Development of an Aircraft Aero-Icing Suite Using Chapel Programming Language

Hélène Papillon Laroche, master student
Simon Bourgault-Côté, Research associate
Matthieu Parenteau, Ph. D. candidate
Éric Laurendeau, Professor

Fièrement propulsée par les FRQ

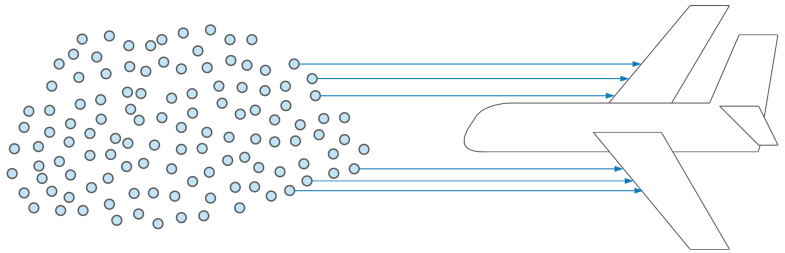
NSERC CRSNG

POLYTECHNIQUE MONTRÉAL

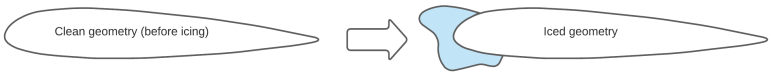
TECHNOLOGICAL UNIVERSITY

Table of contents

Aircraft Icing Phenomena

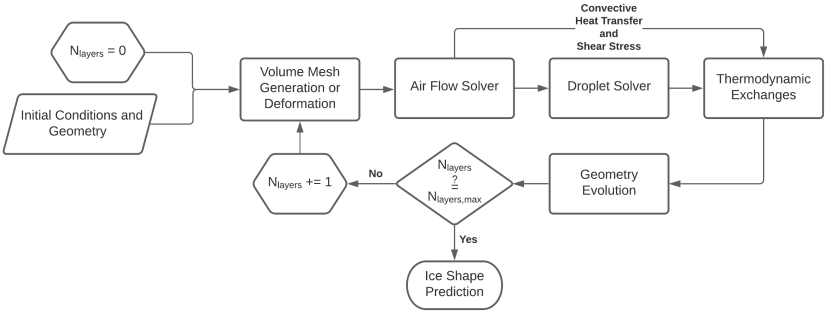


Impact of the ice accumulation



- Ice accretion leads to changes in the airfoil geometry;
- Performances are greatly affected by these changes → Security issues.

Overview of Aircraft Icing Modelisation



Complex multi-physical phenomena

- Airflow field and droplets impingement $\rightarrow$ volume mesh;
- Thermodynamic exchanges and geometry evolution $\rightarrow$ surface mesh;
- Evolution in time of the computational domains (since the geometry changes).

Why Use Chapel?

Challenges of multi-physics simulations

We have to balance :

- the fidelity of multiple solvers;
- the performances → computational costs;
- the productivity → addition of multiple physical models.

Chapel was initially used for the implementation of a flow solver : **Ch**apel **M**ulti-**P**hysics **S**imulation (CHAMPS) [2].

Benefits from Chapel's features

- Productivity → fast prototyping with high level syntax;
- Natively distributed → Overcome the barrier of entry of parallel distributed programming in an academic context (2 years) [2];
- Modularity → Generic classes and records to reuse structures;
- Memory management strategies.

Data Structure for Multi-Physics Simulations

Overview of what was previously implemented

- Object oriented structure with generic base classes;
- Multi-zones structure suitable for distributed memory computations with interface exchanges (communication between computational nodes) [2].

What we added to complete the multi-physic framework

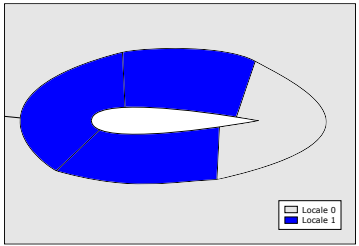
- Treatment of the surface mesh;
- Extension of the multi-zones structure to the additional modeled physics.

Challenges of Multi-Physic Simulations

Surface and Volume meshes

Distribution of the zones across the computational nodes
$\rightarrow$ to better distribute the *volume mesh*.

- Adapted for the flow and droplet solvers (resolution on the volume mesh);
- Can lead to an unbalanced distribution across the computational nodes for the thermodynamic exchanges and the geometry evolution.



Challenges of Multi-Physic Simulations

How can we re-balance the distribution and reduce under-utilized resources?

No redistribution

With redistribution

Distribution handling

The *GlobalHandle_c* generic class handles the distribution across the Locales of the computation domain (mesh) [2]:

- Difference between surface and volume meshes → the number of topological dimensions;
- *GlobalHandle_c* is generic as the mesh type is a type alias of the class → it works for any number of topological dimensions;
- The re-distribution of the surface mesh re-uses the *GlobalHandle_c*, as the distribution of the volume mesh.

Challenges of Multi-Physic Simulations

How can we re-balance the distribution and reduce under-utilized resources?

No redistribution

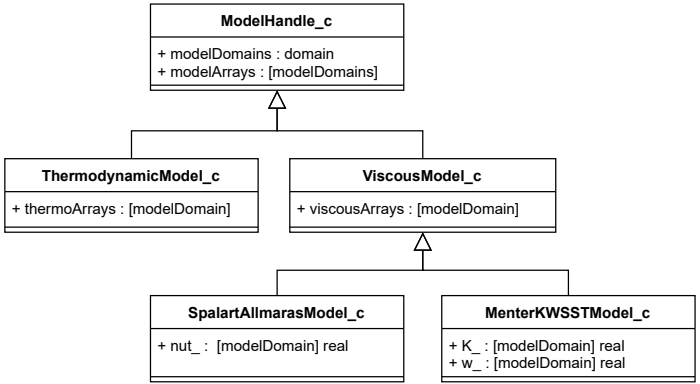
With redistribution

Re-distribution handling

1. Delete the instance of *GlobalHandle_c(VolumeMesh)* $\rightarrow$ by simply exiting the scope since everything uses `owned` memory management;
2. Instantiate again a *GlobalHandle_c*, but with the surface mesh (after reading the latter);
3. With this instance of *GlobalHandle_c*, the surface zones are automatically distributed with a Block distribution across all the Locales.

Challenges of Multi-Physic Simulations

The addition of modeled physics (turbulence, droplet impingement, thermodynamic exchanges and geometry evolution) required a flexible structure to enable a productive framework.



Challenges of Multi-Physic Simulations

ModelHandle_c → simple hierarchical structure but ...

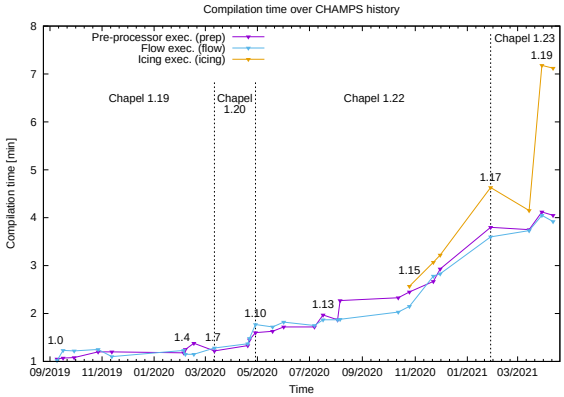
Compilation difficulties encountered throughout the implementation of the new models.

- Combination of object-oriented programming and generic objects for the zones and the models;
- Definition of the methods of the children objects often require a tight control with `where` statements to avoid compilation errors due to non-existing fields or methods in parents or in siblings.

```
where globalHandle.type == GlobalHandleFlow_t || globalHandle.type ==
    BorrowedGlobalHandleFlow_t || globalHandle.type == GlobalHandleIcing_t ||
     globalHandle.type == BorrowedGlobalHandleIcing_t
where zone.type < MeshFlow_c
```

Challenges of the Chapel Implementation

One of the main drawbacks of the growth of CHAMPS with new components or modules is the increase in compilation time and required memory, especially for the icing executable.



Challenges of the Chapel implementation

At its highest point, the compilation time could take around 15 minutes, whereas memory usage was seen to reach up to 30GB of memory (RAM).

Why?

Simple causes (introduced by unfamiliar students with the Chapel language):

- The overuse of generic function arguments in some modules;
- The `use` of too many modules everywhere even when they were not required.

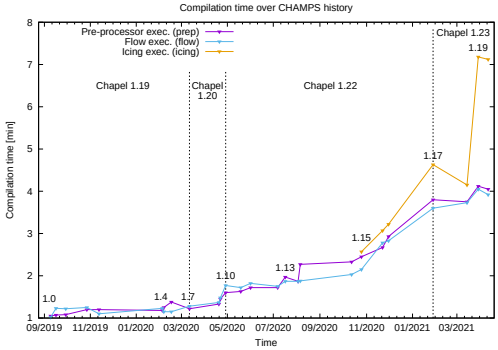
Complex causes :

- The addition of new components in the code (new modules);
- The duplication of generic functions for multiple flavors of the mesh and model objects (even outside programmed combinations).

Challenges of the Chapel implementation

How to reduce the compilation costs?

- Split the compilation in two phases for memory usage:
  1. the generation of the C code from the Chapel files;
  2. the compilation of the C code.
- Address the overuse of generic functions arguments and modules;
- Properly use `where` statements to limit the duplication of generic functions.



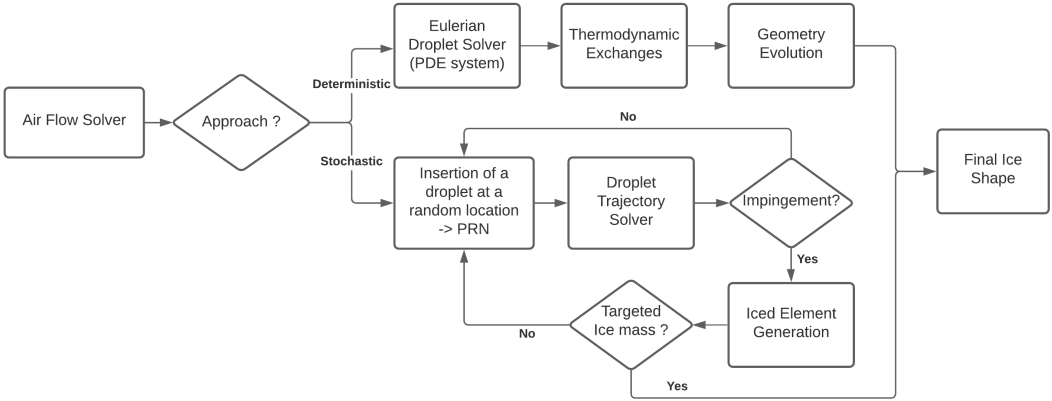
What remains?

- The optimization operation performed after version 1.17 was lost for some reasons in version 1.19;
- An hypothesis lies within the `where` statements discussed before, but it is not yet verified.

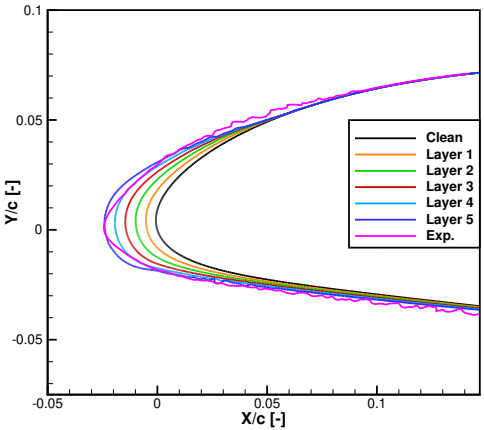
Results

Two approaches are available in CHAMPS :

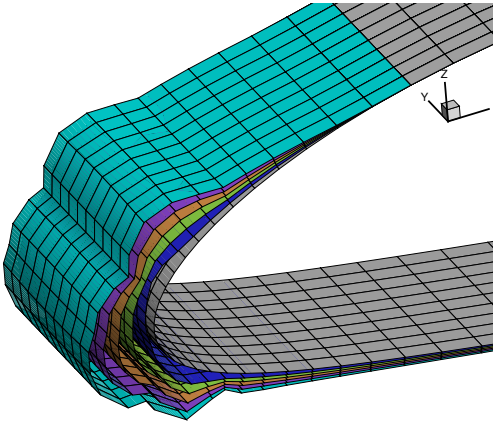
- Deterministic;
- Stochastic.



Results

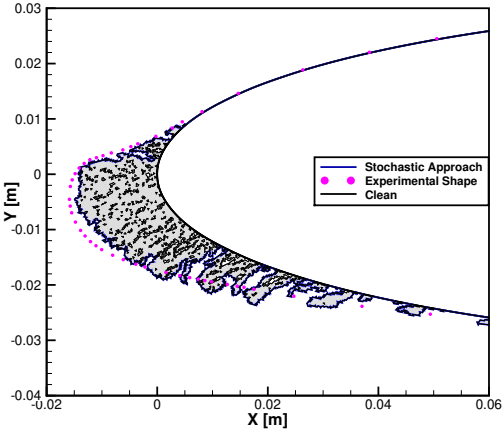


2D rime ice: Case 241 from IPW1 [1]

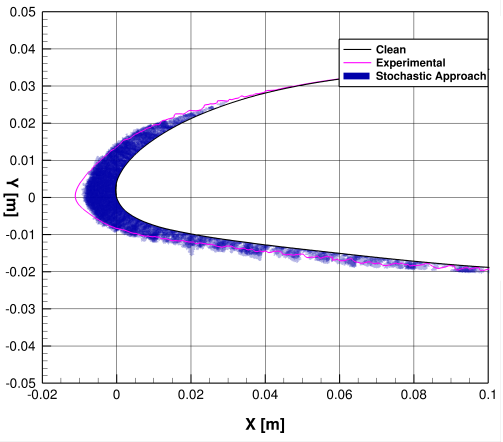


3D rime ice: Case 241 from IPW1 [1]

Results



2D rime ice: Case 01 from [3]



2D rime ice: Case 241 from IPW1 [1]

Outcomes

In an academic context

- Allows the graduate students to focus on the physics they model;
- Enables fast prototyping of various physic models;
  - Aircraft icing is still not well understood.

Participation to the 1st Icing Prediction Workshop

- Brings together organizations to compare icing predictions;
- Code-to-code comparisons to CFD software using more traditional programming languages (Fortran, C, C++);
  - Ansys FENSAP-ICE, NASA LEWICE, ONERA IGLOO3D;
- Allows us to assess the fidelity of CHAMPS compared to more traditional CFD software.

Reference I

[1] AIAA.
1st AIAA Ice Prediction Workshop.
https://folk.ntnu.no/richahan/IPW/.

[2] Matthieu Parenteau, Simon Bourgault-Côté, Frédéric Plante, Engin Kayraklioglu, and Éric Laurendeau.
Development of parallel cfd applications with the chapel programming language.
January 2021.

[3] Pierre Trontin, Ghislain Blanchard, Alexandros Kontogiannis, and Philippe Villedieu.
Description and assessment of the new ONERA 2D icing suite IGLOO2D.
In *9th AIAA Atmospheric and Space Environments Conference*. AIAA Paper 2017-3417, June 2017.