Scalability of OpenFOAM for Viscoelastic Solver on High Performance Systems

Gang HUANG[a], Can-Qun YANG[b], Xiao-Wei GUO[c,*], Cheng-Kun WU[d], Xiang ZHANG[e]

College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China

[a]huanggang15@nudt.edu.cn, [b]guoxiaowei@nudt.edu.cn, [c]canqun@nudt.edu.cn, [d]chengkun_wu@nudt.edu.cn, [e]zhangxiang_43@aliyun.com

[*]Corresponding author

Keywords: Scalability, Communication Overhead, Global Reduction.

Abstract. In this paper, we investigate the scalability of OpenFOAM for the viscoelastic solver which is implemented in our previous study on HPC platforms. Results show that the solver scales reasonably well up to 256 cores. Further profiling shows that the scalability is greatly restricted by global reduction, which is introduced by numerous scalar product operations of parallel PCG algorithm in OpenFOAM.

Introduction

OpenFOAM is an open source toolbox for solving Particle Differential Equations based on the Finite Volume Method (FVM). As one of the most popular open-source tools, OpenFOAM is widely used in CFD and continuum mechanics. Written in C++, OpenFOAM makes good use of OOP (Object Oriented Programming) and offers a flexible framework for users to customize and extend its existing functionality freely [1]. With the help of sufficient abstraction in OpenFOAM, users could focus on the mathematical model of the problem without considering the implementation detail. This good feature helps OpenFOAM to foster wide acceptance in both academic and industrial CFD communities [2].

OpenFOAM is parallelized with MPI (Message Passing Interface) and encapsulates the basic MPI calls in its functional libraries, which makes it easy to be extended and optimized. However, such feature also brings difficulties to study the parallel behavior of OpenFOAM on the high-performance systems. The scalability of OpenFOAM is still an open question and several related studies have been conducted in recent years. Orlando Rivera and Fürlinger, Karl test the scalability of OpenFOAM on a supercomputer SGI Altix 4700, with a large Eddy Simulation (LES) benchmark *pisoFoam* up to 2.15 million cell mesh [3]. They used more than 256 MPI tasks and found that the solver has an acceptable scalability up to 64 MPI tasks. Massimiliano Culpo test the parallel performance of OpenFOAM with the lid-driven cavity flow benchmark on two supercomputers: CINECA PLX (PRACE Tier-1 system) and TGCC CURIE (PRACE Tier-0 system) , and analyzed the bottleneck in scalability of OpenFOAM [2]. He showed a reasonably well scalability of the application up to 1000 tasks and discussed the cause of this bottleneck. Duran *et.al* used TGCC CURIE (a Tier-0 system) to investigate the scalability of a bio-medical flow simulation with the solver *icoFoam* [4]. They achieved super linear speed-up up to 2048 cores for the 64 million cells.

All the previous studies of the scalability mainly focused on the existing solver and seldom mentioned applications extended from OpenFOAM. In this work, we turn to study the scalability of the viscoelastic solver which extends from a built-in solver of OpenFOAM. This viscoelastic solver has been implemented in our recent study on non-equilibrium steady states for a binary viscoelastic fluid [5]. The rest of this paper is organized as follows: The model and the test environment is introduced in section 2 briefly. The test results are shown in section 3. Section 4 presents the analysis and discussion for the test results. Section 5 summaries this work and contains our conclusions.

Model and Experimental Configuration

Recently, we gave a numerical study on sheared entangled polymer mixtures and first revealed the existence of the viscoelastic non-equilibrium steady states [5]. In that work, we implemented a parallel viscoelastic fluid solver extending from *icoFoam*, a built-in solver of OpenFOAM. This solver stems from the unified two-fluid model [6]: Flory-Huggins-Rolie-Poly (FH-RP) fluid model, which combined the Flory-Huggins free energy function [7] and the Rolie-Poly constitutive equation [8]. For more details about the FH-RP model, please refer to [5].

The simulation domain of this solver is a two-dimensional rectangular box. The top and bottom boundaries are physical walls and the other two sides are periodic boundary conditions. A steady shear is applied by moving the upper wall boundary to the right at a constant speed. The flow geometry is shown as Fig. 1.

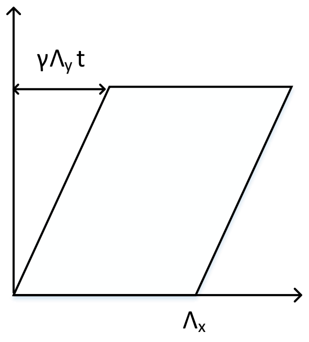


Fig. 1. Flow geometry of sheared flow

In this paper, we solve the FH-RP model with PISO algorithm in OpenFOAM. For spatial discretization terms of the equation, Gauss MINMOD and Gauss linear scheme are applied. For temporal terms, Euler scheme is applied. After this treatment, all the equations can be reduced to linear systems, so that we can use the iterative solvers in OpenFOAM to get the solutions at every time step. Preconditioned conjugate gradient (PCG) method is used to solve the Pressure equation, and preconditioned biconjugate gradient (PBICG) method is for other equations.

All the tests are executed on a high-performance cluster with 390 nodes, each with 24 Intel Xeon E5-2692 CPU and 64GB of memory per node. Each test has been run without I/O for 2000 time steps ($\Delta t = 0.02$). The OpenFOAM version is v1600+ and complied with GNU 4.9.2 compiler. In order to evaluate the parallel performance, the code is instrumented with Integrated Performance Monitor 2.0.6 (IPM), which is a portable profiling infrastructure for parallel codes and provides a low-overhead performance profile in a parallel program [9].

Scalability Results

In this section, we conduct a series of experiments to analyze the scalability of the viscoelastic solver. We start with the strong scaling study from 32 to 2048 cores, then we turn to investigate the weak scaling. Finally, we explain experimental results by analyzing the PCG algorithm.

Strong Scaling Study

To study the strong scaling of the viscoelastic solver, we increase the number of cores from 32 to 2048 on a 2.1 million mesh (2048$\times$1024). All runs are executed parallel under the same condition except the number of cores. The domain is decomposed by means of Scotch method with the uniform weight, so that each subdomain is assigned the same number of cells to ensure the same computational cost for each core. Fig. 2(a) and Fig. 2(b) show the wall time and speed-up of the strong scaling study varying with the number of cores respectively.

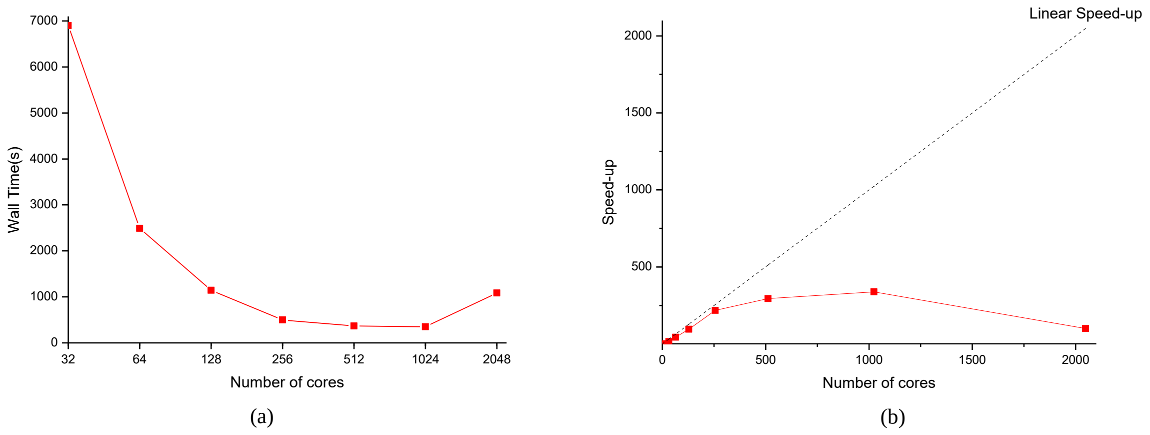


Fig. 2. Result of strong scaling studying. (a) Wall time varies with the number of cores. (b) Speed-up varies with the number of cores

From Fig. 2(a), we find that the wall time decreases drastically from 32 to 256 cores, which indicates this solver can effectively scale up to 256 cores. After that, the benefit of time reduction brought by extra cores cannot makes up the side-effect, such as resource overhead. Compared with 256 cores, the test on 512 cores uses double cores but its time is only reduced by 26%. For 1024 cores, the wall time is almost the same as 512 cores.

Furthermore, the slowdown of the execution time occurs when the number of cores reaches 2048, so it is meaningless to continue increasing cores and the bottleneck of scalability occurs. In Fig. 2(b), this phenomenon is manifested as the significant deviation of the linear scalability in speed-up when the number of cores exceeds 256.

To find the cause of this bottleneck in scalability, we have a further study on the communication overhead of each run. With the help of IPM, we plot the percentage of wall time spent in MPI in Fig. 3.

As shown in Fig. 3, the percentage of wall time spent in communication keeps growing with the increase of cores. It even accounts for more than 90% of wall time in 2048 cores, which exceeds the computation overhead and becomes the most time-consuming part. At this time, the reduction of computational cost cannot make up the increase of communication overhead. So the total time begins to increase, and the bottleneck of scalability occurs.

IPM not only measures the overall time spent in MPI, but also gives statistics for specific MPI routine overhead. Fig. 4(a) shows specific MPI function overhead as a fraction of wall time varies with the number of cores, where the ratio of *MPI_Allreduce* is far more than other routines and increase indefinitely. By contrast, ratios of the other functions either remain constant or decrease slightly. Fig. 4(b) is a pie chart, showing the ratio of each MPI routine to the total communication overhead on 128 cores. The blue part of Fig. 4(b) is the ratio of *MPI_Allreduce*, which accounts for the largest proportion of 84.19%. It is clearly seen that *MPI_Allreduce* consistently dominates the time spent in communication.

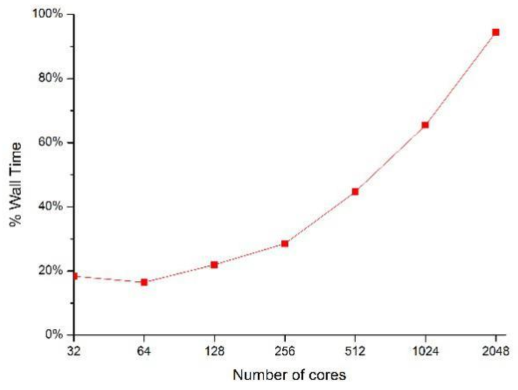
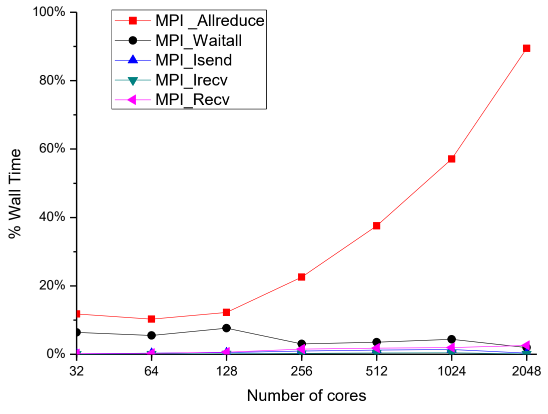
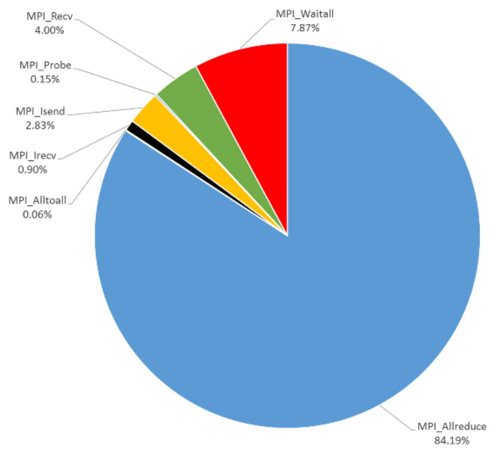


Fig. 3. Percentage of wall time spent in MPI varies with the cores count for strong scaling



(a)



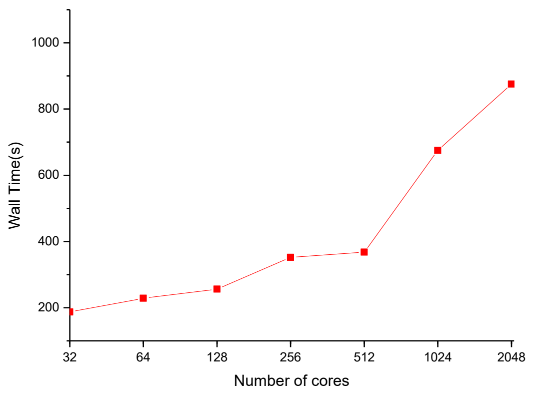
(b)

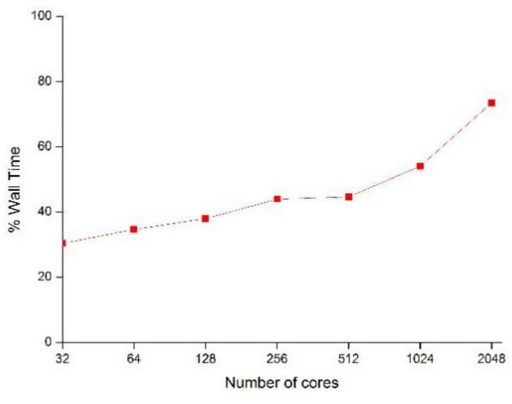
Fig. 4. Statistics of specific MPI routine overhead for strong scaling. (a) Specific MPI function overhead as a fraction of wall time. (b) Ratio of each MPI function to the total communication time

Weak Scaling Study

To reveal problems which are not related with load imbalance due to small domains, we turn to study the weak scaling. Weak scaling is defined as how the wall time varies with the number of processors for a fixed cell per core [10]. In this study, we fix 4096 cells per core to investigate the weak scaling, for this problem size has proved to be large enough for 256 cores in the strong scaling test.

In Fig. 5(a), the wall time keeps growing as the problem size increases. In other words, the scalability gets worse with the increase of problem size. Fig. 5(b) presents the percentage of wall time spent in MPI varies with the number of cores for 4096 cells per core. Similar with Fig. 3, the time spent in MPI keeps increasing, which indicates that the limitation of weak scaling is also due to the increasing communication overhead.





(a) (b)

Fig. 5. Result of weak scaling study with 4096 cells per core. (a) Wall time varies with the number of cores. (b) Percentage of wall time spent in MPI varies with the number of cores

Analysis and Discussion

In the strong and weak scaling study, we find that the communication overhead is proportional to the number and the area of subdomains for a specific simulation. As the number of subdomains increases along with the number of cores, the network of cluster could be jammed by sending and receiving numerous messages, which may cause the bottleneck in scalability of OpenFOAM [10].

According to the strong scaling study, the global reduction operations accounts for the largest proportion of communication overhead in all MPI routines. To identify the source of global reduction, we turn to analyze the main flow of PCG algorithm.

Table 1 shows the main flow of PCG algorithm, which is used to solve main governing equations of our solver (The flow of PBICG and PCG are of the same, except that matrix A is replaced by A^*). PCG algorithm contains two main types of operations: matrix-vector multiplication and scalar product. Due to the discretization of the finite volume method, the coefficient matrix used in PCG is usually a sparse matrix that non-zero elements are only near the main diagonal. When each row of the sparse matrix is multiplied by a column vector in parallel computation, only data from adjacent process is needed in general, so the communication mode of matrix-vector multiplication is point-to-point. As for the parallel scalar product, the operation needs to gather each element of two vectors, which requires data from all cores. Thus the global reduction, especially *MPI_Allreduce* is essential for the scalar product.

As shown in Table 1, step 6,8,9 contain the matrix vector multiplication, while the scalar product is in step 6,10,13 (step13 requires to gather all the local residuals from all the cores, so it also needs a global reduction).

From Table 1, we can find that the internode communication is very frequent in solving the linear equations. More importantly, there are numerous collective communication operations in PCG algorithm. The global reduction requires not only the synchronization of all cores, but also implicit barriers in every iteration, which makes the corresponding overhead increases drastically as the parallel scale expands. Due to the call of each global reduction, *MPI_Allreduce* also has a dramatic increase and dominates the time spent in communication. Therefore, it is the global reduction in parallel PCG algorithm that causes the bottleneck in the scalability of OpenFOAM.

Table 1. Main flow of PCG algorithm

```
Input: $A$ :coefficient matrix, $\vec{b}$ :constant vector, $P$: preconditioner matrix, $\vec{x}_0$ :initial solution $tol$ :bound of residual
Output: $\vec{x}$
1: $\vec{r}_0 = \vec{b} - A\vec{x}$
2: $\vec{z}_0 = P\vec{r}_0$
3: $\vec{p}_0 = \vec{z}_0$
4: $k := 0$
5: **repeat**
6:     $\alpha_k = \frac{\vec{r}_k^T \vec{z}_k}{\vec{p}_k^T A \vec{p}_k}$
7:     $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$
8:     $\vec{r}_{k+1} = \vec{r}_k - \alpha_k A \vec{p}_k$
9:     $\vec{z}_{k+1} = P\vec{r}_{k+1}$
10:    $\beta_k = \frac{\vec{z}_{k+1}^T \vec{r}_{k+1}}{\vec{z}_k^T \vec{r}_k}$
11:    $\vec{p}_{k+1} = \vec{z}_{k+1} + \beta_k \vec{p}_k$
12:    $k = k + 1$
13:**until** $\left|\vec{r}_k\right| < tol$
14: $\vec{x} = \vec{x}_k$
```

Conclusion

In this paper, we test the scalability of an OpenFOAM-based viscoelastic solver which is implemented in our previous work. In the strong scaling study, we have achieved an acceptable scalability up to 256 cores. After that, the bottleneck of scalability occurs due to the dramatic increase in communication overhead. Furthermore, we find that the global reduction function *MPI_Allreduce* dominates the time spent in communication. Since the global reduction requires the synchronization of all cores, the corresponding overhead increases drastically as the parallel scale expands, which causes the bottleneck of scalability in OpenFOAM. By analyzing the flow of PCG algorithm used in our solver, we reveal that *MPI_Allreduce* is introduced by numerous scalar product operations from PCG algorithm. Thus, the global reduction in parallel PCG algorithm is the intrinsic cause of the bottleneck in the scalability of OpenFOAM.

Acknowledgement

This work is supported by the National Key Research and Development Program of China, No. 2016YFB0200400, and the National Natural Science Foundation of China (NSFC), No. 31501073.

References

[1] Weller, H.G., et al., A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in physics, 1998. 12(6): p. 620-631.

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

[3] Rivera, O., K. Fürlinger, and D. Kranzlmüller. Investigating the scalability of OpenFOAM for the solution of transport equations and large eddy simulations. in International Conference on Algorithms and Architectures for Parallel Processing. 2011. Springer.

[4] Duran, A., et al., Scalability of OpenFOAM for bio-medical flow simulations. The Journal of Supercomputing, 2015. 71(3): p. 938-951.

[5] Guo, X.-W., et al., Non-equilibrium steady states of entangled polymer mixtures under shear flow. Advances in Mechanical Engineering, 2015. 7(6): p. 1687814015591923.

[6] Guo, X.-W., et al., Interface instabilities and chaotic rheological responses in binary polymer mixtures under shear flow. RSC Advances, 2014. 4(105): p. 61167-61177.

[7] Tanaka, H., Unusual phase separation in a polymer solution caused by asymmetric molecular dynamics. Physical review letters, 1993. 71(19): p. 3158.

[8] Likhtman, A.E. and R.S. Graham, Simple constitutive equation for linear polymer melts derived from molecular theory: Rolie–Poly equation. Journal of Non-Newtonian Fluid Mechanics, 2003. 114(1): p. 1-12.

[9] Fuerlinger, K., N.J. Wright, and D. Skinner. Effective performance measurement at petascale using ipm. in Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on. 2010. IEEE.

[10] Zou, S., et al., The Performance Analysis and Parallel Optimization of the OpenFOAM-Based Viscoelastic Solver for Heterogeneous HPC Platforms. Applied Mechanics and Materials, 2014. 670: p. 873.