

A Study of Parallel Scalability and Dynamic Workload Balancing in GlennICE

Zaid H. Sabri* and Christopher E. Porter†
NASA Glenn Research Center, Cleveland Ohio, 44135 USA

The Glenn Icing Computational Environment (GlennICE) is a computational tool designed to calculate ice growth on complex three-dimensional geometries. It utilizes user-supplied computational fluid dynamics solutions for the geometry of interest. Key developments include advancements in convergence of collection efficiency, trajectory optimization, and refinement methodology. These improvements have significantly enhanced GlennICE’s efficiency for practical engineering applications.

A recent study focused on benchmarking GlennICE’s scalability in a parallel environment using static scheduling. Findings indicated a potential twofold increase in efficiency through workload balance enhancements. This paper presents an analysis of the solver’s new workload balancing improvements, incorporating shared memory and dynamic scheduling routines. Results demonstrate a highly efficient and consistent algorithm across high-performance computing clusters.

I. Introduction

NASA’s Aeronautics Research Mission Directorate (ARMD) is investing in icing technology development to enable future airplane designs with novel vehicle features, e.g., the Transonic Truss Braced Wing (TTBW) project and better inclement weather performance efficiency. NASA’s Advanced Air Transport Technology (AATT) and Transformational Tools and Technologies (TTT) projects are also interested in tackling more general icing challenges to enable the next generation of vehicles and engine designs.

Recent growth and speedup of computational engineering tools have worked to leverage computational fluid dynamics as an instrument for practical engineering applications and have enabled the integration of many design considerations, such as aircraft icing, earlier in the design process. Icing is one such design consideration. Traditional computational icing tools have been primarily focused on simulating ice accretion on cantilevered wings. For these configurations, NASA’s two-dimensional and quasi-three-dimensional tools (LEWICE [1] and LEWICE3D [2], respectively) are widely utilized, having been validated against an extensive database of ice-shape geometries [3, 4]. Current and future icing challenges are more general, requiring a fully three dimensional ice prediction tool. These challenges include icing on probes and radomes, engine icing, and unconventional configurations like truss braced wings and urban air mobility vehicles [5].

A significant collaborative research effort was recently completed whereby high-quality, benchmark icing and aerodynamic databases were developed to evaluate icing simulation tools for a large-scale swept wing based on the Common Research Model (CRM). NASA, the Federal Aviation Administration (FAA) and the French Aerospace Lab (ONERA) sponsored this research and has been referred to as SUNSET II or ‘SWIP’ [6]. To meet the aforementioned computational needs, a shift is required from the two and quasi-three-dimensional approaches to a more generalized, fully three-dimensional approach for arbitrary geometries.

The Glenn Icing Computational Environment (GlennICE) [7] is a computational tool designed to calculate ice growth on complex three-dimensional geometries using input from a user-supplied computational fluid dynamics solution for a geometry of interest. The NASA John H. Glenn Research Center at Lewis Field is developing this tool to aid those evaluating, designing, and certifying aircraft, engines, and aircraft components for flight in icing conditions. This domestically available software is being developed to introduce new icing physics into a computational environment in a manner open for evaluation and eventual use by industry, academia, and other government organizations.

Recent advancements in GlennICE have focused on several key areas, including the efficient determination of trajectories, the convergence of collection efficiency, enhancements to the automatic refinement methodology, and benchmarking the solver in a parallel computing environment [8–11]. These developments have resulted in a notable

*Research Aerospace Engineer, Icing Branch, 21000 Brookpark Rd, MS 11-2, AIAA Member

†Research Aerospace Engineer, Icing Branch, 21000 Brookpark Rd, MS 11-2, AIAA Member

reduction in computational time. Moreover, they have highlighted the potential for a significant increase in efficiency, with preliminary findings suggesting the possibility of achieving a twofold enhancement through the implementation of workload balancing improvements. This paper conducts a preliminary analysis of the solver's new dynamic scheduling method, providing valuable insights into the resultant performance enhancements and efficiency gains.

II. Methodology

A. CFD Solver Requirements

The workflow for computing ice shapes for GlennICE can be seen in Fig. 1. GlennICE uses a discretized flow field generated from a computation fluid dynamics solver. To maintain flexibility with a wide variety of commercial and in-house CFD solvers, the coupling between GlennICE and the CFD software are entirely one-way coupling, as such GlennICE can be classified as a CFD post-processor. This coupling level has been sufficient for single-shot external icing problems. However, it should be noted that many storage variants are associated with GlennICE. Computational efficiency can be achieved in various methods, including memory parallelization of the particle trajectory routines in GlennICE.

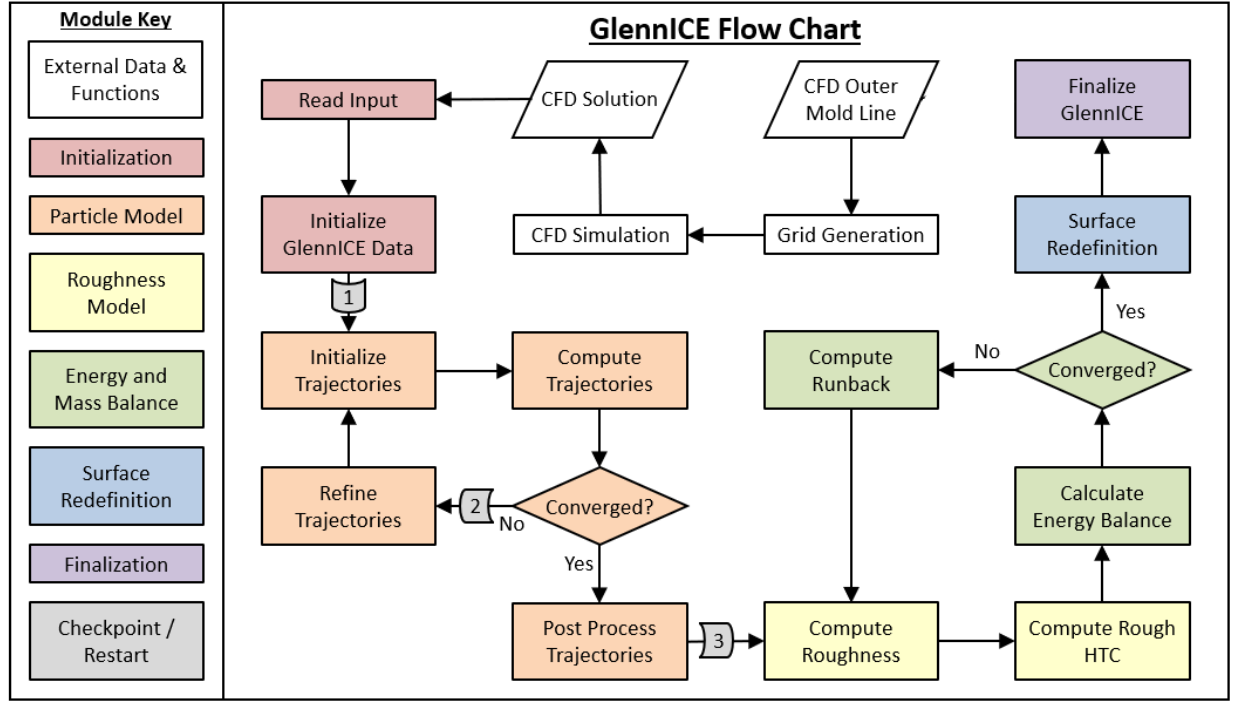


Fig. 1 Workflow of a computational ice accretion simulation using GlennICE.

B. Particle Trajectories

Computational methods for ice accretion can simulate the impact of water drops and ice crystals on an aircraft surface in a Lagrangian or Eulerian reference frame. In an Eulerian reference frame, particles are considered a continuous fluid, while individual particle trajectories are calculated in a Lagrangian frame. Methods that utilize the Eulerian reference frames are typically easier to develop, as established modules used for continuum mechanics can be leveraged. The Eulerian systems [12] can provide results since the user does not have to simulate millions of particles to achieve converged results. Eulerian systems, however, typically use the same grid for the flow solution and the particle flow. A grid designed to capture gradients in the airflow can only capture large gradients in the water flow if a more refined grid is used, reducing the computational advantages available for an Eulerian system. The critical issue is that mass loss of the fluid particles can occur within an Eulerian reference frame due to numerical dissipation, which can be unfavorable

for icing applications. Commercially available tools such as ANSYS's FENSAP-ICE and SIEMENS have addressed simulating icing encounters over external geometries using Eulerian methods [12].

The Lagrangian reference frame, on the other hand, is more physically accurate. Particle flows may not represent a continuum, especially when dealing with particle flows such as splashing, break up, high turbulence, and turbulent mixing. A Lagrangian routine can more easily use flow solutions from any CFD solver. They are also better at capturing gradients in the particle flow when using a mesh refined for CFD. NASA's GlennICE, a Lagrangian ice-accretion tool, simulates the icing encounter over many external geometries [13].

GlennICE integrates particle trajectories using a method developed by Dormand and Prince (DOPRI5) [14]. It is a 4th order Runge-Kutta scheme that is designed to minimize the error of the 5th order term. This means that it computes both the fifth-order and fourth-order results and returns the difference as the error. The algorithm was developed by Hairer et al. [15] and is the default integrator used by MATLAB and GNU Octave. The interface between the DOPRI5 routines and GlennICE resides in a subroutine containing the derivative function, which is the equation of motion in the Lagrangian reference frame. Particles are released from a user-supplied bounding box and subsequently refined based on the distance of the trajectory to the surface or whether it hits the desired icing surface.

III. Parallel Computing Techniques and Optimization Strategies

A. Memory Parallelization

One standard procedure to deal with large computations (such as the particle trajectory routines in GlennICE) is to use some form of parallel processing. Parallel computing is the type of computation in which many calculations or processes are carried out simultaneously [16]. In a parallel environment, more extensive, time-consuming computations are broken down into multiple smaller ones; small computations are then all calculated simultaneously.

An important sub-classification of parallel computing is performed according to their memory distribution. Shared Memory processors or multiprocessors share a global memory space. The essential feature of the shared-memory model, seen in Fig. 2, is that any processor (or CPU) can access all the node's memory available to the machine. This is a beneficial parallel implementation. Primarily when the same calculation is performed at different locations in a domain, such as within an Eulerian approach. However, memory access becomes a severe bottleneck as the number of processors increases when using a shared memory parallelization algorithm.

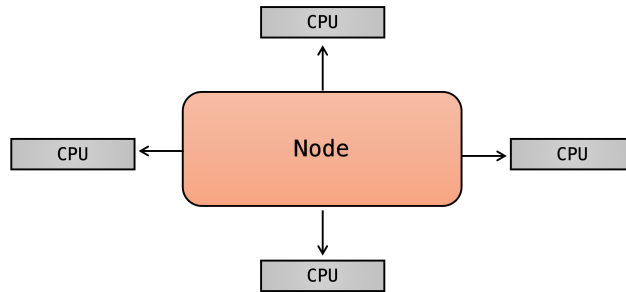


Fig. 2 A schematic of a shared memory system.

On the other hand, a Distributed Memory multiprocessor refers to a parallel system in which each processor has its private memory on a network. On a distributed memory processor, a memory address is specific to each processor, and computational tasks can only operate on local data, as seen in Fig. 3. If processors need to interact with each other to exchange data, interconnection communications are needed. This communication process must not interfere with any particular action on the part of the process whose local memory is being read or written to; otherwise, the communication process will be costly. A distributed memory parallelization is more practical than a shared memory distribution for a Lagrangian reference frame solver. The vital issue with distributed memory is how to distribute the data over each processor efficiently. For the same number of processors, the cost of distributed-memory systems is usually less than that of a shared-memory environment.

Distributed shared memory multiprocessors refer to a parallel environment where remote memories can be addressed as a single shared address space. This can be seen as a hybrid between distributed and shared memory parallelization. Distributed shared memory can scale well with a large number of nodes and can handle complex jobs without repetition

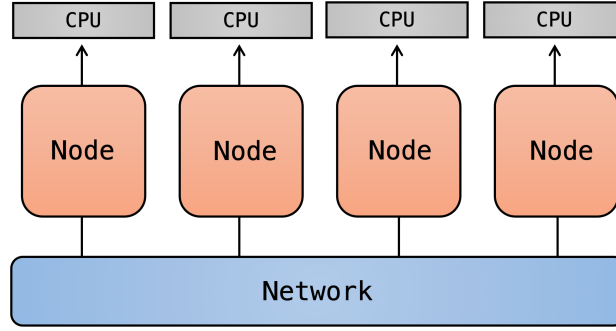


Fig. 3 A schematic of a distributed memory system joined by a network.

while limiting the communication between remote processors. In this configuration, a single copy of data is placed in shared memory, and multiple processors can retrieve or broadcast this data without creating additional copies. Distributed shared memory broadcasting reduces memory consumption and communication overhead and enhances data consistency and performance when several processors require access to the same data.

Multiple configurations can be used in a distributed shared memory architecture. One example can be seen in Fig. 4, where the network uses two nodes with four CPUs on each node. In this hybrid configuration, the total number of processors is equal to the sum of the CPUs on each node, eight.

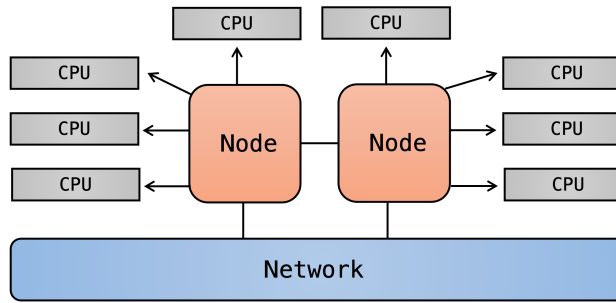


Fig. 4 A schematic of a distributed shared memory system joined by a network.

B. Static and Dynamic Scheduling

Having explored the benefits of shared memory broadcasting in parallel computing, we now focus on task allocation and execution. Choosing between static and dynamic scheduling strategies is pivotal in optimizing computational tasks. Static scheduling is a method used in parallel computing where jobs are given to processors at the beginning of a program's execution. It determines task assignments and execution orders based on predetermined criteria, such as task priorities or a fixed schedule. Static scheduling can ensure predictability and is advantageous in applications with well-defined, balanced workload tasks.

Dynamic scheduling, on the other hand, is a method in parallel computing where task assignments and execution orders are determined dynamically during runtime based on real-time factors such as workload distribution, resource availability, and task priorities. This adaptive method allows for efficient utilization of computational resources and is well-suited for jobs with uneven workload balance. An issue commonly faced with dynamic scheduling is communication cost. Dynamic scheduling involves making real-time decisions about which tasks should be executed by which processors, and these decisions often require communication between processors to coordinate and share task assignments. Failure to minimize this cost can nullify the potential benefits of dynamic scheduling.

Dynamic scheduling is frequently paired with shared memory broadcasting to address this issue. Instead of disseminating data and distributing tasks among processors, all processors maintain a single shared memory copy of the data. The dynamic scheduler then focuses solely on task-to-processor assignments, mitigating communication overhead.

C. Particle Trajectory Memory Parallelization

In previous versions of GlennICE, trajectory routines employed a simplistic static scheduling mechanism [11]. Under this mechanism, trajectories were evenly distributed among processors, with each node assigned a fixed number of unique trajectories. Subsequently, nodes allocated an equal number of trajectories to processors and awaited their completion. Figure 5a and Table 1 depict the static scheduling process for 12 trajectories across four processors. While earlier findings demonstrated a notable reduction in computational time, they also revealed an almost twofold decrease in efficiency. This inefficiency arose from the uneven distribution of workload among trajectories. For instance, trajectories in the freestream required fewer time-step computations compared to those near the surface. Since each processor received an equal number of trajectories, some processors remained idle after completing their tasks, while others engaged in active work. The imbalance in workload allocation led to inefficient resource utilization.

To address this problem, the GlennICE trajectory routines have transitioned to a distributed shared memory configuration along with a dynamic scheduler for particle trajectories. Under this configuration, each trajectory is initially allocated an address within the shared memory of the initialized GlennICE data. Thus, all trajectories have simultaneous access to and a 'view' of the entire domain. Afterwards, a designated processor responsible for decision-making and workload allocation manages the dynamic scheduling process. This processor will have the following responsibilities:

- Communicating trajectories to each processor
 - Involves providing an integer array of where the calculation begins and how many trajectories are to be computed.
- Tracking the status of completed trajectories.
- Monitoring processors engaged in active computations.
- Managing processors that have finished a task on a trajectory and require additional work.

The manager processor however, will not engage in the calculation of any trajectories. While it may appear counter-intuitive to sacrifice one processor's capacity for efficiency gains, as the number of processors increases, the impact of losing one processor diminishes and the solver's efficiency approaches 100%. The current workflow for dynamic scheduling is structured as follows:

- 1) The manager processor initiates the distribution of an initial subset of trajectories among processors.
- 2) Each processor independently calculates its allocated subset of trajectories.
- 3) Whenever any processor completes its work on the assigned subset of trajectories, it requests an additional subset from the manager processor.
- 4) Steps (1), (2), and (3) are repeated iteratively until all trajectories have been calculated.

A notable practical advantage of the new algorithm is its adaptability, allowing the dynamic scheduler to mimic a static scheduling. A new variable, referred to as `chunk_percent`, has been introduced to enable this functionality. This variable acts as a control parameter, influencing the type of scheduler utilized by controlling the distribution of the initial subset of trajectories:

$$[\text{Initial Subset of Trajectories}] = \max \left[\left(\text{chunk_percent} \times \frac{\text{Total Number of Trajectories}}{\text{Number of Processors} - 1} \right), 1 \right]$$

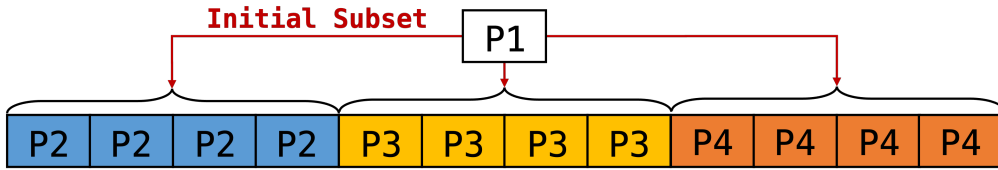
Figure 5 along with Table 1 illustrates dynamic and static scheduling computations over 12 trajectories using four processors. In this example, the initial processor assumes the role of the manager processor, overseeing the scheduling process. Meanwhile, processors 2 through 4 are initially assigned a subset of trajectories depending on the value of `chunk_percent`. The distribution of trajectories is determined based on the progress of each processor's workload, with new trajectories allocated to the processor that completes its tasks first. When `chunk_percent` is set to 100%, the sum of the initial subset of trajectories from step (1) equals the total number of trajectories, and a static scheduling routine is retained (Figure 5b). When set to zero, it triggers a pure dynamic routine where each processor receives just one trajectory (Figure 5e). Any value between zero and 100% represent a seamless transition between static and dynamic scheduling, (Figures 5c and 5d).

Table 1. Number of trajectories (per processor) for static and dynamic scheduling.

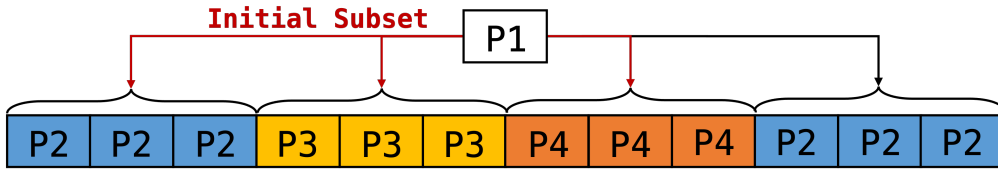
	Processor 1 (Manager)	Processor 2	Processor 3	Processor 4
Static Scheduling	3	3	3	3
Dynamic Scheduling (chunk_percent = 100%)	0	4	4	4
Dynamic Scheduling (chunk_percent = 75%)	0	6	3	3
Dynamic Scheduling (chunk_percent = 50%)	0	4	2	6
Dynamic Scheduling (chunk_percent = 0%)	0	5	2	5



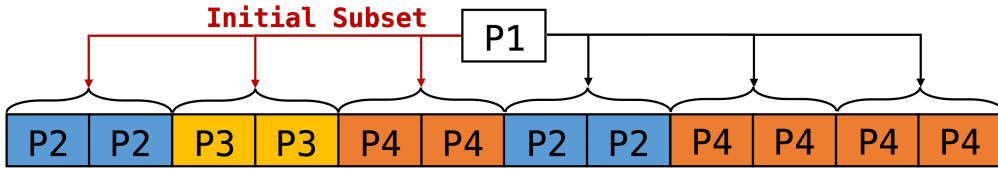
(a) Static scheduling.



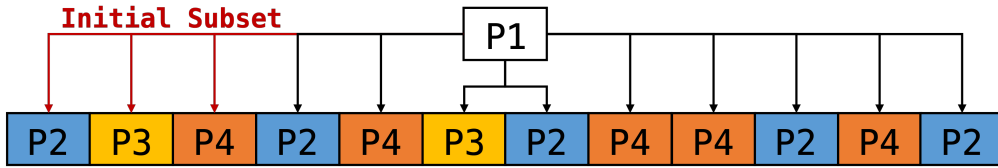
(b) Dynamic scheduling (chunk_percent = 100%).



(c) Dynamic scheduling (chunk_percent = 75%).



(d) Dynamic scheduling (chunk_percent = 50%).



(e) Dynamic scheduling (chunk_percent = 0%).

Fig. 5 Trajectory routine for static and dynamic scheduling.

IV. Benchmarking Approach

A. Computational Resources

In the pursuit of enhancing GlennICE’s efficiency in handling increasingly complex geometries, two viable options stand out: utilizing a workstation equipped with multiple processors or harnessing the power of a High-Performance Computing Cluster (HPCC). The local workstation employed for this paper’s analysis features two sockets with two physical processors on the motherboard, totaling 36 cores available for utilization. Each processor, detailed in Table 2, is an Intel Xeon Gold 6140 system processor. To ensure precision in calculations across processors, the turbo-boost feature, which temporarily elevates a CPU’s clock speed above its base frequency to accommodate resource-intensive tasks, will be intentionally disabled.

Table 2. Local workstation node details.

Number of Cores	18
Number of Threads	36
Node Type	Intel Xeon Gold 6140 Processors
Processor Speed	2.30 GHz
Max Turbo Frequency	3.70 GHz
Total # of Nodes Available	1

On the contrary, the utilization of an HPCC cluster enables the execution of larger and more intricate simulations or data analyses within significantly shorter timeframes compared to those achievable on a single machine. An HPCC cluster comprises numerous separate nodes interconnected via a network. The NASA Advanced Supercomputing (NAS) Division provides a cohesive, high-performance computing environment to expedite NASA missions and facilitate groundbreaking advancements and discoveries. A notable supercomputer within NASA’s Advanced Supercomputing division is Pleiades, a distributed-memory SGI/HPE ICE cluster interconnected with InfiniBand in a dual-plane hypercube technology. InfiniBand, tailored for high-performance computing environments like supercomputers, boasts low latency, high bandwidth, and efficient data transfer capabilities, rendering it ideal for parallel computing tasks. Pleiades hosts various Intel Xeon processor types, including E5-2680v4 (Broadwell), E5-2680v3 (Haswell), E5-2680v2 (Ivy Bridge), and E5-2670 (Sandy Bridge) [17, 18]. In this study, the Pleiades Ivy Bridge Nodes will serve as benchmarks for evaluating GlennICE’s computational performance, with configuration details provided in Table 3.

Table 3. Pleiades Ivy bridge node detail.

Number of Cores	10
Number of Threads	20
Node Type	Intel Xeon E5-2680v2 Processors
Processor Speed	2.80 GHz
Max Turbo Frequency	3.60 GHz
Total # of Nodes Available	5,256

B. Common Research Model High Lift

To examine the efficiency of the parallel scheme implemented within GlennICE, a case and geometry were chosen that help illustrate the code’s applicability to relevant engineering problems. A geometry that meets the criteria outlined is the Common Research Model (CRM). The Common Research Model is studied by industry, academia, and government to leverage high-quality open experimental data on applicable problems of interest to the community. The large physical size of the CRM wing presented specific challenges to the ice-accretion testing for the IRT [19]. As a result variety of subtly different versions exist within the CRM atmosphere. For instance, a 65% size scaled-down version of the CRM the full scale, baseline, and reference geometry was used for SUNSET II effort [6]. Other configurations, such as the High-Lift CRM (CRM-HL), include design changes that aim to study high-lift setups on relevant engineering

problems collaboratively. The CRM-HL was the focus of the 4th AIAA High Lift Prediction Workshop [20], with Fig. 6 illustrating one of the high lift configurations from that effort.

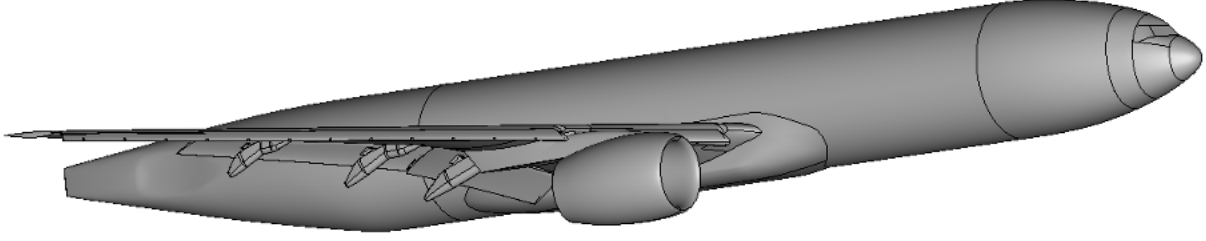


Fig. 6 Depiction of the HLPW-4_CRM-HL_40-37_Nominal_v2.igs model from the 4th AIAA High Lift Prediction Workshop.

There are a variety of icing scenarios that can impact the performance of a vehicle. Table 4 illustrates a high-level description of common icing scenarios relevant to the geometry being examined. Typically, the most critical ice shapes are those accreted in the stowed configuration. Figure 7 depicts the CRM-HL model in the stowed configuration.

Table 4. Computational icing configurations of interest.

Accretion Configuration	Performance Degradation Configuration
Stowed	Stowed
Stowed	Deployed
Deployed	Deployed
Stowed & Deployed	Deployed

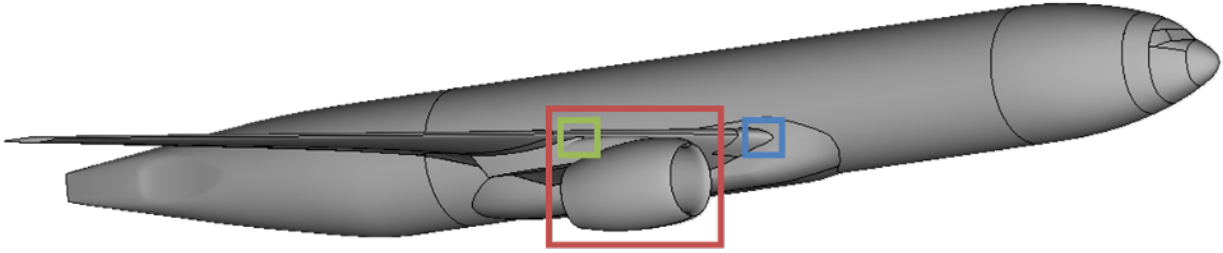


Fig. 7 Depiction of the CRM-HL in the stowed configuration, with colored boxes showing the location of the surface mesh depicted in Fig. 8

Simulated Ice Shapes on a Common Research Model High Lift using LEWICE3D and GlennICE have been generated [13]. The same triangular surface and tetrahedral volume mesh generated from the reference is used here, shown in Fig. 8. NASA's Computational Fluid Dynamics solver FUN3D [21] was used to solve for a steady flow solution around the uniced vehicle. The focus, however, will be on the parallelization of the calculated trajectories by GlennICE and the impact locations as highlighted in Fig. 9.

C. Refinement Restart

While this study's primary focus lies in assessing the parallelization scheme, it is crucial to acknowledge the impact of serial routines on GlennICE's performance. Specific tasks, such as reading the CFD grid, processing the flow solution, and generating the GlennICE grid can demand substantial computational resources within GlennICE. The time

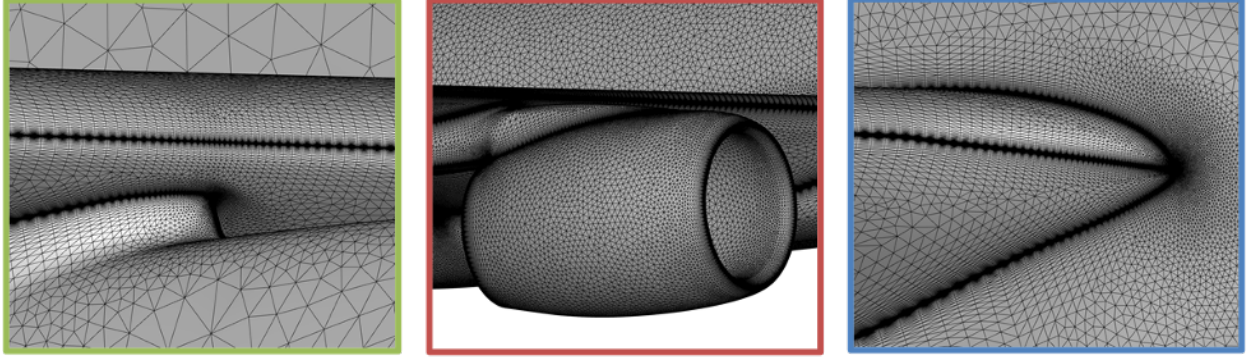


Fig. 8 Depictions of the surface mesh on the CRM-HL in a stowed configuration.

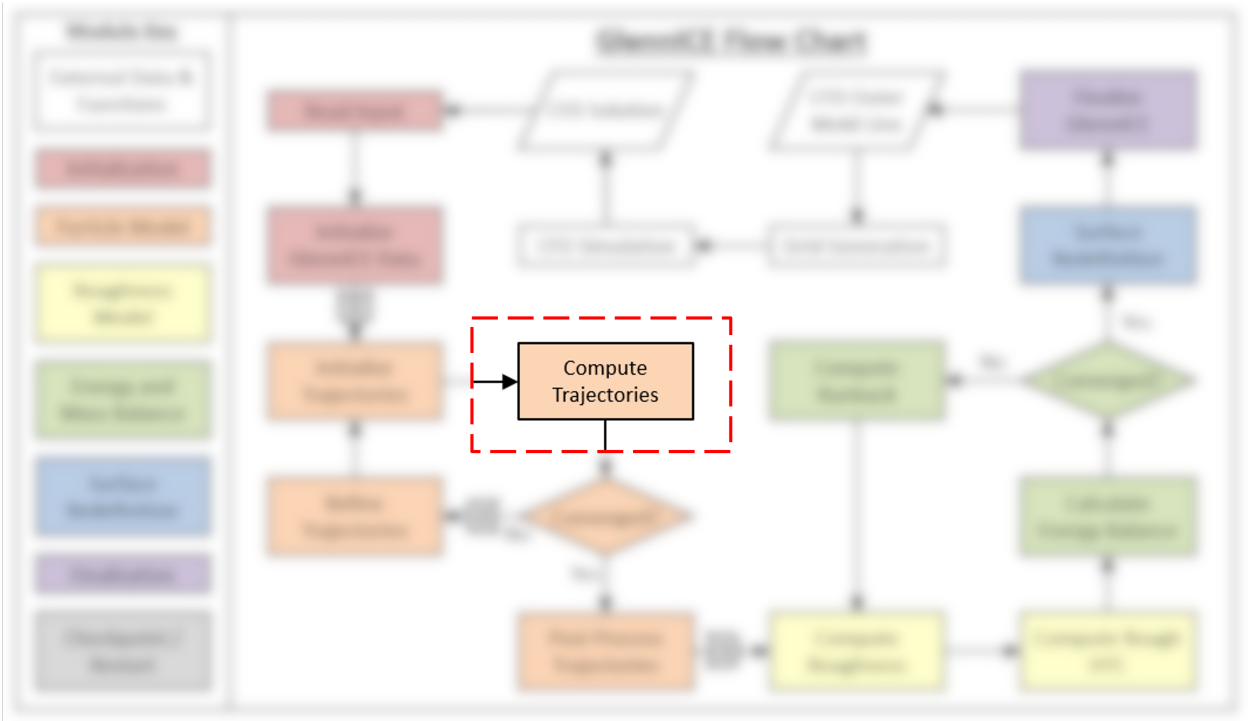


Fig. 9 Implementing MPI in a workflow of a computational ice accretion simulation using GlennICE.

consumed by these serial routines serves as a foundational bottleneck, establishing a theoretical limit to the speedup achievable in a parallel computing environment. To extract the pure effects of parallelization, leveraging GlennICE's refinement restart capability, introduced in version 2.2.0 [7]. With this feature, GlennICE generates a refinement restart file once the initial icing computation concludes. Running GlennICE using this restart file significantly reduces serial processing time, as it utilizes existing trajectories for collection efficiency calculations and adds further trajectory refinement to the ongoing computation. Most importantly, this approach eliminates the need for all serial tasks within GlennICE, enabling us to conduct an isolated parallelization study.

V. Results

The present study utilizes the stowed configuration of the CRM-HL shown previously in Figures 6, 7, and 8. The hold conditions are shown in Table 5.

A refinement restart file was generated from the initial run that had been completed. The refinement restart file calculates 498,267 additional trajectories, bringing the total trajectories computed by GlennICE to 804,612. A metric

Table 5. Run conditions for CRM-HL.

Altitude (ft)	Mach	AoA (deg)	Diameter (μm)	T ($^{\circ}\text{F}$)
15,000	0.4	3.5 $^{\circ}$	20	21.2

known as speedup is calculated to quantify how much faster a parallel system can complete a task or computation when multiple processors or cores are utilized. Speedup measures the relative performance improvement of a parallel algorithm or system compared to a sequential (single-processor) implementation:

$$\text{Speedup} = \frac{\text{Computation Time For 1 Processor}}{\text{Computation Time For } x \text{ Processors}} \quad (1)$$

Using the speedup definition, an efficiency is calculated as follows:

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of Processors}} \quad (2)$$

A. Local Workstation

The computational approach employed for generating results on a local workstation follows a systematic procedure. Initially, a run is executed on GlennICE utilizes a single processor. Subsequently, GlennICE is re-executed, this time utilizing its refinement restart feature while recording the computation time. This iterative process is repeated for varying processor counts, including 1, 2, 4, 8, 16, 24, and 32. Each computation is repeated 30 times for every processor configuration and an average computation time is recorded. The turbo-boost feature is turned off for these computations to ensure proper comparisons between test cases. Finally, the speedup is computed to evaluate the performance enhancements achieved through parallelization.

With the processor sweep and methodology established, the findings are presented in Fig. 10 and detailed in Table 6. As shown in Fig. 10, dynamic scheduling (with `chunk_percent` = 0%) initially exhibits a noticeable decrease in efficiency due to the involvement of a single manager processor. However, as the number of processors increases, the impact of losing one processor diminishes and the solver's efficiency steadily approaches 100%. Notably, the results indicate that communication costs have minimal influence on the outcomes, despite the use of maximum inter-processor communication in a purely dynamic scheduler setup. This can be attributed to transmitting all initialization data before the work phase, leveraging a shared memory implementation. Consequently, real-time communication during the work phase only involves an integer specifying which initialization data the remote process should handle, as the corresponding data is already shared, eliminating the need for additional communication.

In contrast, static scheduling demonstrates high efficiencies initially, owing to reasonably balanced load distribution among processors. However, as the number of processors scales up, workload imbalances become apparent, resulting in an inevitable decline in efficiency. Evidence of this workload imbalance is illustrated in Fig. 11, where particles encountering the geometry or residing in the free stream necessitates fewer time steps for integration. Where trajectories narrowly avoiding the surface require significantly more time-step calculations. In a static scheduling scheme, processors are assigned equal trajectories. The uneven workload distribution among trajectories leads to disparate workloads among processors, resulting in a significant decline in efficiency with an increasing number of processors.

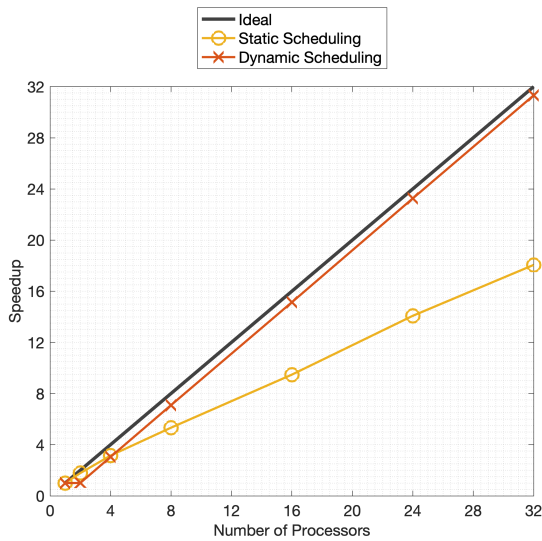
B. HPCC

Continuing from the discussion on the local workstation, a similar procedure is followed for obtaining results on a computer cluster. This exploration, however, aims to analyze the solver's performance with multiple nodes, which may introduce overhead in inter-node communication. Initially, a run on GlennICE is conducted using just one processor, followed by subsequent runs utilizing its refinement restart feature for varying processor counts, ranging from 1 to 320 processors. Increasing the number of processors is achieved by leveraging additional nodes for the computation. Adding each node allows for the utilization of up to 20 additional processors. The computation is repeated five times for each processor configuration, and an average computation time is recorded, followed by the calculation of speedup to assess the parallelization's performance enhancements.

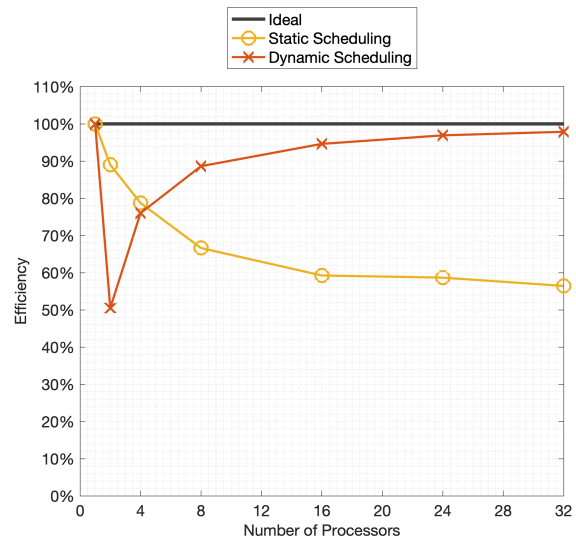
With the methodology established, the outcomes are detailed in Fig. 12 and Table 3. As depicted in Fig. 12, dynamic scheduling (`chunk_percent` = 0%) initially experiences a noticeable efficiency drop due to the involvement

Table 6. Parallel efficiency and speedup performance on a local workstation.

Number of Processors	Static Scheduling			Dynamic Scheduling		
	Time (min)	Speedup	Efficiency	Time (min)	Speedup	Efficiency
1	277.6	1.00	100.00%	278.0	1.00	100.00%
2	155.7	1.78	89.11%	274.6	1.01	50.62%
4	88.0	3.15	78.85%	91.5	3.04	75.95%
8	52.1	5.33	66.59%	39.2	7.09	88.58%
16	29.3	9.48	59.23%	18.3	15.15	94.86%
24	19.7	14.08	58.65%	12.0	23.23	96.92%
32	15.4	18.06	56.45%	8.9	31.32	97.87%



(a) Parallel speedup performance.



(b) Parallel efficiency performance.

Fig. 10 Parallel performance on a local workstation.

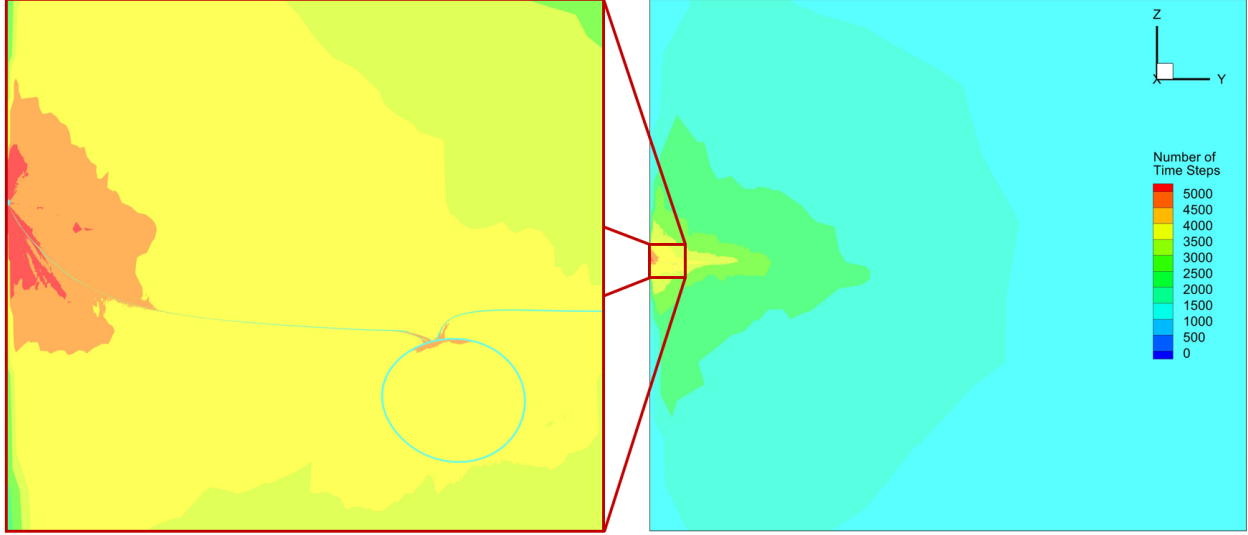


Fig. 11 Contour of the number of time steps for the CRM-HL.

of a single manager processor. Unlike the local workstation scenario, the solver's efficiency does not approach 100% as the number of processors increases. This prompts the question: Does the efficiency drop stem from the supercomputing system's architectural and design aspects ?

Recall from Section IV.A, each Ivy bridge node comprises 2 Ten-Core processors with a base speed of 2.8GHz and a max turbo frequency of 3.60 GHz. The turbo boost feature dynamically increases the clock speed of one or more cores beyond their base frequency, enhancing performance in tasks requiring additional processing power. This activation typically occurs when a low number of cores are utilized, such as in a single processor case. However, in Eq. (1), the speedup calculation relies on the ratio of computation time between one processor and x number of processors. If the computation time on one processor operates at the maximum turbo frequency while other test cases run at regular processor speed, significant drops in speedup may occur.

To address this issue, we can either deactivate the turbo-boost feature (similar to the approach taken on a local workstation) or reconfigure the cores-to-node arrangement to enable turbo boost for each processor. The latter option involves ensuring a single processor exists on each node. For instance, if the computation utilizes 120 processors, 120 nodes would be requested instead of 6 nodes with 20 processors on each node. Thus, the processor sweep is repeated, ranging from 1 to 320 processors, using one core per node. Analogous to the workstation scenario, as the number of processors increases, the impact of losing one processor diminishes, and efficiency approaches 100%.

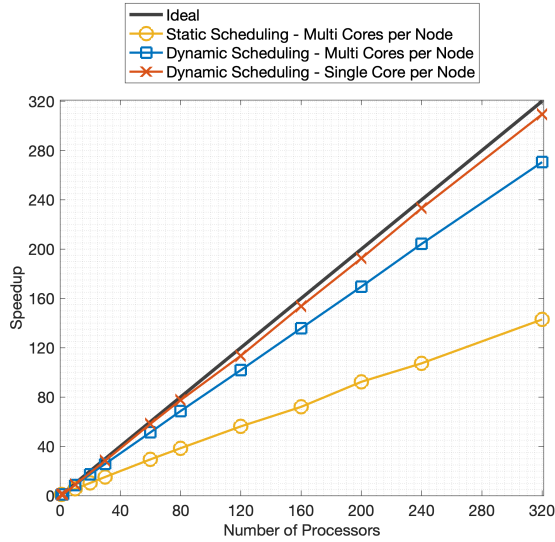
It is crucial to acknowledge that the efficiency outcomes of static scheduling depicted in Fig. 12 are contingent on the specific case, given the variability in load balance across cases. Static scheduling efficiencies for other geometries have been documented to span from 5% to 90%. The inconsistency observed across test cases for static scheduling, coupled with the uncertainty regarding the efficiency metric beforehand highlights dynamic scheduling as a more resilient option across various geometries, yielding consistent efficiency results.

VI. Conclusion

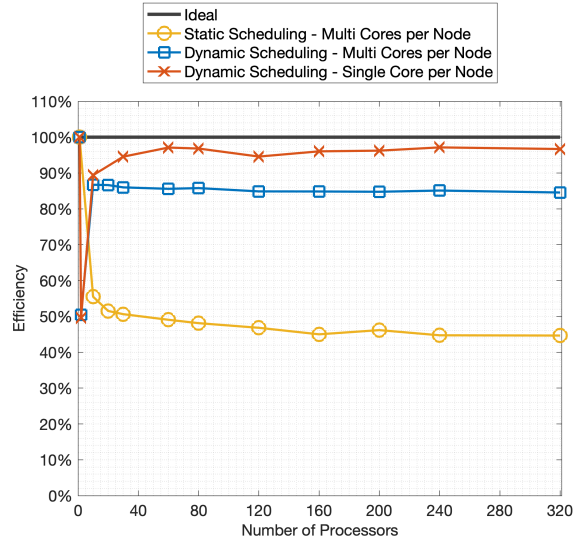
The primary objective of this study was to highlight the effectiveness of parallelization within GlennICE to enhance computational performance. The findings provide valuable insights into the impact of system architecture and design on computational efficiency. Results shed a light on how workload distribution and communication overheads influence overall performance for static and dynamic scheduling. A noticeable two-fold increase in efficiency was achieved through shared memory and dynamic scheduling routines. Unlike its counterparts of static scheduling, these routines also yield consistent efficiency metrics across any geometry. Examination of processor configurations, including the utilization of turbo-boost features, also highlights the significance of resource configuration in optimizing computational efficiency. Finally, the findings from this work indicate a minimal communication cost between processors, enhancing the feasibility of implementing an efficient dynamic scheduler algorithm.

Table 7. Parallel efficiency and speedup performance on the Pleiades HPCC cluster.

Number of Processors	Static Scheduling (Multi-Cores per Node)			Dynamic Scheduling (Multi-Cores per Node)			Dynamic Scheduling (Single Core per Node)		
	Time (min)	Speedup	Efficiency	Time (min)	Speedup	Efficiency	Time (min)	Speedup	Efficiency
1	395.07	1.00	100%	404.39	1.00	100.00%	397.94	1.00	100%
2	207.90	1.90	95.01%	401.15	1.01	50.40%	401.15	0.99	49.60%
10	71.20	5.55	55.49%	46.63	8.67	86.72%	44.52	8.94	89.39%
20	38.34	10.30	51.52%	23.34	17.33	86.64%	21.60	18.42	92.11%
30	26.01	15.19	50.62%	15.68	25.80	85.99%	14.03	28.37	94.56%
60	13.42	24.43	40.72%	7.88	51.35	85.59%	6.83	58.26	97.10%
80	10.26	38.49	48.12%	5.89	68.63	85.79%	5.14	77.46	96.83%
120	7.03	56.19	46.82%	3.97	101.85	84.87%	3.51	113.50	94.59%
160	5.49	72.01	45.00%	2.98	135.76	84.85%	2.59	153.66	96.04%
200	4.28	92.35	46.17%	2.39	169.53	84.77%	2.07	192.47	96.24%
240	3.68	107.36	44.73%	1.98	204.26	85.11%	1.71	233.15	97.14%
320	2.76	142.92	44.66%	1.49	270.59	84.56%	1.29	309.40	96.69%



(a) Parallel speedup performance.



(b) Parallel efficiency performance.

Fig. 12 Parallel performance on the Pleiades HPCC cluster.

Acknowledgments

The authors would like to thank NASA's Advanced Air Transport Technology (AATT) and Transformational Tools and Technologies (TTT) projects for their financial support of this research effort. The NASA's Advanced Supercomputing Division (NAS) for computing resources needed to obtain the results presented. In addition, the authors would like to thank the GlennICE development group (Eric Galloway, David Rigby, William Wright, and Thomas Ozoroski) for their assistance and efforts.

References

- [1] Wright, William Butler. "User Manual for the NASA Glenn Ice Accretion Code LEWICE. Version 2.2.2." (1999)., [NASA/CR-2002-211793](#), Aug., 2002.
- [2] Bidwell, C. S., and Potapczuk, M. G., "User's Manual for the NASA Lewis Three-Dimensional Ice Accretion Code (LEWICE 3D)", [NASA TM-105974](#), Dec., 1993.
- [3] Wright, W. A summary of validation results for LEWICE 2.0. In 37th Aerospace Sciences Meeting and Exhibit. <https://doi.org/10.2514/6.1999-249>
- [4] Wright, W. B., "Validation Results for LEWICE 3.0", [AIAA-2005-1243](#)
- [5] Richard, E. K., Andy, P. B., Ru-Ching, C., & Peter, M. S. Aircraft Icing Analysis of Alternatives. In AIAA AVIATION 2022 Forum. <https://doi.org/10.2514/6.2022-4024>
- [6] Broeren, A. P., Potapczuk, M., Riley, J., Villedieu, P., Moens, F., & Bragg, M. Swept-Wing Ice Accretion Characterization and Aerodynamics. In 5th AIAA Atmospheric and Space Environments Conference. <https://doi.org/10.2514/6.2013-2824>
- [7] Porter, Christopher, Mark Potapczuk, Thomas Ozoroski, Zaid Sabri, Eric Galloway, David Rigby, William Wright, and Paul Tsao. "GlennICE Manual 4.1.0." NASA/TM-20240002191, 2024. <https://ntrs.nasa.gov/citations/20240002191>.
- [8] Wright, W. B., Porter, C. E., Potapczuk, M. G., Galloway, E., & Rigby, D. L. An Automated Refinement Process for Particle Trajectory Methods in GlennICE. In AIAA AVIATION 2021 FORUM. <https://doi.org/10.2514/6.2021-2631>
- [9] Rigby, D. L., & Wright, W. B. Convergence Criteria for Lagrangian Collection Efficiency Simulations. In AIAA AVIATION 2022 Forum. <https://doi.org/10.2514/6.2022-3693>
- [10] Porter, C. E. A Comparison of Trajectory Refinement Schemes for GlennICE. In AIAA AVIATION 2022 Forum. <https://doi.org/10.2514/6.2022-3692>
- [11] Sabri, Z. and Porter, C., "Scalability of GlennICE in a Parallel Environment," SAE Technical Paper 2023-01-1482, 2023, <https://doi.org/10.4271/2023-01-1482>.
- [12] Morency, F., Beaugendre, H., Baruzzi, G., & Habashi, W. FENSAP-ICE - A comprehensive 3D simulation system for in-flight icing. In 15th AIAA Computational Fluid Dynamics Conference. <https://doi.org/10.2514/6.2001-2566>
- [13] Andy P. Broeren, Thomas Ozoroski, Christopher E. Porter (2023). Simulated Ice Shapes on the High Lift Common Research Model Using LEWICE3D and GlennICE
- [14] J., R., Dormand., P.J., Prince. (1980). A family of embedded Runge-Kutta formulae. Journal of Computational and Applied Mathematics, 6(1):19-26. doi: 10.1016/0771-050X(80)90013-3
- [15] Ernst Hairer, G. W., Syvert P. Norsett. Solving Ordinary Differential Equations I. Springer Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-78862-1>
- [16] Almasi, G. S., & Gottlieb, A.(1989). Highly Parallel Computing. Benjamin/Cummings.
- [17] NAS. (n.d.). NASA Advanced Supercomputing (NAS) Division. Retrieved March 27, 2023, from <https://www.nas.nasa.gov/>
- [18] HECC. (n.d.). High-End Computing Capability. NASA. Retrieved March 27, 2023, from <https://www.nas.nasa.gov/hecc/support/kb/>
- [19] Tsao, J.-C., & Porter, C. E. Characterization of Collection Efficiency of the Common Research Model Midspan Wing Section in the IRT. In AIAA AVIATION 2021 FORUM. <https://doi.org/10.2514/6.2021-2681>
- [20] Rumsey, C. (2022, July 12). 4th AIAA CFD High Lift Prediction Workshop (HLPW-4) <https://hilitpw.larc.nasa.gov/>
- [21] FUN3D. Software Package. NASA Langley Research Center, Hampton, VA