Optimisation of LESsCOAL for large-scale high-fidelity simulation of coal pyrolysis and combustion

Kaidi Wan[1,2], Jun Xia[2], Neelofer Banglawala[3], Zhihua Wang[2], Kefa Cen[2]

1. State Key Laboratory of Clean Energy Utilization,
Zhejiang University, Hangzhou 310027, China

2. Department of Mechanical, Aerospace and Civil Engineering & Institute of Energy Futures,
Brunel University London, Uxbridge UB8 3PH, UK

3. Edinburgh Parallel Computing Centre (EPCC),
The University of Edinburgh, Edinburgh EH8 9YL, UK

Abstract

The present report summarizes the work done and results obtained during the eCSE05-13 project to optimize the LESsCOAL (**L**arge-**E**ddy **S**imulation**s** of **COAL** combustion) code. The project aims to significantly enhance the parallel code performance of LESsCOAL, which is a dedicated numerical solver for pulverised-coal combustion, on massively parallel supercomputers such as ARCHER. Specifically, the target is to achieve 80% of the theoretical parallel efficiency when up to 3,000 computing cores are used on ARCHER.

To achieve the project aim, the major work includes to upgrade the domain decomposition approach to a fully three-dimensional one, implement new MPI and FORTRAN functionalities, and upgrade the pressure, radiation, and particle modules of the code. These 3 modules are the major hurdles identified to degrade the scaling performance. After adopting the optimization strategies, the parallel efficiency of LESsCOAL on ARCHER has been significantly improved and the aim of the project has been achieved.

Keywords: Parallel efficiency; MPI; HYPRE; Discrete ordinates; Load balancing.

1. Introduction

Coal is the most abundant fossil fuel on earth, accounting for over half of the fossil fuel reserve. About 25% of the electricity in the UK is produced by coal-fired power stations. In China the figure is 70%. It is clear that proper prediction, control and optimisation of pulverised-coal combustion characteristics are strategically important for both the UK and China. Poor optical access to coal-fired furnaces makes it difficult to apply advanced laser diagnostics, which has been widely employed in gas combustion experiments. With rapid development of computing capacity, numerical approaches have become an important and effective research tool in this area, especially high fidelity simulation enabled by high-performance computing. Development of high-performance software tools for coal combustion will enable computational study of advanced clean coal technologies such as coal/biomass co-firing and oxy-coal combustion.

The research work in this project aims to significantly enhance the parallel code performance of LESsCOAL (**L**arge-**E**ddy **S**imulation**s** of **COAL** combustion), which is a dedicated numerical solver

for pulverised-coal combustion, on massively parallel supercomputers such as ARCHER. With the project objectives achieved, we will be able to perform large-eddy simulation (LES) of pulverised-coal combustion in a large domain such as a laboratory-scale or a small-scale industrial furnace, greatly facilitating numerical experiments of coal combustion. Development and optimization of such a high-performance software tool can benefit a variety of academic researchers working on turbulent multiphase flow and combustion.

To achieve the project aim, the major work includes to upgrade the domain decomposition approach to a fully three-dimensional one, and upgrade the pressure, radiation, and particle modules of the code. These 3 modules are the major hurdles identified to degrade the scaling performance. The original objectives of the present work are listed in order of priority as follows:

(1) Develop and implement a new parallel particle-tracing algorithm to radically improve the load balance among processor cores.

(2) Implement a three-dimensional domain decomposition approach.

(3) Improve the pressure solver, considering both robustness and efficiency.

(4) Improving the initialisation of the iteration in the radiation module.

(5) Implement new MPI and FORTRAN functionalities.

2. Overview of the optimization work

2.1. Parallelization efficiency of LESsCOAL before optimization

The software environments of the DNS/LES code LESsCOAL include an Intel FORTRAN compiler, HYPRE (a solver for large, sparse linear systems of equations [1]), FFTW and LAPACK. Its parallel computing architecture is based on MPI communications.

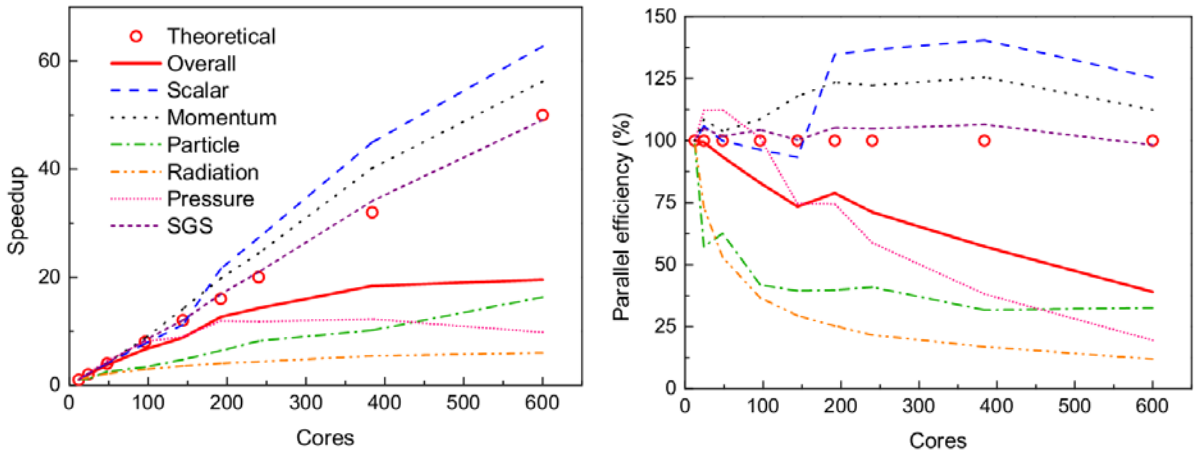


Figure 1. The scaling performance of LESsCOAL on ARCHER before optimization: speedup (left) and parallel efficiency (right) vs. number of cores.

LESsCOAL consists of 6 main modules: (1) the **momentum** module to solve the Navier-Stokes equations, (2) the **scalar** module to transport gas species and temperature, (3) the **particle** module to trace coal particles, (4) the **radiation** module, (5) the **pressure** module to solve the Poisson equation, and (6) the **SGS** module to calculate SubGrid-Scale model terms. Figure 1 shows the code performance on ARCHER before optimization in a strong scaling test with 10 million grid cells and more than 0.6 million particles. The detailed scaling of the 6 major modules is shown with the overall

scaling of the code (red solid line) and the theoretical scaling (red circle). It can be seen that the scalar, momentum and SGS modules have a very good speedup, while the others (the particle, radiation and pressure modules) show poor scaling performance at large core numbers. Overall, the code has a satisfactory scaling up to 200 cores, but its performance becomes unacceptable when more cores are used. The speedup of the pressure module even decreases after 400 cores.

2.2. The optimization strategies adopted

In order to improve the parallel efficiency of LESsCOAL and achieve the objectives of the present project, following optimization strategies have been employed:

(1) New MPI and FORTRAN functionalities such as One-Sided Communications (provided in MPI-2), non-blocking collectives (MPI-3) and C-like pointers (FORTRAN 2008) have been implemented. These upgrades are helpful to shorten the communication latency and therefore improve the parallel efficiency of the code.

(2) The domain decomposition approach has been upgraded from two-dimensional to a fully three-dimensional decomposition. The fluid variable information can only be transferred to adjacent sub-domains in one iteration. When more cores are employed, we need more iterations to transport the information from the domain boundary to inner sub-domains, which is a disadvantage to the scaling performance. The poor scaling performance of the pressure and radiation modules can be largely due to the inefficient transfer. With a 3D decomposition method, the core numbers in one dimension (with the same total core number) becomes much less, which means physical information of the fluid phase can be transferred more efficiently inside the computational domain.

(3) The pressure module has been optimized. As the pressure equation is solved by HYPRE, in total 14 different combinations of available parallel solvers and preconditioners in HYPRE for solving large, sparse linear systems of equations have been tested. Considering both the scalability and stability, the GMRES-PFMG and PCG-PFMG methods are found to be the optimized ones. (GMRES and PCG are the solvers and PFMS is the preconditioner.) The detailed technical procedure will be discussed in Section 3.

(4) For the radiation module, two different optimization approaches, i.e., the optimization method 1 (optimized computation sequence of radiation rays) and the optimization method 2 (the diagonal slicing method), have been attempted. After comparing the two optimized methods, the optimization method 1 is found to be more suitable for the pulverized-coal combustion cases which we considered; while the diagonal slicing method should have a better performance for a computational domain like a tube. More details about the optimization of the radiation module will be discussed in Section 4.

(5) For the particle module, a new parallel algorithm of particle phase, i.e., the OhHelp (One-handed Help) method, has been implemented into the new code to solve the load imbalance issue in parallel solving of the particle phase. After the optimization, particles can be transferred from high-load CPU cores to low-load CPU cores to keep the overall load (i.e., number of particles) of each CPU core dynamically balanced. The parallel efficiency of the particle module is therefore largely improved after the optimization. More technical details will be presented in Section 5.

3. Optimization of the pressure module

The pressure equation is a Poisson's equation:

$$\frac{\partial^2 P}{\partial x^2}+\frac{\partial^2 P}{\partial y^2}+\frac{\partial^2 P}{\partial z^2}=S \tag{1}$$

where *P* is the pressure which needs to be solved, and *S* represents the RHS term.

The equation is solved by calling HYPRE, which is an open-source software package designed for solving large, sparse linear systems of equations on massively parallel computers. Before the optimization work, LESsCOAL employs the SMG multi-grid solver in HYPRE, which is a particularly robust but relatively slow method. HYPRE also provides other solvers, e.g., PFMG, PCG, GMRES, BICGSTAB, and HYBRID, and preconditioners, e.g., DIAGONAL and PFMG. (PFMG and SMG can be used as solvers or preconditioners).

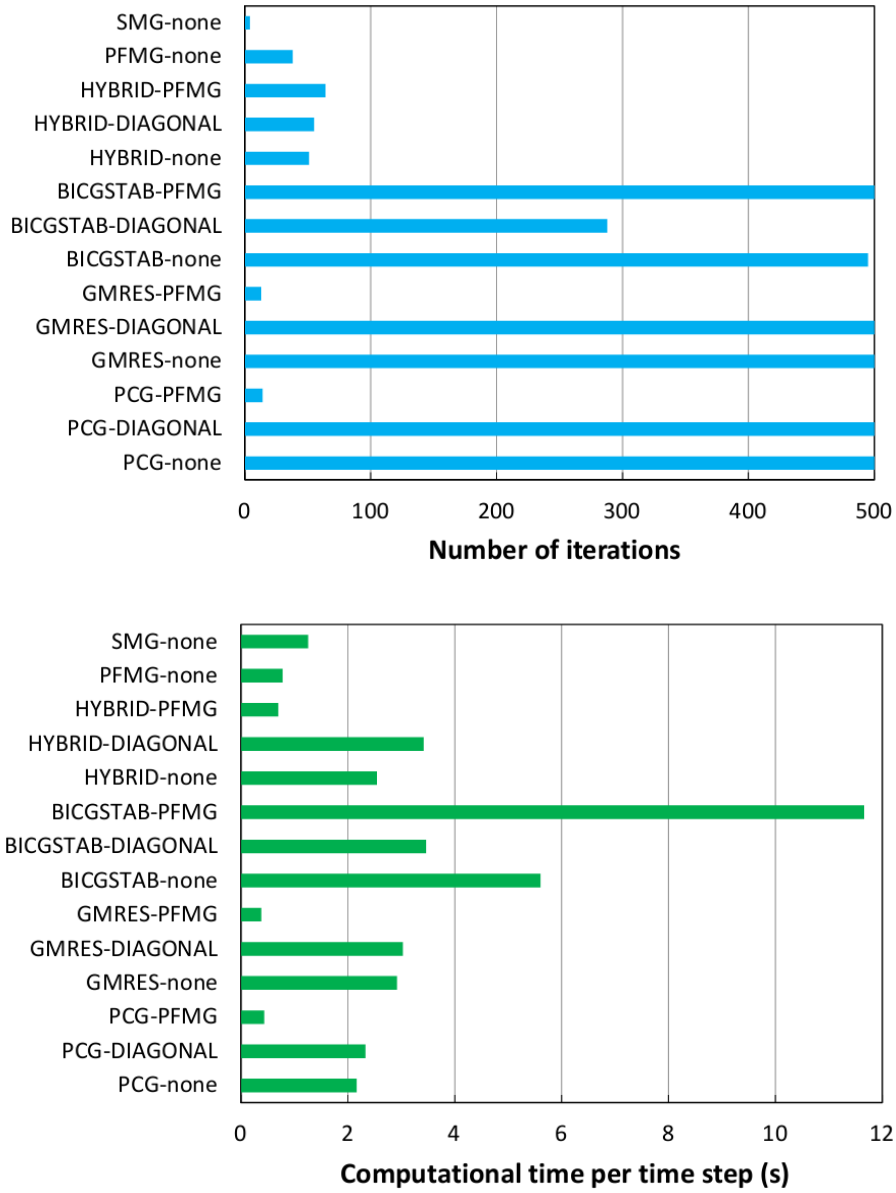


Figure 2. Number of iterations and computational time per time step of the pressure module with various solvers and preconditioners.

In order to explore the performance of different solvers and preconditioners on solving the pressure equation in LES of pulverized-coal combustion, 14 setups with different solvers and preconditioners were selected for testing and comparison. Each case was run on a 10-million-cells grid

with 192 CPU cores. The convergence criterion for the iterative solution of the pressure equation was that the residual was reduced to below 10^{-5}. At the same time, the maximum number of iterations is limited to 500. The results are shown in Fig. 2. The labels on the left represent the solver and preconditioner used in each case. For example, “SMG-none” means the SMG solver was used with no preconditioner; “HYBRID-PFMG” means the HYBRID solver and PFMG preconditioner were employed. It can be found that although “SMG-none” has the least number of iterations, its computational time is not the shortest. The least time consuming methods are “GMRES-PFMG” and “PCG-PFMG” (Fig. 2). This reflects that each iteration of “SMG-none” consumes a relatively longer time. Therefore although the number of iterations is small, but the total iteration process of “SMG-none” costs a long time. After comparison, the “GMRES-PFMG” and “PCG-PFMG” methods in HYPRE are selected to solve the pressure equation in LESsCOAL.

4. Optimization of the radiation module

LESsCOAL simulates the radiative heat transfer in pulverized-coal combustion using the discrete ordinates method (DOM) with the S_4 angular discretization scheme, in which the entire space is discretized into 24 directions. The discretized radiative heat transfer equation solved in LESsCOAL reads:

$$I = \frac{|\mu_m| AI_{x,r} + |\xi_m| BI_{y,r} + |\eta_m| CI_{z,r} + \alpha S}{|\mu_m| A + |\xi_m| B + |\eta_m| C + \alpha\left(k + k_p + \sigma_p\right)} \tag{2}$$

where I is the radiation intensity of the cell center; A, B and C are the difference coefficients; μ_m, ξ_m and η_m are the angular coefficients of the S_4 scheme. S represents the following term:

$$S = k\frac{\sigma\tilde{T}^4}{\pi} + E_p + \frac{\sigma_p}{4\pi}\int_0^{4\pi} I\left(\vec{r},\vec{s}'\right)\Phi\left(\vec{s}\cdot\vec{s}'\right)\mathrm{d}\Omega' \tag{3}$$

$I_{x,r}$, $I_{y,r}$ and $I_{z,r}$ are the radiation intensities of the cell face in the incoming ordinate direction. They are related to the radiation intensity of the cell center by the following equation:

$$I = \alpha I_{i,e} + \left(1-\alpha\right) I_{i,r} \quad (i=x,y,z) \tag{4}$$

where $I_{x,e}$, $I_{y,e}$ and $I_{z,e}$ are the radiation intensities of the cell face in the outgoing ordinate direction. α is the finite-difference weighting factor. The common values for α are 1/2 and 1. Lathrop’s diamond-difference scheme is obtained if α = 1/2 and is globally second-order accurate. When α = 1, first-order upwind scheme is obtained. To ensure the stability of the solution, the first-order upwind scheme with α = 1 is adopted in the present work.

From Eq.(2), the radiation intensities $I_{x,r}$, $I_{y,r}$ and $I_{z,r}$ in the incoming ordinate direction, i.e., the upwind direction, of the radiation intensity is required to solve the radiation intensity I. In the case of serial computation using a single CPU core, for any direction of radiation, we can start from the most upstream boundary grid point, and then sequentially visit each grid point in the direction that the radiation beam propagates with using Eq.(2) to get the radiation intensity I. After completing the calculation of the radiation in the 24 directions, i.e., 24 radiation rays, the new radiation intensity value is used to update the radiation scattering term, which is the last term in the right-hand side (RHS) of Eq.(3). Then the 24 radiation ray equations are iteratively computed until convergence to get the final radiation intensity field.

When multiple CPU cores are used for parallel computation, the radiated intensity data of the upstream CPU core is required for the inner central subdomain to complete the calculation [2]. In

other words, the solving method requires each CPU core to wait for all data to become available on its upwind boundaries for each ordinate direction before it can begin calculations, which means the method is inherently serial. Alternatively, the downstream CPU core will not wait for the data from upstream CPU cores, but directly use the old data from previous iteration as an initial guess. Since the last term in the RHS of Eq.(3) couples the radiation intensity of the 24 rays together, this alternative method helps to accelerate the coupling between the 24 radiation rays, and therefore is more efficient to achieve convergence.

To improve the parallel efficiency of radiation calculation, the following two optimization strategies have been implemented and compared: (1) adjust the computation sequence of the 24 radiation rays in different CPU cores in the order to optimize the transport of radiation information from the domain boundary to inner subdomains, and (2) the diagonal slicing method, which performs multiple communications in each iteration to optimize the transport of radiation information from the domain boundary to inner subdomains.

4.1. Optimization method 1: optimized computation sequence of radiation rays

After using the S_4 angular discretization scheme, we need to calculate the radiation by 24 rays, as shown in Table 1. The 24 different radiation rays can be divided into 8 groups based on its radiation direction:

Group (1): Ray number 13, 17 and 18 with positive numbers for μ_m, ξ_m, η_m, which means the radiation direction is x, y, z = (+, +, +);

Group (2): Ray number 14, 19 and 20, and the radiation directions is x, y, z = (+, +, –);

Group (3): Ray number 2, 7 and 8, and the radiation directions is x, y, z = (+, –, –);

Group (4): Ray number 1, 5 and 6, and the radiation directions is x, y, z = (+, –, +);

Group (5): Ray number 4, 11 and 12, and the radiation directions is x, y, z = (–, –, –);

Group (6): Ray number 3, 9 and 10, and the radiation directions is x, y, z = (–, –, +);

Group (7): Ray number 15, 21 and 22, and the radiation directions is x, y, z = (–, +, +);

Group (8): Ray number 16, 23 and 24, and the radiation directions is x, y, z = (–, +, –).

Table 1. Angular coefficients of the S_4 scheme

Ray number	μ_m	ξ_m	η_m
1	+0.9082483	–0.2958759	+0.2958759
2	+0.9082483	–0.2958759	–0.2958759
3	–0.9082483	–0.2958759	+0.2958759
4	–0.9082483	–0.2958759	–0.2958759
5	+0.2958759	–0.2958759	+0.9082483
6	+0.2958759	–0.9082483	+0.2958759
7	+0.2958759	–0.9082483	–0.2958759
8	+0.2958759	–0.2958759	–0.9082483
9	–0.2958759	–0.2958759	+0.9082483
10	–0.2958759	–0.9082483	+0.2958759
11	–0.2958759	–0.9082483	–0.2958759
12	–0.2958759	–0.2958759	–0.9082483
13	+0.9082483	+0.2958759	+0.2958759
14	+0.9082483	+0.2958759	–0.2958759
15	–0.9082483	+0.2958759	+0.2958759

16	–0.9082483	+0.2958759	–0.2958759
17	+0.2958759	+0.2958759	+0.9082483
18	+0.2958759	+0.9082483	+0.2958759
19	+0.2958759	+0.9082483	–0.2958759
20	+0.2958759	+0.2958759	–0.9082483
21	–0.2958759	+0.2958759	+0.9082483
22	–0.2958759	+0.9082483	+0.2958759
23	–0.2958759	+0.9082483	–0.2958759
24	–0.2958759	+0.2958759	–0.9082483

Before optimization, the 24 rays are computed in the same order for different CPU cores. After optimization, in each iteration the computation sequence of radiation rays in different CPU cores is adjusted according to the radiation direction of one selected group of rays in order to improve the transport efficiency of radiation information of the selected group between different subdomains. To facilitate the discussion, a two-dimensional computational domain is employed as an example, as shown in Fig. 3. Assuming that the first group of rays with the ray direction $x, y = (+, +)$ is selected to be optimized, then the computational sequence of the 24 rays in different CPU cores can be determined as follows. The CPU core #1 in Fig. 3 puts the first group of rays in the top of the computational sequence, and therefore the sequence should be 13, 17, 18, ... The CPU core #2 firstly compute a different ray and then the first group of rays, which leads to the sequence to be *X*, 13, 17, 18, ... (where *X* represents a different ray). The computational sequence of the CPU core #3 is *X*, *X*, 13, 17, 18, ... The computational sequence of the rest CPU cores can be similarly deduced. After the above-mentioned optimization, the calculation process in different CPU cores is as follows. First, each core calculates the first ray, which means the CPU core #1 calculates the Ray number 13. Then after a MPI communication, each core calculates the second ray, where the CPU core #2 calculates the Ray number 13, and it has already obtained the required data from the upstream CPU core #1. It can be seen that the radiation information can be transported downstream once per ray calculation; while before optimization the transport frequency of the radiation information is once per 24 ray calculations. After optimization, the transport efficiency of the radiation information in the computational domain can be largely improved, and therefore the parallel efficiency of the radiation module is optimized.

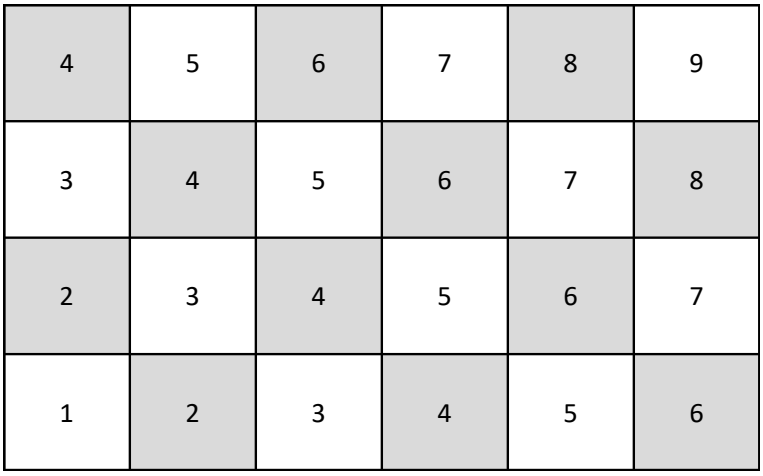


Figure 3. 2D domain for showing the optimized computation sequence of radiation rays.

4.2. Optimization method 2: the diagonal slicing method

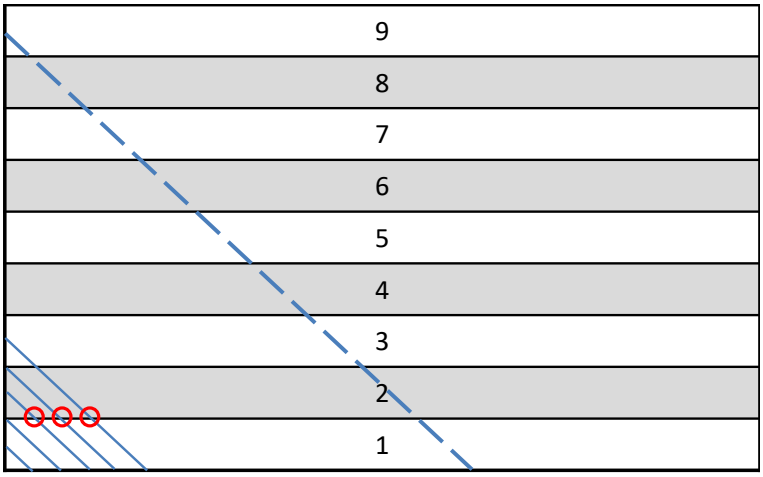


Figure 4. 2D domain for showing the diagonal slicing method.

The diagonal slicing method [3, 4] employs a different approach to optimize the transport of radiation information in the computational domain. Similar as in the previous section, a two-dimensional computational domain is employed to facilitate the discussion, as shown in Fig. 4. The diagonal slicing method requires that the dimension of the parallel decomposition of the computational domain is one dimensional lower than the dimension of the domain. Specifically, 2D domain decomposition should be used for a 3D computational domain; while 1D domain decomposition should be used for a 2D computational domain. For the first group of rays (ray grouping see Section 4.1), the ray direction is x, y = (+, +). It can be found that the grid points on the diagonal lines are independent of each other, and they are in the same downstream location of the radiation direction. Therefore, in the diagonal slicing method, the grid points are sequentially visited in a diagonal way. Take the CPU core #1 as an example, it visits the grid points sequentially with the order showing by the blue line in Fig. 4, starting from the most upstream boundary point in the lower left corner; while the other CPU cores (#2-9) are idle and wait. When Core #1 reaches the boundary of the subdomain shared with core #2, the boundary data is stored in an array, and then the calculation continues. As core #1 visits more grid points, it gets more boundary data. When the size of the boundary data reaches a preset value (named Nblock), core #1 packs this part of the boundary data, e.g., the grid points of the three red circles in Fig. 4, and sends it to core #2 via MPI communications. Once the core #2 receives this part of the boundary data, it can immediately starts computation and store the boundary data in the same way, and then send it to core #3. The frequency of MPI communication between CPU cores can be adjusted by changing the value of Nblock. In one extreme case, when Nblock value is 1, once each CPU core gets a boundary data, the data will immediately be sent to the corresponding downstream core. In this case, when the code runs to the blue dash line in Fig. 4, all the cores are involved in the radiation computation, and the dash line represents the grid points which being computed at different CPU cores. At the other extreme case, Nblock value is the total number of boundary grid points of a single core, and each CPU core only sends the boundary data to the downstream core after its calculation is completed. Ideally we would expect the value of Nblock to be as small as possible because the smaller the value of Nblock means a more frequent

transport of radiation information inside the computational domain, which helps to reduce the waiting time of the downstream cores. However, in practice, too frequent MPI communication will also reduce the parallel efficiency of the program. Therefore, the optimal value of Nblock is a trade-off between the cost of MPI communication and the waiting time of the downstream CPU cores.

The main purpose of the diagonal slicing method is to enhance the transport of radiation information to the downstream CPU cores. As shown in Fig. 4, core #9 obtains the required boundary data when core #1 completes about 2/3 of the computational tasks after the diagonal slicing method is employed. However, in the original method, it takes 8 iterations to transport the radiation information to the core #9.

4.3. Optimization performance of the radiation module

First, for the optimization method 2, as described in Section 4.2, it is necessary to find out the optimal value of Nblock. Figure 5 shows the average computational time per time step of the radiation module optimized with the diagonal slicing method under different Nblock values. Each test case employs a 10-million-cell grid and 48 CPU cores. It can be found that either the Nblock value is too small or too large will lead to an increase of the computational time. Finally, the Nblock value is set to 50, under which the calculation of the radiation module takes the least time.

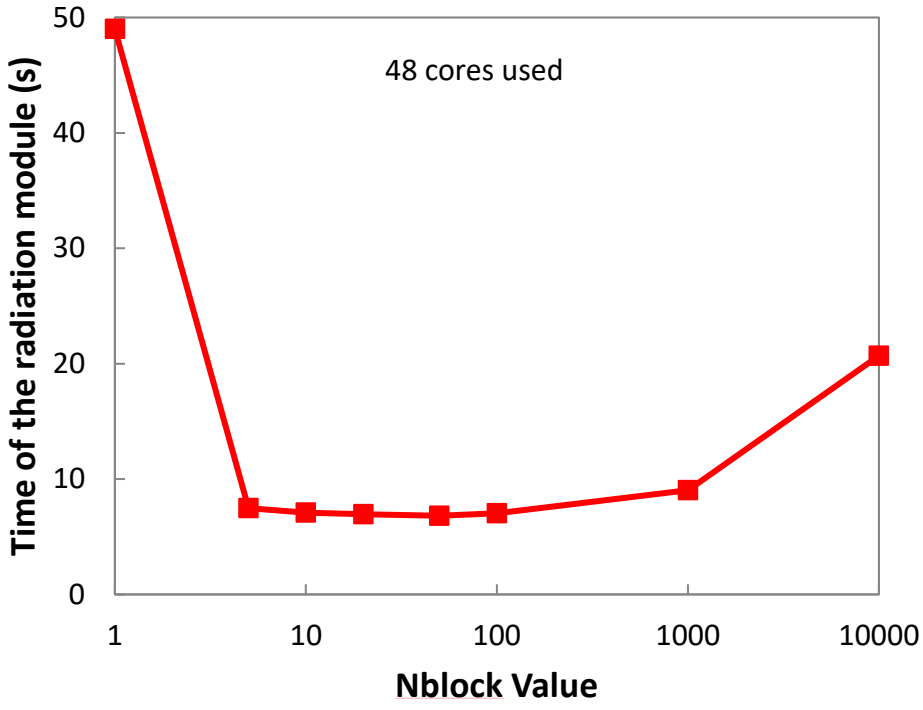


Figure 5. The average computational time per time step of the radiation module with different Nblock values in the diagonal slicing method.

Figure 6 compares the average computational time per time step of the radiation module using different number of cores in the original method, the optimization method 1 (optimized computation sequence of radiation rays) and the optimization method 2 (the diagonal slicing method). The grid cell number of each test case is 10 million and remains constant when different number of CPU cores are employed. Comparing the results of the original method and the optimization method 1, it can be

found that when the number of CPU cores used is small (such as 48 cores), the optimization effect of the method 1 is not obvious. However, with the increase of the CPU cores used, the optimization effect becomes more remarkable. When using 3072 cores, the computational time of the method 1 is 60% shorter than the original method, which means the parallel efficiency of the radiation module has been significantly improved.

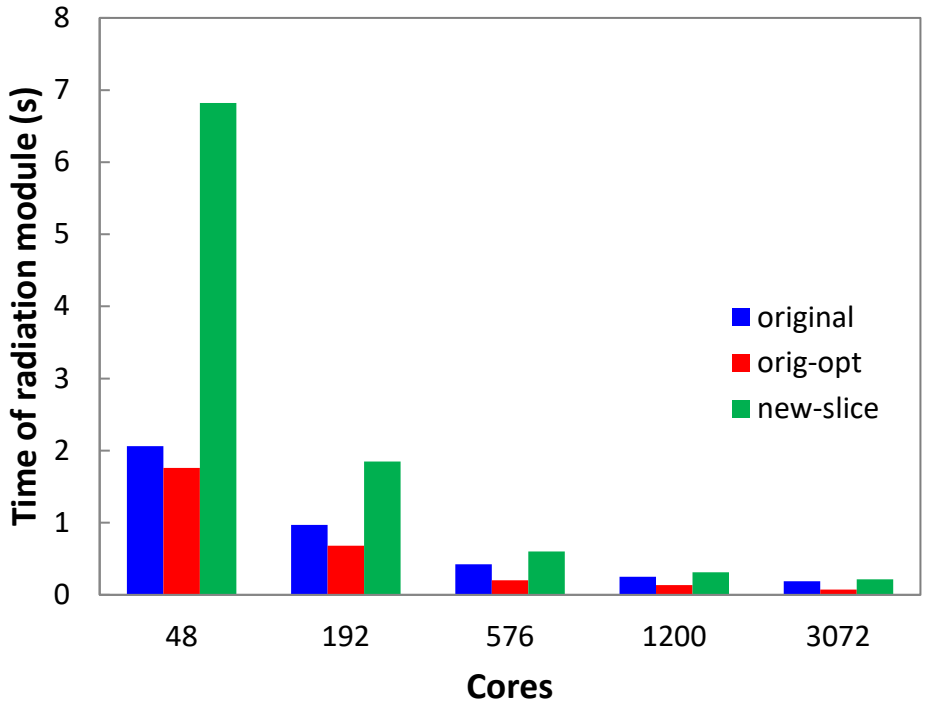


Figure 6. The comparison of the average computational time per time step of the radiation module using different number of cores in the original method, the optimization method 1 (orig-opt) and the optimization method 2 (new-slice).

The performance of the diagonal slicing method is very interesting. When the number of cores is small (such as 48 cores), the computational time of the radiation module in method 2 is much longer than that in the original method. But as more number of cores are used, its computational time decreases sharply. When using 3072 cores, the computational time of method 2 is reduced to a similar level but slightly higher than that of the original method. The reason is that in method 2 each CPU core needs to wait for the boundary data sent by its upstream CPU core before it can participate in the computation task. This waiting process affects the efficiency of the method 2. On the other hand, in method 2 the gird points are visited in the order of diagonal slices, while the corresponding data is stored in memory with the order of Cartesian coordinates. Therefore, the efficiency of memory access in method 2 is lower, thereby affecting the efficiency of the radiation module.

After the above-mentioned comparison, the optimization method 1 is found to be the least time consuming method when using 3072 cores. However, it should be noted that although the performance of the optimization method 2 in the present test is not satisfactory, the parallel acceleration of the method 2 is very good. It may achieve a better performance when more CPU cores are used. On the other hand, it can be seen from Fig. 4 that the method 2 is actually more suitable for computational

domains like long pipes. When the number of grid points in one dimension is very large, the chance of all the CPU cores work together in the method 2 will greatly increase, which is very beneficial to improve the parallel efficiency of the method.

5. Optimization of the particle module

There are mainly two parallel strategies, i.e., particle decomposition and domain decomposition, for parallel computing of gas-solid two-phase Euler-Lagrangian simulation. When using the particle decomposition strategy, all the particles in the computational domain are divided into a series of equal-sized subgroups according to the number of CPU cores and assigned to each CPU core. Each CPU core requires the access to the gas flow field data in the whole domain. This strategy can ensure the perfect load balance between the CPU cores, and also does not need to transfer the particle information among the CPU cores. However, this strategy requires that every CPU core be able to obtain all the gas flow field data. For a distributed memory supercomputer using MPI communication architecture, it means that each CPU core needs to store a copy of all the gas flow field data in the local memory, which results in a high demand for memory. In general, the particle decomposition strategy is suitable when the computational cost of particle phase is much larger than that of the gas phase.

The domain decomposition strategy is the parallel strategy adopted by LESsCOAL before optimization. When the domain decomposition strategy is adopted, the computational domain is divided into a series of subdomains and assigned to each CPU core. At the same time, the particles located in each subdomains are also assigned to the corresponding CPU core. When the particles move through the subdomain boundaries, MPI communication is employed to send the particles to adjacent CPU cores. However, if the particle phase is not uniformly distributed in the computational domain, which means the number of particles in different subdomains can be significantly different, this strategy will lead to highly load imbalance between the CPU cores and therefore reduce parallel efficiency. In general, the domain decomposition strategy is suitable when the particle phase is uniformly distributed in the physical space or the computational cost of the gas phase is much larger than that of the particle phase so that the load imbalance issue of the particle phase becomes a very minor one.

5.1. OhHelp method

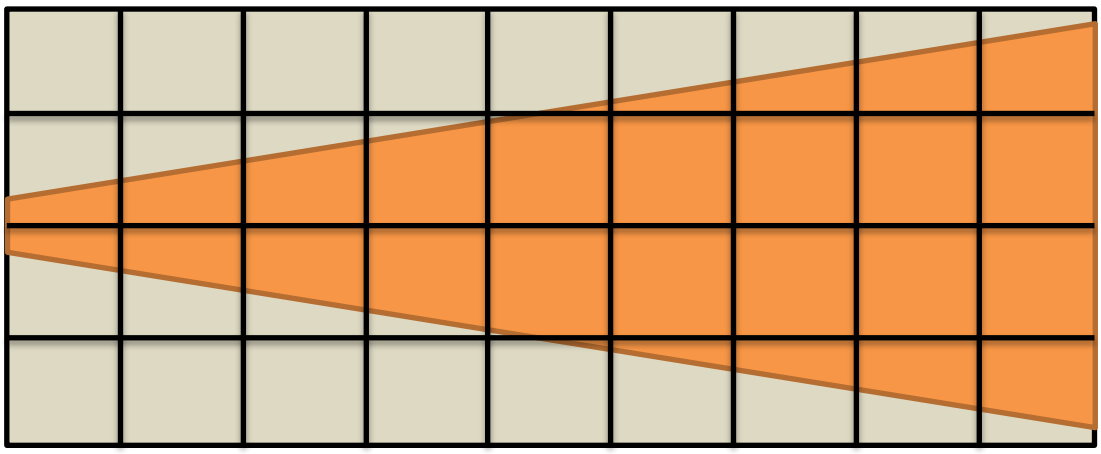


Figure 7. Schematic diagram of the load imbalance issue when domain decomposition strategy is employed for a gas-solid two-phase jet.

For LES of gas-solid two-phase turbulent jet, the computational cost of the gas phase is dominant, but the particle phase also contributes a considerable proportion. So the particle decomposition strategy is not suitable here, while the parallel efficiency of the domain decomposition strategy will be affected by the load imbalance of particle phase. A typical gas-solid two-phase jet case with domain decomposition is shown in Fig. 7, where each black square represents a subdomain and the orange zone represents the distribution range of most particles. It can be seen that the distribution of the particle phase in the computational domain is quite uneven. Most of the particles are located near the jet centerline. To improve the parallel efficiency of the particle module, a new parallel strategy, named OhHelp (One-handed Help), has been implemented into LESsCOAL. Based on the domain decomposition strategy, the OhHelp method can also ensure the load balance between CPU cores for the particle phase, and therefore achieve good scalability. Its load balancing mechanism can be summarized as follows:

(1) The computational domain of the gas phase is divided into a series of subdomains in accord with the domain decomposition strategy and assigned to each CPU core as the primary subdomain.

(2) When some of the subdomains contain too many particles (i.e. the particle load imbalance exceeds the preset limit), each CPU core except the root core is assigned another subdomain as the secondary subdomain of the CPU core.

(3) A subgroup of the particles will be transferred from the primary subdomains of heavily loaded CPU cores to the secondary subdomains of lightly loaded CPU cores to ensure that the number of particles between the CPU cores are balanced.

The detailed introduction and explanation of the OhHelp method can be found in [5] by Nakashima et al., the developer of the method.

5.2. The implementation of OhHelp into LESsCOAL

In Nakashima et al. [5], the OhHelp method is used for parallel simulation of plasmas; here, the method is first applied to the numerical simulation of gas-solid two-phase turbulent flow. After employing OhHelp, the method is mainly used to investigate the load imbalance between the CPU cores and provide the particle transfer table. Then according to the table, LESsCOAL performs the particle transfer between the CPU cores via MPI communication to achieve load balancing. Moreover, because of the two-way coupling between the gas phase and the particle phase, the gas phase information at the physical location of a particle is required when solving the particle phase equations. Therefore, the corresponding gas phase data should be sent along with the particle phase data. Since in the OhHelp method, each CPU core is only responsible for one primary subdomain and one secondary subdomain (except that the root core has no secondary subdomain), here the required gas phase data of all the grid points in the primary subdomain is copied to the corresponding secondary subdomains of other CPU cores. When solving the particle equations, the two-way coupling source terms of the secondary particles i.e., the particles belong to the secondary subdomain, should be stored separately. The source terms of the secondary particles should be transferred back to the corresponding primary subdomain to project on the correct physical location, after all the particles have been advanced.

The implementation of the OhHelp method significantly improves the load imbalance issue of the particle module. Figure 8 compares the number of particles loaded in each CPU core in the original method (before optimization) and the optimized method (with OhHelp). The total number of particles in the computational domain is ~2 million, and 3072 CPU cores are used in the test case. It can be seen that the number of particles loaded in each CPU core is significantly unbalanced when the original

domain decomposition strategy is adopted, as shown by the blue line. After the OhHelp method is involved, the load imbalance issue disappears and the number of particles loaded in each CPU core is almost the same, as shown by the red line.

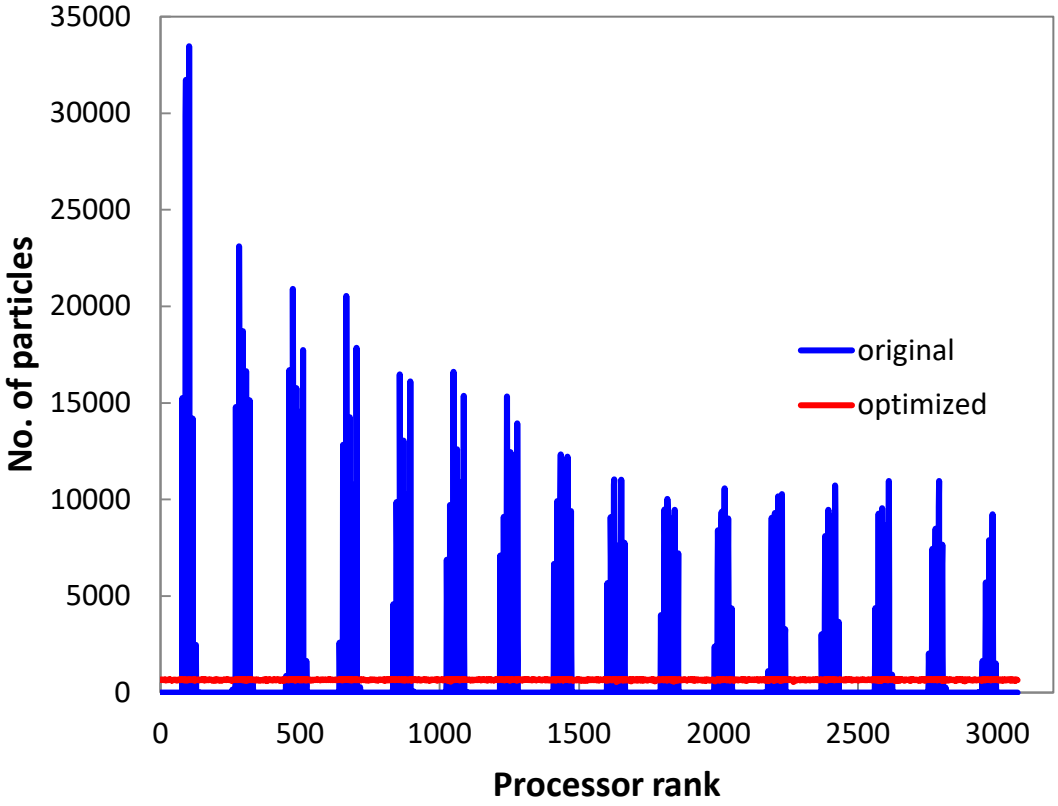


Figure 8. Comparison of the number of particles loaded in each CPU core in the original method and the optimized method.

Figure 9 compares the average computational time per time step of the particle module in the original method and the optimized method using 48 and 3072 CPU cores. It can be found that the computational time of the particle module is almost reduced by an order of magnitude after the optimization, using either 48 or 3072 cores. Since the implementation of the OhHelp method provides a very good load balance between the CPU cores, the efficiency of the particle module is significantly improved.

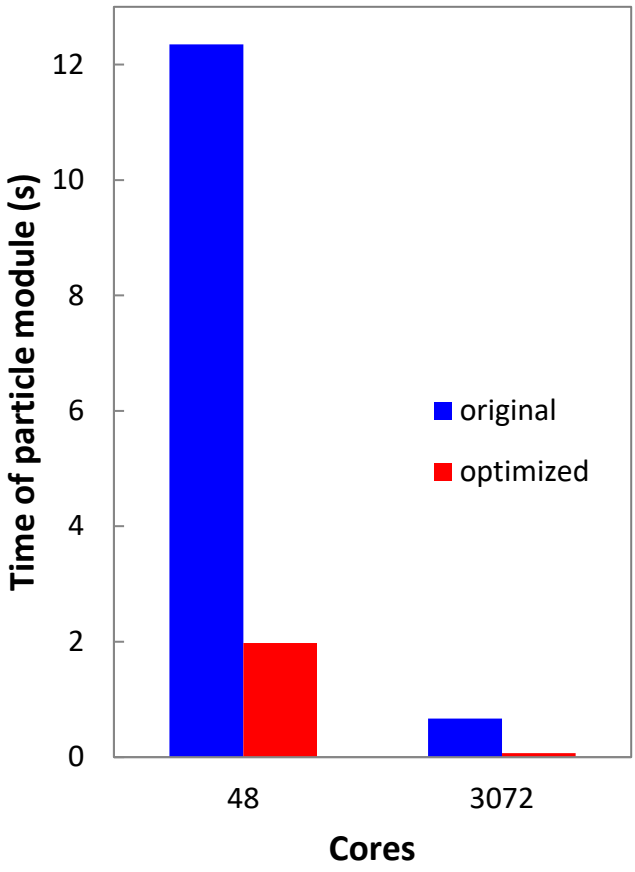


Figure 9. Comparison of the average computational time per time step of the particle module in the original method and the optimized method.

5.3. Optimization of bookkeeping using MPI one-sided communications

In the particle transfer scheme, a bookkeeping step is first performed where all cores send the information about the number of particles which will be transferred to the destination core [6]. Typically, MPI collective communication functions, e.g., MPI_Allgather, MPI_Allreduce and MPI_Alltoall, are employed for the bookkeeping step. However, the MPI collective operations can be expensive when large number of cores are used. Especially in the bookkeeping step, the size of the particle transfer schedule data which needs to be communicated grows as the square of the number of cores. To solve this problem, MPI one-sided communications, which is scalable with large number of cores, are implemented for the bookkeeping step.

Figure 10 compares the average computational time per time step of the particle module with different core numbers using the MPI collective communication function MPI_Allgather and the MPI one-sided communication function MPI_Put for the bookkeeping step. It can be seen that when the number of cores is small, e.g., 576 and below, there is no obvious difference in the computational time between collective and one-sided communications. However, when the number of cores used is larger than 1200, the drawbacks of collective communications begins to unfold. When 3072 cores are employed, one-sided communications result in an order of magnitude faster than collective communications, thereby significantly improve the parallel efficiency of the particle module.

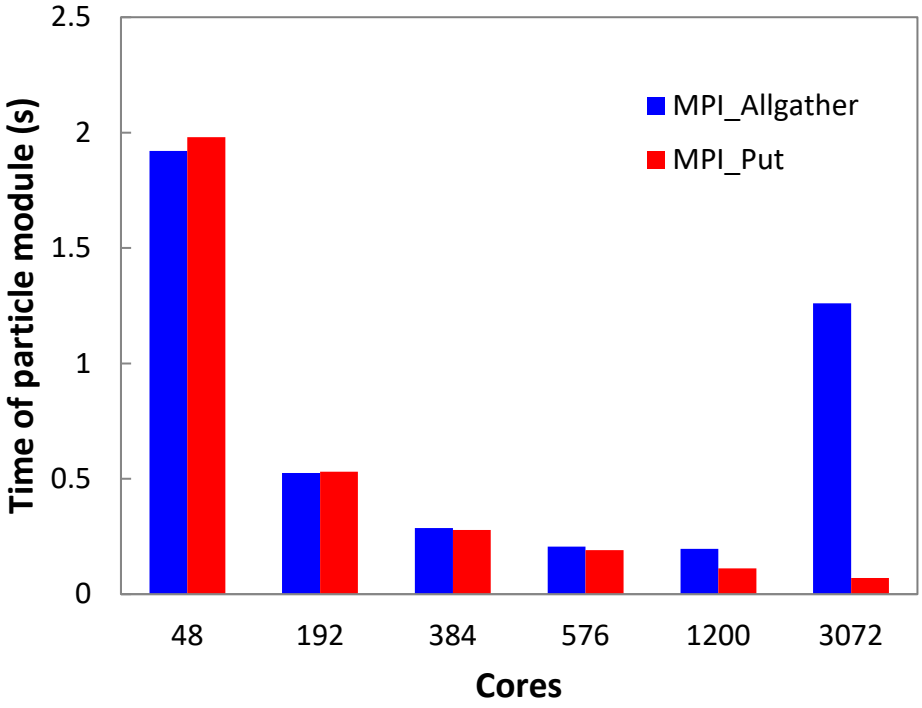


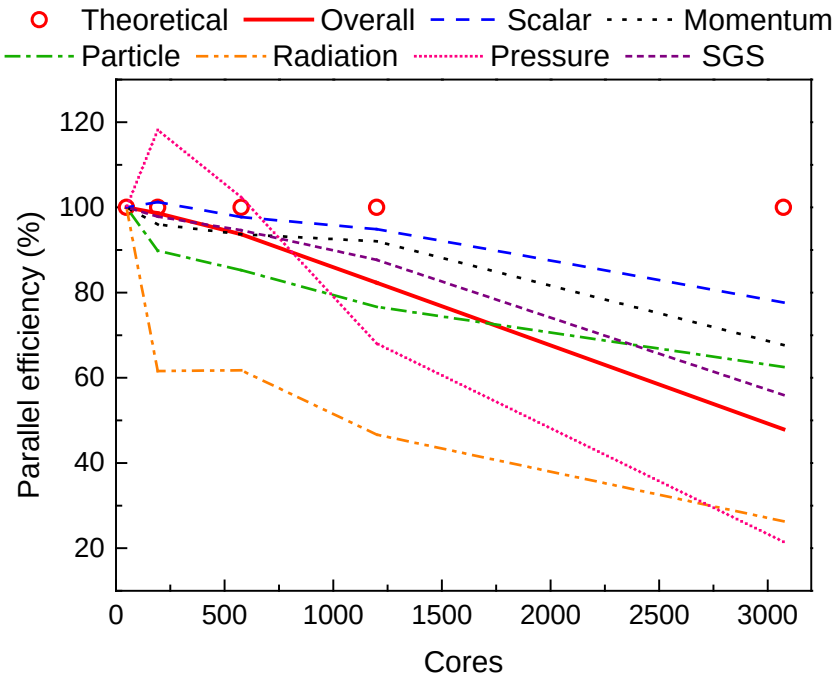
Figure 10. Comparison of the average computational time per time step of the particle module using the MPI collective communication function MPI_Allgather and the MPI one-sided communication function MPI_Put.

6. Parallel performance after optimization

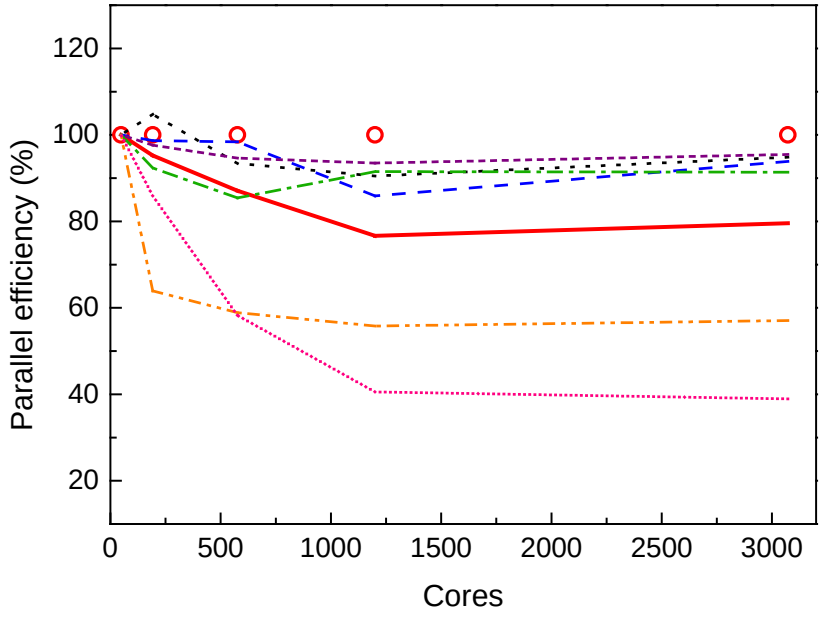
After adopting the abovementioned optimization strategies, the parallel efficiency of LESsCOAL on ARCHER has been significantly improved. Figure 11 shows the scaling performance of the overall code and the major modules (scalar, momentum, etc.) in both the strong and weak scaling tests. The average computational time per time step using 48, 192, 576, 1200, 3072 cores is measured and converted to obtain the parallel efficiency with the computational time using 48 cores as the benchmark.

For the strong scaling test (Figure 11a), a 10-million-cell grid and ~10 million particles are employed. It can be found that the optimized LESsCOAL can achieve good parallel efficiency (> 82%) when using no more than 1200 cores. However, when using 3072 cores, the parallel efficiency of the code dropped to 48%. This is mainly because the number of grid cells and particles assigned to each CPU core is too small (both ~3255), resulting in a relatively high communication/computation ratio.

For the weak scaling test (Figure 11b), the number of grid cells and particles are both fixed at 25,000 per CPU core. Compared with the strong scaling test, the weak scaling test is more reasonable for the evaluation of parallel efficiency of LES simulations. Because usually small-scale LES uses fewer CPU cores, and for large-scale LES more CPU cores are employed. It can be seen that the optimized LESsCOAL code can achieve 80% of the theoretical parallel efficiency when using 3072 cores on ARCHER.



(a) Strong scaling test



(b) Weak scaling test

Figure 11. The scaling performance of the optimized LESsCOAL on ARCHER.

7. Conclusions

The parallel efficiency and optimization work of the LESsCOAL code for LES of pulverized-coal combustion has been presented and discussed. Before the optimization work, the original code has a satisfactory scaling up to 200 cores, but its performance becomes unacceptable when more cores are used. The scalar, momentum and SGS modules of LESsCOAL have a very good speedup, while the others (the particle, radiation and pressure modules) show poor scaling performance at large core numbers. In order to perform LES of pulverised-coal combustion in a large domain such as a laboratory-scale or a small-scale industrial furnace, the parallel code performance of LESsCOAL

needs to be significantly enhanced. The optimization strategies includes:

(1) New MPI and FORTRAN functionalities such as One-Sided Communications (provided in MPI-2), non-blocking collectives (MPI-3) and C-like pointers (FORTRAN 2008) have been implemented.

(2) The domain decomposition approach has been upgraded from two-dimensional to a fully three-dimensional decomposition.

(3) The parallel solvers and preconditioners in HYPRE used for solving the pressure equation has been optimized.

(4) For the radiation module, two different optimization approaches, i.e., the optimization method 1 (optimized computation sequence of radiation rays) and the optimization method 2 (the diagonal slicing method), have been attempted and compared.

(5) For the particle module, a new parallel algorithm of particle phase, i.e., the OhHelp method, has been implemented to solve the load imbalance issue in parallel solving of the particle phase.

After adopting the abovementioned optimization strategies, the parallel efficiency of LESsCOAL on ARCHER has been significantly improved. In the weak scaling test, the optimized LESsCOAL code can achieve 80% of the theoretical parallel efficiency when using 3072 cores on ARCHER, which meets the aim of the project.

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

References

[1] *HYPRE: Scalable Linear Solvers and Multigrid Methods*. Available at http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods.

[2] A.J. Chandy, D.J. Glaze, and S.H. Frankel, *Parallelizing the Discrete Ordinates Method (DOM) for Three-Dimensional Radiative Heat Transfer Calculations using a Priority Queuing Technique*, Numerical Heat Transfer, Part B: Fundamentals 52 (2007), pp. 33-49.

[3] G.J. Pringle, D. Barrett, D. Turland, M. Weiland, and M. Parsons. *Performance and Extension of a Particle Transport Code using Hybrid MPI/OpenMP Programming Models*, Cray User Group 2015, Chicago, United States, April 26-30, 2015.

[4] G.J. Pringle. *Lessons Learnt from Implementing a Mixed-Mode Wavefront Algorithm*, PPAR Lunchtime Seminar Series, Edinburgh, UK, April 8, 2015.

[5] H. Nakashima, Y. Miyake, H. Usui, and Y. Omura. *OhHelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations*, Proceedings of the 23rd international conference on Supercomputing, Yorktown Heights, NY, USA, 2009; 90-99.

[6] H. Sitaraman, and R. Grout, *Balancing conflicting requirements for grid and particle decomposition in continuum-Lagrangian solvers*, Parallel Computing 52 (2016), pp. 1-21.