International Journal of Aeronautical and Space Sciences (2023) 24:1124–1135
https://doi.org/10.1007/s42405-023-00601-8

IJASS International Journal of Aeronautical and Space Sciences

ORIGINAL PAPER

Coarray Fortran Parallel Implementation of a Finite Volume Method-Based Aircraft Ice Accretion Simulation Code

L. Prince Raj[1] · E. Esmaeilifar[2] · B. Sengupta[2] · H. Jeong[2] · R. S. Myong[2]

Received: 17 December 2022 / Revised: 9 March 2023 / Accepted: 1 April 2023 / Published online: 26 May 2023

Abstract
The aircraft industry often uses computational methods to quantify ice accretion, investigate aerodynamic penalties, and conduct certification processes. The computational simulation of aircraft icing is computationally intensive owing to three consecutive runs of air, droplet, and ice accretion solvers. This study developed a parallel code using MPI and Coarray methods to reduce the computation time of an FVM-based ice accretion solver. The computational results were validated by comparison with the experimental data. The parallel performance of the MPI and Coarray methods were compared and found to be similar on the airflow solver. Further, the Coarray-based implementation on the water droplet solver showed good speedup and efficiency for the given number of mesh elements and processors.

Keywords CFD · Multi-phase flow · Ice accretion · Coarray · Parallel computation

1 Introduction

In Earth's atmosphere, two-phase flows of air and water droplets of various sizes may exist in a cloud containing supercooled water droplets [1–4]. Such flows can be found in the air-mixed supercooled droplet fields around aircraft flying inside a cloud, as well as around wind turbine blades, communication towers, and network cables operating in cold weather. Exposure to this condition for a considerable period may cause significant ice accretion on the surfaces of critical components such as wings, blades, cables, sensors, and engine inlets. This is called atmospheric icing in the field and it remains a critical issue in aircraft safety and other structural performance [5–8]. The study of atmospheric icing effects on aircraft is of critical importance because of the safety risks it poses, and the significant degradation of aerodynamic performance [7, 9, 10].

As with all other airplane components and systems, civil aircraft certification authorities need to certify ice protection systems [11]. To obtain certification, an aircraft must pass a strict certification process and demonstrate its performance under various metrological conditions dictated by civil aircraft certification authorities. Different icing certification methods may be employed to demonstrate aircraft safety under different icing envelopes and known icing conditions.

Computational methods are playing an increasing role in the icing certification process. They can be used to determine ice accretion and the effect of an ice protection system on an airplane under various metrological conditions. Computational analysis is also used in the pre-design of ice protection systems, and for metrological conditions that cannot be reproduced in wind tunnels or during natural flight tests. The computational cost of such analyses is far lower than the cost of wind tunnels and other flight tests. In addition to being able to simulate metrological conditions which cannot be reproduced by other methods, computational simulation permits the study of a wide range of metrological conditions. There is no restriction on the size of the model. Hence, scaling is unnecessary, allowing the real model with full configurations to be simulated.

Numerical ice accretion modeling requires a deep knowledge of fundamental icing physics, efficient mathematical techniques, and sufficient computation power. Generally, as

✉ R. S. Myong
myong@gnu.ac.kr

[1] Department of Aerospace Engineering and Applied Mechanics, Indian Institute of Engineering Science and Technology, Shibpur, Howrah 711103, India

[2] School of Mechanical and Aerospace Engineering and ACTRC, Gyeongsang National University, Jinju, Gyeongnam 52828, Republic of Korea

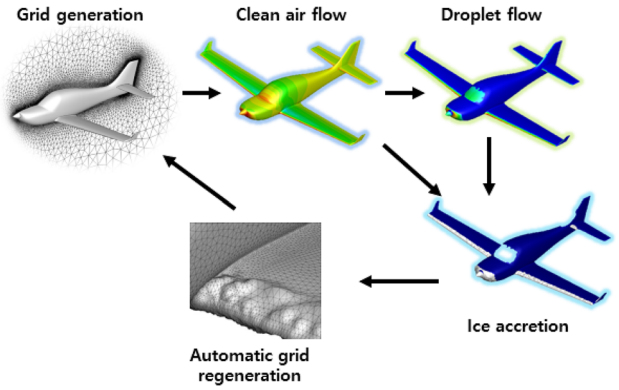

Fig. 1 The flow of ice accretion simulation solver

illustrated in Fig. 1, the ice accretion modeling starts by predicting the flow field around the model and continues with a droplet model, which predicts droplet impingement on the surface. Finally, a thermodynamic model is necessary to predict ice accretion and surface water flow. One of the challenging issues of icing simulations is the higher computational cost. This high cost is due to the large number of cells used to maintain the desired Y-plus value, needed to accurately evaluate shear stress and heat flux. In the growing computational world, parallel computing would be a better option to reduce computing time and avoid higher computational costs.

The air solver and droplet solver can be parallelized by using a Message Passing Interface (MPI) library [12, 13], Coarray Fortran library (Opencoarrays) [14], CUDA, OpenCL, etc. MPI and Opencoarrays guarantee maximum flexibility with parallel programming, as well as portability, and scalability of the distributed memory parallel architectures. Further, MPI and Coarray are easy to implement with unstructured computational fluid dynamics solvers, compared with any GPU-based parallel methods.

In the present work, a two-dimensional computational model based on partial differential equations for evaluating water film thickness, ice height, and equilibrium surface temperature was developed to investigate glaze ice accretion on complex configurations. As the first step, a compressible Navier–Stokes–Fourier air flow solver is used to determine the flow field around the model. A shallow water droplet model based on the Eulerian framework is then employed to estimate droplet impingement and the collection efficiency around the model. The airflow solver is parallelized by MPI and Coarray Fortran (CAF). It was found that the efficiency of Coarray Fortran was comparable to MPI. Then the droplet solver is parallelized using Coarray Fortran, because of its easy implementation and efficiency. The parallel performance, such as speedup and efficiency, was calculated and discussed in detail.

2 Computational Modeling of Ice Accretion

Numerical ice accretion modeling requires an in-depth knowledge of icing physics, efficient mathematical techniques, and sufficient computation power. Generally, ice accretion modeling starts by predicting the flow field around the model, followed by a droplet model, which predicts the droplet impingement on the surface. Finally, a thermodynamic model is necessary to predict ice accretion and surface water flow [15].

2.1 Airflow Solver

The present computational methodology is based on a one-way coupling model, in which the airflow field data are used as inputs to the water droplet solver through the source terms [15]. For this purpose, a well-known compressible Navier–Stokes–Fourier equation is employed. A finite volume method (FVM) based airflow solver coupled with the Spalart–Allmaras turbulent model was used [4, 16]. Generally in aerodynamic simulations, surfaces are considered to be smooth. However, since surface roughness increases with ice accretion, it is essential to include the roughness effect in the airflow simulations. The increase in surface roughness also increases the chance that flow will turn into turbulence. Surface roughness increases surface heat transfer by increasing the effective area on the surface due to growing skin friction. A correlation developed by NASA, which relates the metrological condition to equivalent sand-grain roughness, was used in the current simulations.

For the boundary conditions, non-slip and Riemann invariant conditions are applied to the solid surface and the far fields, respectively. An ideal gas equation is used to close the system of equations,

$$\begin{bmatrix} \rho_a \\ \rho_a \mathbf{u}_a \\ E \end{bmatrix}_t + \nabla \cdot \begin{bmatrix} \rho_a \mathbf{u}_a \\ \rho_a \mathbf{u}_a \mathbf{u}_a + p\mathbf{I} \\ (E+p)\mathbf{u}_a \end{bmatrix} = \nabla \cdot \begin{bmatrix} 0 \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \cdot \mathbf{u}_a + \mathbf{Q} \end{bmatrix}, \tag{1}$$

where,

$$\boldsymbol{\tau} = 2\mu[\nabla \mathbf{u}_a]^{(2)}, \ \mathbf{Q} = k\nabla T. \tag{2}$$

Here ρ_a, $\mathbf{u}_a$, p, and E represent the density, the velocity vector, the pressure, and the total energy of the air, respectively. The non-conserved variables $\boldsymbol{\tau}$ and $\mathbf{Q}$ denote the viscous shear stress tensor and the heat flux vector, respectively. Further, μ and k are the viscosity and thermal conductivity, respectively, and depend on the air temperature. The symbol $[\mathbf{A}]^{(2)}$ in the shear stress tensor stands for the traceless symmetric part of tensor $\boldsymbol{A}$. The ideal equation of state $p = \rho RT$ is used for the airflow.

2.2 Water Droplet Solver

A shallow water droplet model based on the Eulerian framework developed in previous work is employed for the water droplet solver [4, 16]. The conservative form of SWDE can be written as,

$$\begin{bmatrix} \rho \\ \rho \mathbf{u} \end{bmatrix}_t + \nabla \cdot \begin{bmatrix} \rho \mathbf{u} \\ \rho \mathbf{u}\mathbf{u} + \rho \boldsymbol{g} d \mathbf{I} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{S}_D + \mathbf{S}_G + \mathbf{S}_S \end{bmatrix}. \tag{3}$$

Here ρ, $\mathbf{u}$, and d denote the density of droplets in terms of liquid water content (LWC), the velocity components of the droplet, and the reference size of droplets (MVD), respectively. Further, $\mathbf{S}_G = \rho g[0, 0, 1 - \rho_g/\rho_w]^{\mathrm{T}}$ is the resultant force of gravity and the buoyancy of droplets, and the term $\mathbf{S}_{\mathrm{D}} = A_{\boldsymbol{u}}(u_g - \boldsymbol{u})$ denotes the drag on droplets caused by the airflow. And $\mathbf{S}_{\mathrm{S}} = \nabla \cdot (\rho g d)\mathbf{I}$ is the added source term to circumvent the non-strictly hyperbolic nature of the droplet equations. The coefficient $A_{\boldsymbol{u}}$ can be expressed as,

$$\begin{aligned} A_u &= 0.75 \cdot \rho \cdot C_D \cdot Re \cdot \mu \big/ \rho_w \cdot d^2, \\ Re &= \rho_g \cdot d \cdot \left| \boldsymbol{u}_g - \boldsymbol{u} \right| \big/ \mu. \end{aligned} \tag{4}$$

The drag coefficient (C_D) can be obtained from Lapple [17] as follows,

$$C_D = \frac{24}{Re}\left(1 + 0.0197 Re^{0.63} + 2.6\mathrm{e}^{-4} \cdot Re^{1.38}\right), \tag{5}$$

which is valid for Re < 1000. The second-order positivity-preserving finite volume framework based on cell-centered unstructured grids is employed to solve the Eulerian-based shallow water droplet equations. The Harten–Lax–van Leer–Contact (HLLC) approximate Riemann solver is used for the flux calculation. A five-stage Runge–Kutta temporal discretization is implemented.

2.3 Ice Accretion Solver

The ice accretion solver was developed using the Messinger model [18], based on a partial differential equation (PDE) formulation, instead of the commonly used control volume method. When implementing the high-fidelity PDE-based thermodynamic model similar to the ICE3D module of FENSAP-ICE [19], a recalibrated equation was used to avoid having different units on both sides of the original energy equations. The revised equation can be written as [15],

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{U}) = \boldsymbol{S}. \tag{6}$$

Equation (6) can be written as

$$\begin{aligned} \boldsymbol{U} &= \begin{bmatrix} h_f \\ h_f T_{\mathrm{equi}} \end{bmatrix}, \boldsymbol{F}(\boldsymbol{U}) = \begin{bmatrix} \frac{h_f^2}{2\mu_w}\tau_{\mathrm{wall}} \\ \frac{h_f^2 T_{\mathrm{equi}}}{2\mu_w}\tau_{\mathrm{wall}} \end{bmatrix}, \\ \boldsymbol{S} &= \begin{bmatrix} \frac{S_M}{\rho_w} \\ \frac{S_E}{\rho_w C p_{,w}} + \frac{T_c S_M}{\rho_w} \end{bmatrix}, \\ S_M &= U_\infty L W C_\infty \beta - \dot{m}_{\mathrm{evap}} - \dot{m}_{\mathrm{ice}}, \\ S_E &= \left[C p_{,w} \tilde{T}_{d,\infty} + \frac{\|\vec{u}_d\|^2}{2} \right] \times U_\infty L W C_\infty \beta - L_{\mathrm{evap}} \dot{m}_{\mathrm{evap}} \\ &+ \dot{m}_{\mathrm{ice}}\left[L_{\mathrm{fus}} - C p_{,\mathrm{ice}} T_{\mathrm{equi}} \right] + h_c\left(T_{\mathrm{equi}} - T_\infty\right) + \sigma_o \varepsilon \left[T_{\mathrm{equi}}^4 - T_\infty^4 \right]. \end{aligned} \tag{7, 8}$$

Here, μ_w, Cp_w, Cp_{ice} represent the dynamic viscosity, the specific heat at constant pressure for water, and the specific heat at constant pressure for ice, respectively. The instantaneous evaporation mass, latent heat of fusion, and latent heat of evaporation are represented by $\dot{m}_{\mathrm{evap}}$, L_{fus} and L_{evap}, respectively. Further, T_c, $\tilde{T}_{d,\infty}$ and $\boldsymbol{u}_d$ represent the critical temperature ($T_c = 273.15$ K), droplet temperature in Celsius, and droplet impact velocity vector, respectively. The terms ε and σ represent the solid emissivity and Boltzmann constant ($\sigma_{\mathrm{a}} = 1.38064852 \times 10^{-23}$ m^2 kg s^{-2} K^{-1}), respectively. By assuming a linear velocity distribution within the film, the velocity of the water film $\left(\overline{\mathbf{u}}_f\right)$ can be represented as a function of the water film thickness and shear stress, as

$$\overline{\mathbf{u}}_f = f\left(h_f\right) = \frac{1}{h_f}\int_0^{h_f} u_f dh = \frac{h_f}{2\mu_w}\tau_{\mathrm{wall}}. \tag{9}$$

The clean airflow solver provides shear stress (τ_{wall}) and heat transfer coefficient (h_c) as inputs to the ice accretion solver. The shallow water type droplet solver provides the droplet impact velocity and collection efficiency as inputs to the ice accretion solver. There are three unknowns to be computed: water film thickness (h_f), equilibrium temperature (T_{equi}), and mass accumulation ($\dot{m}_{\mathrm{ice}}$). Since only two governing equations are available, compatibility relations are necessary to close the system. Based on the physical behavior, the following compatibility equations can be derived,

$$h_f \geq 0, \;\; \dot{m}_{\mathrm{ice}} \geq 0, \;\; h_f T_{\mathrm{equi}} \geq h_f T_C, \;\; \dot{m}_{\mathrm{ice}} T_{\mathrm{equi}} \leq \dot{m}_{\mathrm{ice}} T_C. \tag{10}$$

The first compatibility relation ensures that the film thickness remains positive. The second compatibility relation prevents the melting of accreted ice. The third compatibility relation enforces that the water film only can exist at an equilibrium temperature above freezing point. Finally, the fourth compatibility equation stipulates that ice cannot form for equilibrium temperatures above freezing. Each cell of the

domain is explicitly solved individually with small time steps using the compatibility equations.

The governing equations and the compatibility relations for the water film on the body's surface are solved using a cell-centered finite volume method. The computation elements are spread over the body's surface and must be solved using numerical techniques. Roe's approximate Riemann solver is used to discretize the divergence terms in the governing equations. Overall, all of the essential solvers for clean air, large droplet impingement, ice accretion, and the aerodynamic analysis of ice effects were developed within a unified computational framework based on an unstructured finite volume method in which all the information can be easily communicated in a single grid system during the simulation [15]. More details about the numerical methods used in this current work can be found in the past literature [4, 6, 16, 20].

3 Parallelization of the FVM-Based Ice Accretion Package

The local nature of discretization renders the cell-centered finite volume method compact and highly parallelizable. Since the solution is independently approximated for each element of the inter-element data, to calculate numerical fluxes sharing is only needed among the facing neighbor elements, or elements sharing a common face. Inter-process communication is only required between the corresponding neighboring processes for computations at partition boundary faces, or for faces whose left and right elements have different processes.

The present FVM code was parallelized using a single program multiple data (SPMD) parallel model [4]. The air and droplet solver was parallelized using a Message Passing Interface (MPI) library OpenMPI 4.0 and Coarray Fortran library OpenCoarrays 2.0, respectively. MPI and Opencoarrays guarantee maximum flexibility for the parallel programming and the portability and scalability of the distributed parallel memory architectures. As seen in Fig. 2, all the parallel processing steps were unified using a shell program [4].

A software setup including a high-performance and widely portable implementation of the parallel library and 64-bit compilers with double precision was used for all the floating point operations. Furthermore, using Intel Xeon Gold 6136 Processor with a base frequency of 3.0 GHz, a Linux cluster was established that is shareable among multiple users, equipped with eighty cores interconnected by dual-port Gigabit Ethernet. Parallelization of the FVM solver for airflow included the following steps: (1) domain decomposition (mesh partitioning); (2) communication process; (3) merging of sub-domains, and (4) parallel performance measurements. These steps are further described in the following sub-sections.

3.1 Domain Decomposition

As the first step in parallel programming, the computational domain is decomposed into several sub-domains. Then each sub-domain is assigned to each processor using open-source software, ParMETIS [21], an MPI-based parallel library that implements various algorithms to compute fill-reducing orderings of sparse matrices. To partition the unstructured graphs, the given mesh is decomposed so that each processor has approximately the same number of elements to balance the load for the processors, minimizing the number of links cut by the decomposition. The partitioned results and the included node and element connectivity information after the decomposition of the domain are assigned to the processors. The sub-domains generated by ParMETIS for flows around an airfoil and a three-element airfoil with approximately 50,000 elements are illustrated in Fig. 3.

3.2 Communication process

The present parallel solver for both air and droplet is based on a single program multiple data (SPMD) programming model [4]. It executes the same program in all processors using different data and can manage the processors to conditionally execute only certain parts of the program so that some of the processors may not necessarily need to execute the entire program.

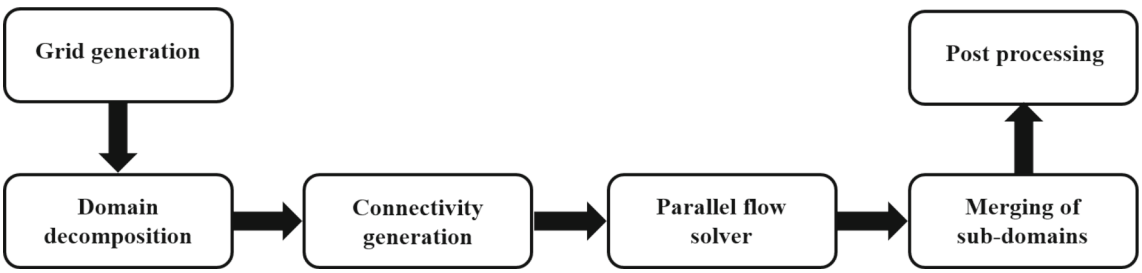

Fig. 2 Schematic diagram of the unified shell program for parallel processes

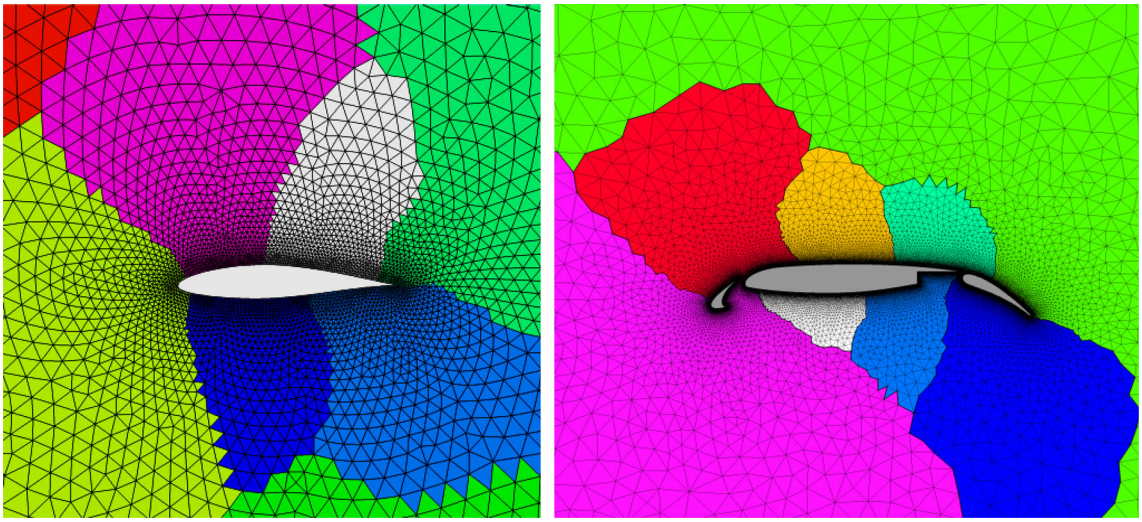

Fig. 3 The mesh partition by ParMETIS. Different colors represent sub-domains owned by different processors

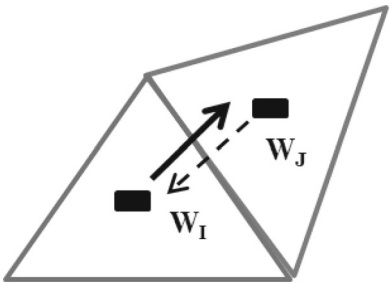

Fig. 4 Point-to-point communication between the cells from different processor groups

The MPI library and Opencoarray library were used to parallelize the air solver to compare the performance. Then Coarray Fortran was utilized to parallelize the Eulerian-based droplet solver. Both libraries have point-to-point and collective communication modules that can be utilized for the present work. A parallel solver's communication process is the parallelization engine, which should be smooth and fast for better performance. A delay in communication may lead to performance loss in the parallel solver. Both parallel libraries have various communication modules that can be applied to parallelize present air and droplet solvers. Hence, selecting a communication module is critical and tricky to avoid unnecessary performance penalties. In general, communication is conducted between two adjacent cells belonging to different processors and sharing the same face. Figure 4 illustrates the point-to-point data communication on the face of the cells.

3.3 Merging of Sub-domains

All the partitioned sub-domains execute the same solver with respective data inputs and solve the flow fields in their local domain during the parallel computations. An individual processor plots its solution after the solution converges for post-processing purposes. However, the results were not visually smooth at the boundaries of the sub-domains, due to the biased interpolation of the solution, and because not all of the vertex neighborhoods for interpolations were considered. Hence, a merging subroutine was devised for better post-processing of the solutions of the parallel computations, where all subdomain results were exported into a single unified domain for better visualization. All the solutions are needed to be interpolated to the node for post-processing software such as TECPLOT although each element contains its own solution so that the biased interpolation that considers all the neighbors of the node could not result in very poor visualization.

Merging of sub-domains was performed to avoid such limitations in post-processing, after terminating parallel processing. The results were irrespective of the number of processors, since the solution in each element is calculated locally and located at the cell center. The unmerged (left) and merged (right) results are illustrated in Fig. 5.

3.4 Measurement of Parallel Performance

Measuring parallel computation is essential, to assess the efficiency and applicability of the parallel solver. Generally, parallel performance is measured by relative speedup, relative efficiency, or scalability. Amdahl's law established the definition of speedup (Sp) [22]. According to this law, Sp is a metric for measuring the relative improvement in performance when executing a task. However, the speedup can also be used more generally to show the effect of any performance

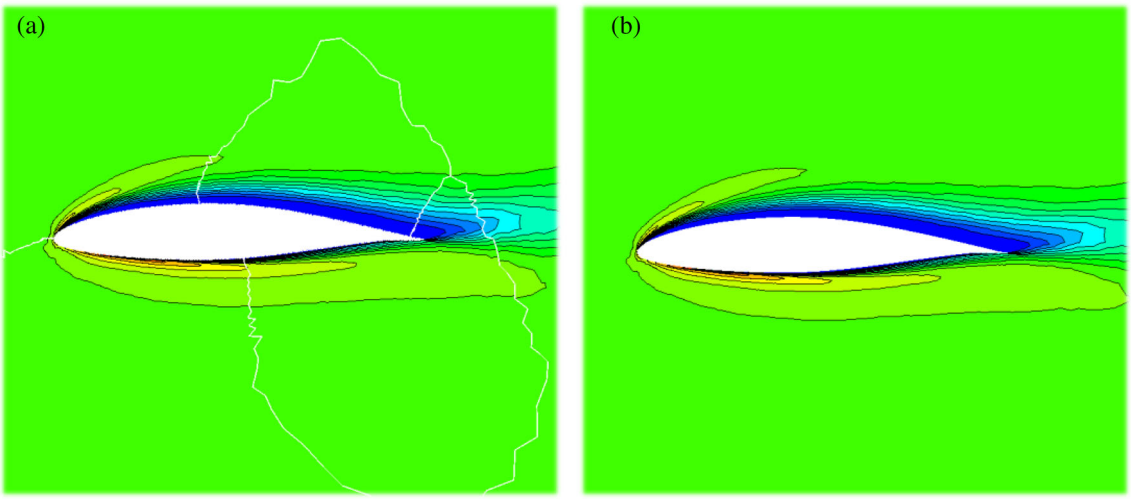

Fig. 5 Merging of the sub-domains for post-processing of the solutions; **a** LWC contour of unmerged sub-domains, and **b** LWC contour of the merged domain

enhancement. The relative speedup is given by

$$S_p = \frac{t_s}{t_p}, \tag{11}$$

where S_p is speedup, t_s and t_p denote the elapsed time taken by a *single* processor and p processors, respectively. Relative efficiency (E) is a metric of the utilization of the resources of the improved parallelized system, read as

$$E = \frac{S_p}{p}. \tag{12}$$

A performance analysis determines the level of speedup and the efficiency of the parallel solver. Speedup of the code varies with the number of processors for a fixed problem size. Linear speedup usually remains less than p, and efficiency lies between 0 and 1. In ideal cases, elapsed time taken by p processors is equal to $t_p = t_1/p$, the relative speedup is equal to$S_p = p$, and the relative efficiency is equal to $E = 1$.

4 MPI and Coarray Fortran FVM Implementation

4.1 MPI Implementation for Air Solver

The present air flow field solver was parallelized using the MPI library and Coarray libraries. The parallelization in the present study was achieved without compromising the serial algorithm for higher parallel performance and allowed MPI communications to completely overlap with the computations [4]. As summarized in Fig. 6, this algorithm is easier to achieve in explicit time marching schemes, which is usually referred to as hiding communication behind computation. The point-to-point communication methodology of MPI, in which a message-passing operation may only occur between two different processors, was used so that while one processor performs a send operation, the other performs a matching receive operation.

Several types of *send* and *receive* routines are available in MPI point-to-point communication, including blocking and non-blocking routines, which are often used in the SPMD model due to their flexibility and implementation. Both methods use a buffer to avoid data loss and confusion during data transmission, such that the data will be copied to the buffer before the partner processor receives it. During the communication process, the data can be temporarily stored in a buffer, which is a region of memory storage. In the blocking send and receive routines, the send routine will only return (block) after the completion of communication so that the computations cannot be performed by the respective processors involved in the communication until the process is completed. In contrast, the non-blocking communication functions return immediately (do not block) even if the communication is not finished. Hence, care should be taken to use the proper wait comment to see whether the communication has finished. The latter communications are primarily used to overlap computation with communication and exploit possible performance gains.

Initially, the communication module of the FVM solver works by sending data adjacent to partition boundaries to neighbor partitions, and then by receiving data from a corresponding neighbor. To save processor waiting time and avoid deadlock, non-blocking sending and receiving were used in the parallelization of the solver. The MPI_TEST and MPI_WAIT routine is used to confirm the completion

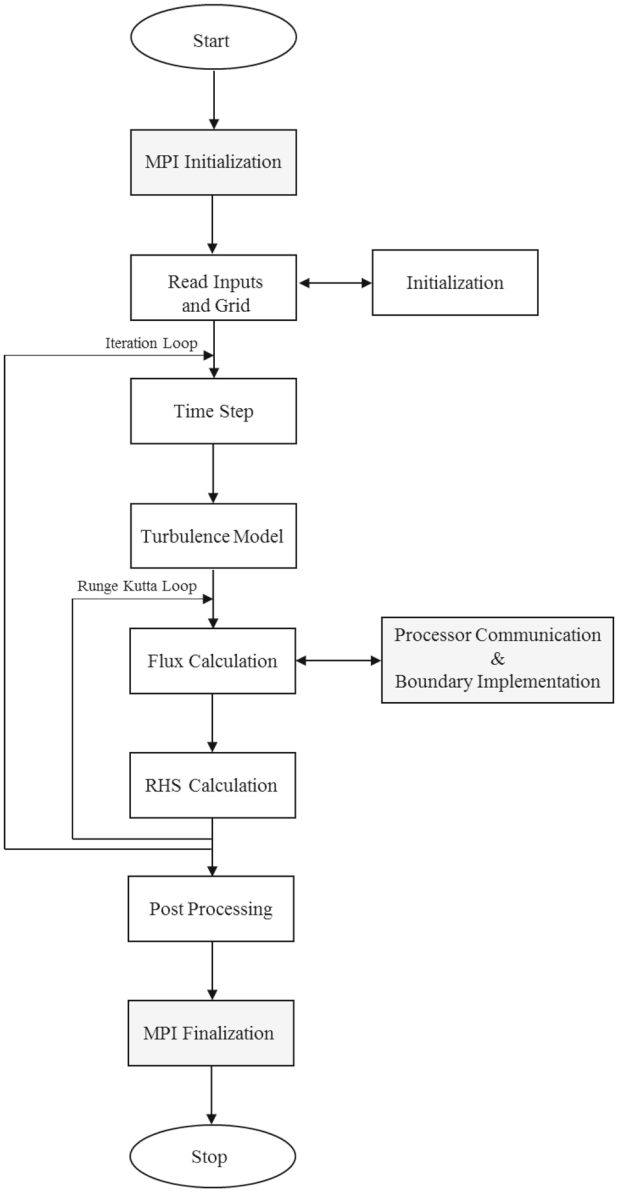

Fig. 6 Flow chart of explicit parallel air solver

of communication without data loss as started by calling standard MPI routines, MPI_ISEND and MPI_IRECV. It is worth mentioning that the MPI_TEST routine is used to check the completion of the communication, and a call to MPI_WAIT returns when the operation identified by request (MPI_TEST) is complete. Figure 7 illustrates the block diagram of the communication algorithm. Once communication is completed, the data received from the neighboring processors are used for further computations.

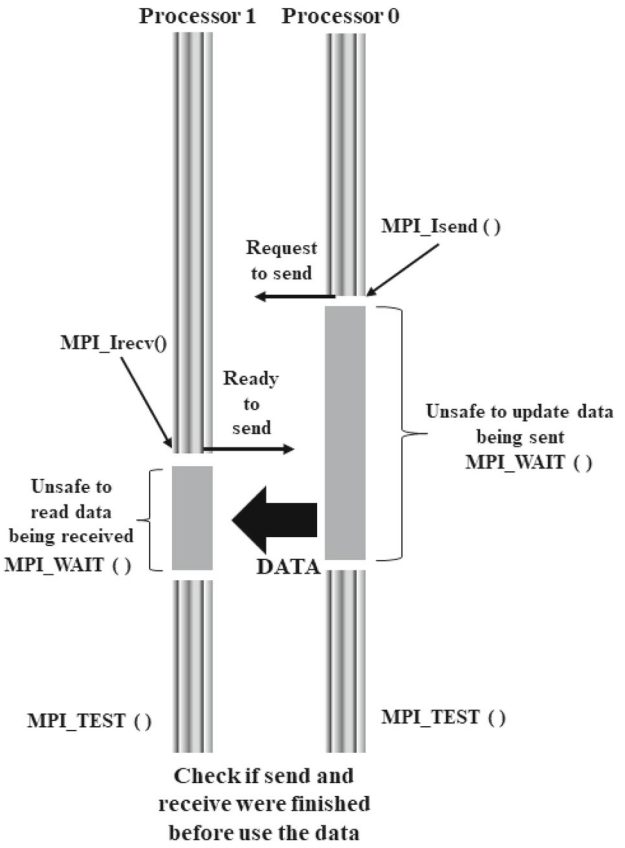

Fig. 7 Isend and Irecv block diagram based on non-blocking communication

4.2 Coarray Implementation for Air and Droplet Solver

Generally, CAF is based on the SPMD programming model, and the user can select the number of parallel sections (program replications) through command line instructions [23, 24]. In CAF, each program replication is called an image, has its own variable set, is executable, and runs asynchronously. The intrinsic function **`num_images()`** is used to determine the available number of images. Each image has its image index, numbered from 1 to **`num_images()`**; the index of an image can be readily available by using an intrinsic function **`image_index().`**

Unlike MPI, the objects in the CAF program to be shared should be declared as a coarray variable, and then they can be accessed from their own image and other images. For example,

```
integer:: x(5)          // Normal declaration of variable 'x'
integer:: y(5)[*]       // coarray declaration of variable 'y'
```

Here, the variables **x** and **y** are available in all images, but only **y** is accessible by all images, and the variable **x** is a local variable that only its own image can access. We can access the y variable of other images as follows,

$$\mathbf{x}(\mathbf{5}) = \mathbf{y}(\mathbf{5})[\mathbf{4}].$$

The number 4 indicates the image number; hence the values in **y** are copied from image number 4 to the variable **x** of the current image. In the parallelization of any program, a few commands, such as synchronization, point-to-point communication, broadcasting, and collective operations, are highly essential.

In Coarray Fortran, all the routines required to implement CFD codes are readily available. For instance, the synchronization is performed using a simple command, **sync_all,** and generally, synchronization is used before and after collective operations. Further, Broadcasting and collective operations such as **co_min, co_max, co_sum**, and **co_reduce** are available and can be used in the parallel CFD solver development.

In short, implementing the coarray is easy, and the number of lines of the program is also effectively reduced without any drastic performance loss. Moreover, the coarray implementation does not need special syntax and call functions. Hence, the program is readable and easy to understand, maintain or extend. In the present work, the Opencoarrays library, an open-source software project which produces an interface used by the GNU Compiler Collection (GCC) Fortran front end to build parallel executable programs, was used to implement CAF. The Opencoarrays library is easy to install and is used in Linux and macOS. It is reliable since it has been tested in several of the world's fastest supercomputers and various operating systems [23–25].

The water droplet solver also follows the basic steps and processes mentioned in Fig. 6 but uses a droplet solver and Opencoarrays library instead of a flow solver and MPI library, respectively. The compiler identifies coarrays during the compile time, from the declaration of variables. Hence, initialization commands are not necessary to initiate coarray execution. There are two options available to copy data from one image to another, "push" and 'pull." In the "push' mode, local data is stored in the remote image, and in the "pull" mode, data is copied from the remote to the local image. For example,

```
LWC [remote] = drop_property(1)          / Push mode
drop_property(1) = LWC [remote]          / Pull mode
```

Previous research [26] has noted that the "`pull`" mode is more efficient than the "`push`" mode of data transfer. Therefore, the "`pull`" data transfer mode is used in the present code. Furthermore, it is possible to combine both the MPI and coarray libraries in a single program, which may improve the parallel performance.

5 Results and Discussion

5.1 Validation of Ice Accretion Simulation Package

To validate the aerodynamic solver, an unstructured mesh was used, as shown in Fig. 8 (left), and the computed pressure coefficient over a NACA65$_2$-415 airfoil was compared with the experimental result shown in Fig. 8 (right). NASA experimented on NACA65$_2$-415 with a 0.9144 m chord airfoil model at a Mach number of 0.23, an angle of attack of

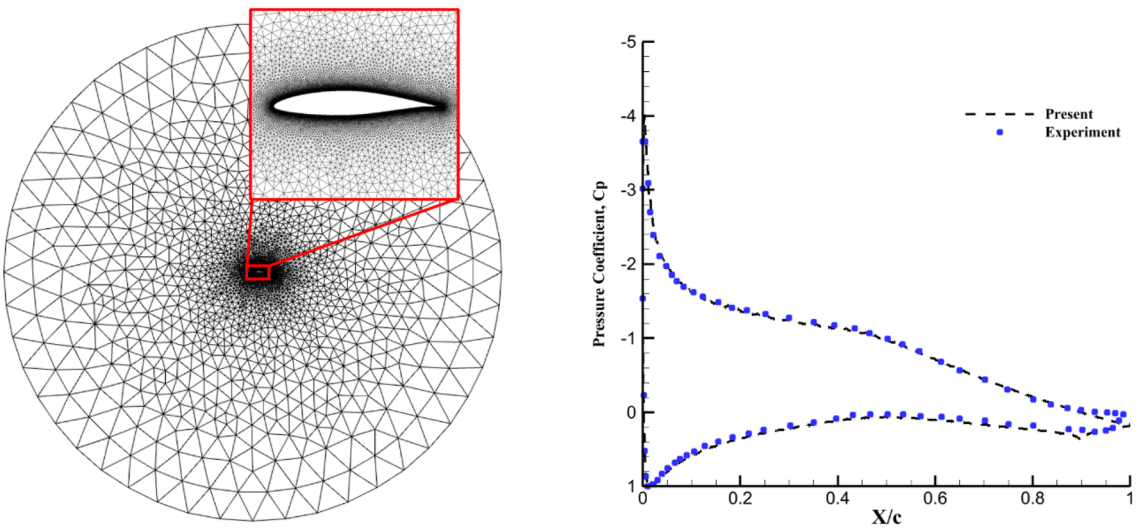

Fig. 8 Grid distribution (left) and Pressure coefficient (right) around the NACA652-415 airfoil at $\alpha = 8o$, $M = 0.23$, Re = 4.9 million

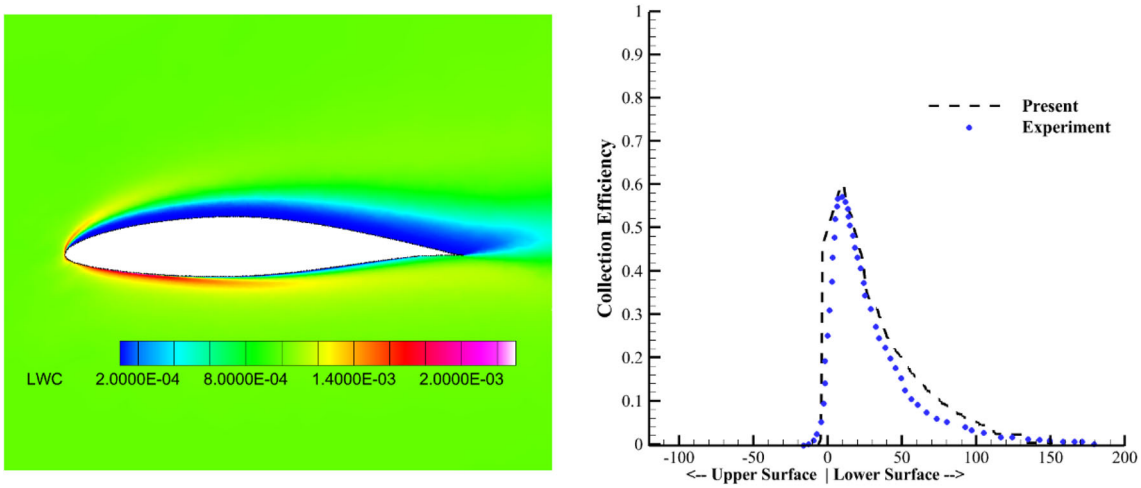

Fig. 9 LWC contour (left) and collection efficiency (right) around the NACA65$_2$-415 airfoil at $\alpha = 8$, $M = 0.23$, LWC $= 1.0$ g/m^3, MVD $= 21$ μm

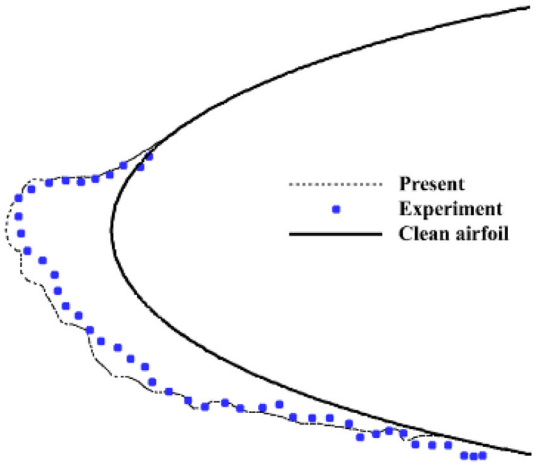

Fig. 10 Ice accretion result comparison with NASA IRT experiments of glaze ice shape on a NACA0012 airfoil at temperature 262.04 K, velocity 102.8 m/s, angle of attack 4°, LWC 0.55 g/m^3, MVD 20 μm, and exposure time 420 s

8°, and a Reynolds number of 5.2 million. The computed droplet collection efficiency was compared with the experimental results to validate the droplet solver. The comparison between the computed result and NASA data is depicted in Fig. 9, where the LWC is 1.0 g/m^3, and the MVD is 21 μm.

Finally, the ice accretion shapes on NACA0012 for glaze ice cases were validated against experiments [24], as elucidated in Fig. 10. The icing conditions selected for the simulation were temperature 262.04 K, velocity 102.8 m/s, angle of attack 4°, LWC 0.55 g/m^3, MVD 20 μm, and exposure time 420 s.

Further, a complex multielement airfoil simulation was conducted to validate the current solvers. Figure 11 (left) shows the LWC distribution, while Fig. 11 (right) shows the collection efficiency around the multielement airfoil at the angle of attack 4°, Mach number 0.24, temperature 276 K, LWC 0.15 g/m^3, and MVD 21 μm. The impingements of droplets are higher on the leading edge of the slat and the pressure side of the flap, while the main element does not acquire significant impingements due to the presence of the leading edge slat. Finally, the ice accretion around the multi-element at an angle of attack 8°, Mach number 0.26, temperature 268.2 K, LWC 0.6 g/m^3, and MVD 20 μm is shown in Fig. 12. Overall, the computed results qualitatively agreed with the experimental results. Consequently, the current computation model can be applied for further investigations.

5.2 Parallel Performance of MPI and Coarray Fortran for Air and Droplet Solvers

Figure 13 (left) illustrates the speedup [calculated using Eq. (11)] for the MPI and Coarray parallel implementations of the air solver for elements ranging from 37,000 to 300,000, and with a range of 1–32 processors. It is worth mentioning that the scalability test considered in this work was performed on a single-node machine. The simulations were repeated five times to statistically measure the computational cost. It was observed that the speedup gradually increased at the lower number of processors and began declining as the number of processors grew higher. Further, it was noted that the speedup of the MPI parallel implementation was almost identical to the Coarray implementation.

Figure 13 (right) shows the relative efficiency [calculated using Eq. (12)] of the parallel code for MPI and Coarray parallel implementations on the air solver. The communication overload increases as the number of processors increases, and, as a result, the required run-time for communication between processors becomes comparable to the computational time of the simulations for cases with a smaller number of elements. Overall, the speedup and efficiency show that

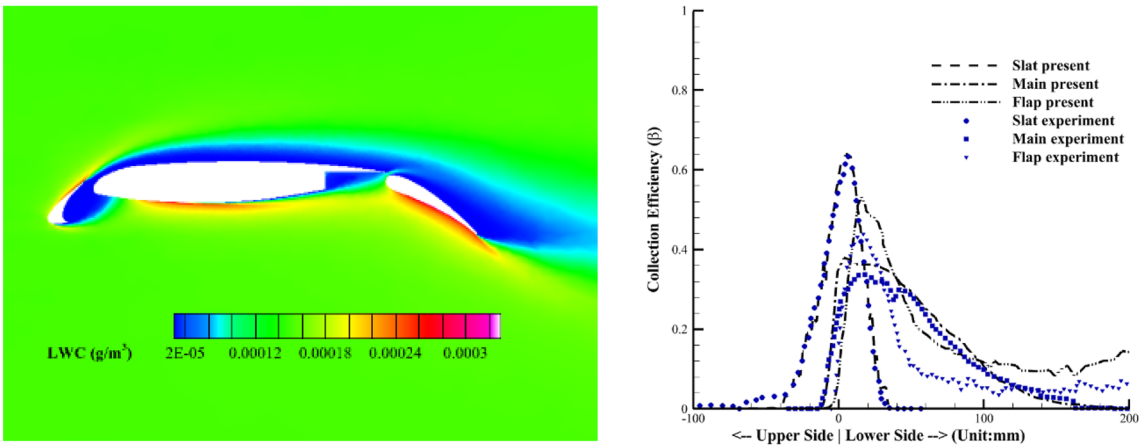

Fig. 11 LWC contour (left) and comparison of collection efficiency (right) around the multi-element airfoil at the angle of attack 4°, Mach number 0.24, temperature 276 K, LWC 0.15 g/m^3, and MVD 21 μm

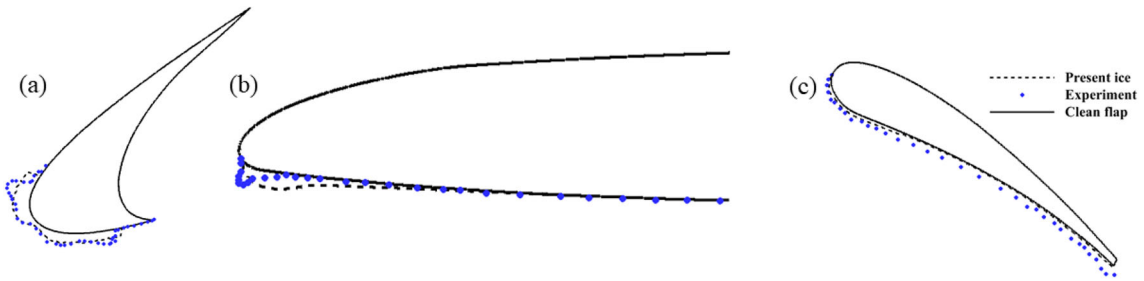

Fig. 12 Ice accretion result comparison with NASA IRT experiments of ice shape on the slat (**a**), main element (**b**), and flap (**c**) of the multi-element airfoil at temperature 268.2 K, Mach number 0.26, angle of attack 8°, LWC 0.6 g/m^3, and MVD 20 μm

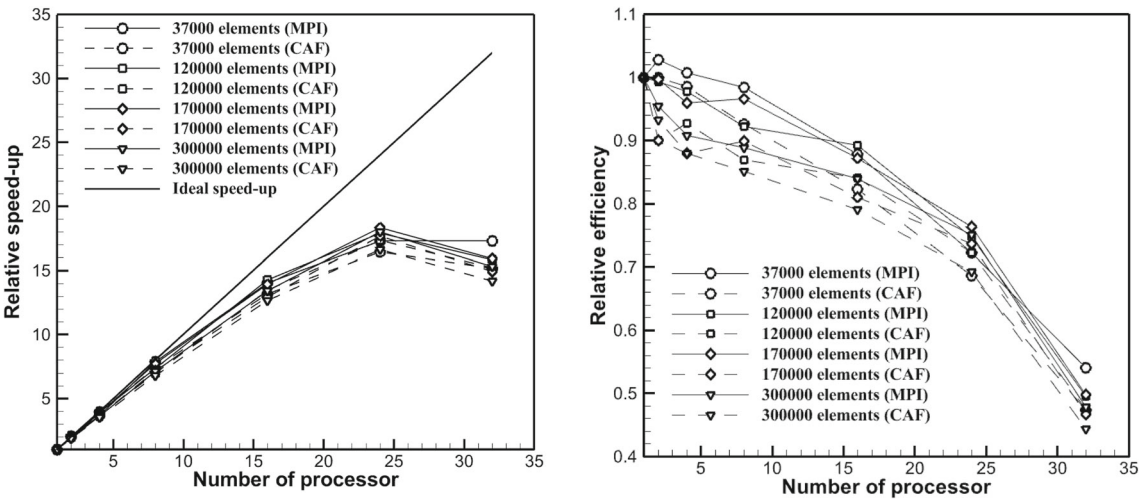

Fig. 13 Comparison of parallel speedup (left) and efficiency (right) of airflow solver for the MPI and Co-array parallel implementations

the MPI and Coarray performance is almost the same at the given number of cells and processors. Hence, only the Coarray implementation was carried out for the droplet solver, because of the easy implementation of Coarray.

Figure 14(left) illustrates the Coarray-based parallel water droplet solver's speedup at various element sizes. The results show that the speedup is increasing linearly and drops at a higher number of cells due to the increase in communication loads. Further, Fig. 14(right) shows the Coarray-based parallel droplet solver's parallel efficiency at various element sizes. It was found that the efficiency was low for the lower number of cells case, and it increased with a large number of

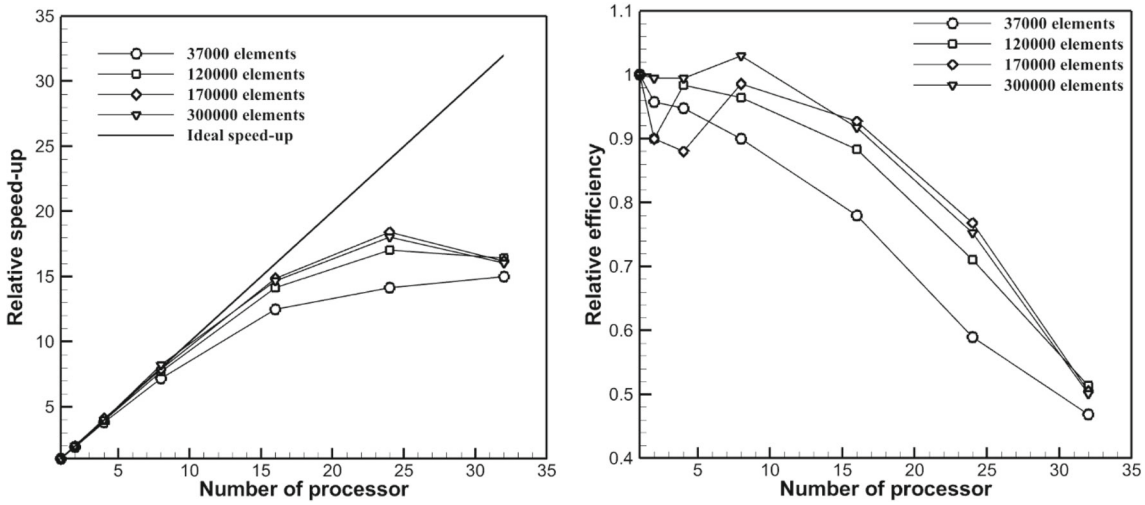

Fig. 14 Comparison of parallel speedup (left) and efficiency (right) of water droplet solver for the Co-array parallel implementations

cells under the given conditions. Overall, the results showed that the Coarray-implemented parallel code performed well, similar to the MPI implementation.

6 Conclusion

An SPMD-based parallel two-dimensional solver for the ice accretion simulation was developed using the MPI and Coarray libraries. For the study, a unified upwind finite volume solver based on a single unstructured grid system was used, which could handle all the modules of airflow, droplet impingement, and ice accretion. The present parallel solver adopts domain decomposition using ParMETIS, and the Coarray library to parallelize the droplet solver. Further, parallelization of the air solver was developed using both MPI and Coarray. The results show that the parallel performance of Coarray is similar to that of MPI for the air solver under a given number of processors. The current parallel implementation of the ice accretion package is intended for stand-alone PCs equipped with a large number of processing cores and for clusters of PCs.

Based on the successful implementation of the parallel ice accretion solvers in the present work, the next topic of research will be the application of these methods to the ice accretion simulation of complex geometry. Notably, the current methodology can be extended to the three-dimensional ice accretion package using Coarray, reducing computation time significantly. We hope to report the results of studies on these problems in due course.

Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Ministry of Science and ICT (NRF-2017-R1A5A1015311), South Korea. Some parts of this article have been presented in two preprints: the first author's doctoral thesis and conference paper 2022-0447 at the AIAA SciTech 2022 Forum, San Diego, USA.

Data availability Data are available from the corresponding author on request.

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Landsberg B (2008) Safety advisor: aircraft icing. AOPA Air Saf. Found
2. Jones SM, Reveley MS, Evans JK, Barrientos FA (2008) Subsonic aircraft safety icing study. NASA/TM-2008–215107
3. Bae J, Yee K (2021) Numerical investigation of droplet breakup effects on droplet-wall interactions under SLD conditions. Int J Aeronaut Sp Sci 22:1005–1018. https://doi.org/10.1007/s42405-021-00374-y
4. Prince Raj L (2017) High-fidelity computational modeling of in-flight ice accretion on aircraft and rotorcraft including super-cooled large droplet. Gyeongsang National University. http://acml.gnu.ac.kr/download/Publications/RAJ_PhD_Thesis-2017.pdf
5. Ahn GB, Jung KY, Myong RS, Shin HB, Habashi WG (2015) Numerical and experimental investigation of Ice accretion on rotorcraft engine air intake. J Aircr 52:903–909. https://doi.org/10.2514/1.C032839
6. Prince Raj L, Yee K, Myong RS (2020) Sensitivity of ice accretion and aerodynamic performance degradation to critical physical and modeling parameters affecting airfoil icing. Aerosp Sci Technol. https://doi.org/10.1016/j.ast.2019.105659
7. Sengupta B, Raj LP, Cho MY, Son C, Yoon T, Yee K, Myong RS (2021) Computational simulation of ice accretion and shedding trajectory of a rotorcraft in forward flight with strong rotor wakes. Aerosp Sci Technol 119:107140
8. Roy R, Raj LP, Jo JH, Cho MY, Kweon JH, Myong RS (2021) Multiphysics anti-icing simulation of a CFRP composite wing structure embedded with thin etched-foil electrothermal heating films in

glaze ice conditions. Compos Struct 276:114441. https://doi.org/10.1016/j.compstruct.2021.114441
9. Jung S, Raj LP, Rahimi A, Jeong H, Myong RS (2020) Performance evaluation of electrothermal anti-icing systems for a rotorcraft engine air intake using a meta model. Aerosp Sci Technol 106:106174. https://doi.org/10.1016/j.ast.2020.106174
10. Yu Z, Li Y, Zhang Z, Xu W, Dong Z (2020) Online safe flight envelope protection for icing aircraft based on reachability analysis. Int J Aeronaut Sp Sci 21:1174–1184. https://doi.org/10.1007/s42405-020-00266-7
11. Ma F, Comeau D (1990) Aircraft de-icing and anti-icing composition. https://www.google.com/patents/US4954279
12. Pacheco PS (1997) Parallel programming with MPI. Morgan Kaufmann Publishers Inc., San Fr. Calif. (n.d.)
13. Qiao K, Xu X (2022) Parallel multiscale numerical framework of the non-linear failure analysis for three-dimension composite structures. Int J Aeronaut Sp Sci 23:77–91. https://doi.org/10.1007/s42405-021-00430-7
14. Fanfarillo A, Burnus T, Cardellini V, Filippone S, Nagle D, Rouson D (2014) OpenCoarrays: open-source transport layers supporting coarray Fortran compilers. In: Proc. 8th Int. Conf. Partitioned Glob. Address Sp. Program. Model, pp 1–11
15. Prince Raj L, Esmaeilifar E, Jeong H, Myong RS (2022) Computational simulation of glaze ice accretion on a rotorcraft engine intake in large supercooled droplet icing conditions. In: AIAA SCITECH 2022 Forum, American Institute of Aeronautics and Astronautics, Reston, Virginia. https://doi.org/10.2514/6.2022-0447
16. Jung SK, Myong RS (2013) A second-order positivity-preserving finite volume upwind scheme for air-mixed droplet flow in atmospheric icing. Comput Fluids 86:459–469. https://doi.org/10.1016/j.compfluid.2013.08.001
17. Lapple CF (2007) Fluid and particle mechanics. Vincent Press
18. Myers TG (2001) Extension to the Messinger model for aircraft icing. AIAA J 39:211–218
19. Beaugendre H, Morency F, Habashi WG (2003) FENSAP-ICE's three-dimensional in-flight ice accretion module: ICE3D. J Aircr 40:239–247. https://doi.org/10.2514/2.3113
20. Prince Raj L, Lee JW, Myong RS (2019) Ice accretion and aerodynamic effects on a multielement airfoil under SLD icing conditions. Aerosp Sci Technol 85:320–333. https://doi.org/10.1016/j.ast.2018.12.017
21. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20:359–392
22. Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. In: Proc. April 18–20, 1967, Spring Jt. Comput. Conf., pp 483–485
23. Numrich RW, Reid J (1998) Co-Array Fortran for parallel programming. In: ACM Sigplan Fortran Forum. ACM New York, NY, USA, pp 1–31
24. Mellor-Crummey J, Adhianto L, Scherer III WN, Jin G (2009) A new vision for Coarray Fortran. In: Proc. Third Conf. Partitioned Glob. Address Sp. Programing Model, pp 1–9
25. Jin G, Mellor-Crummey J, Adhianto L, Scherer III WN, Yang C (2011) Implementation and performance evaluation of the hpc challenge benchmarks in coarray Fortran 2.0. In: 2011 IEEE Int. Parallel Distrib. Process. Symp., IEEE, pp 1089–1100
26. Ashby JV, Reid JK (2008) Migrating a scientific application from MPI to Coarrays. In: CUG 2008 Proceedings, pp 1–8