012, pp. 191–197.