High-Performance, Productive Programming using Chapel with Examples from the CFD Solver CHAMPS

Engin Kayraklioglu, Hewlett Packard Enterprise
Éric Laurendeau, Polytechnique Montréal
Karim Zayni, Polytechnique Montréal

Advanced Modeling & Simulation (AMS) Seminar Series
NASA Ames Research Center, February 20th, 2025

Today's Speakers

Engin Kayraklioglu
Hewlett Packard Enterprise
Principal Software Engineer

Éric Laurendeau
Polytechnique Montréal
Professor

Karim Zayni
Polytechnique Montréal
Ph.D. Student

What is Chapel?

Chapel: A Modern Parallel Programming Language

Imagine a programming language for parallel computing that is as...

…**readable and writeable** as Python

...yet also as...

…**fast** as Fortran / C / C++

…**scalable** as MPI / SHMEM

…**GPU-ready** as CUDA / HIP / OpenMP / Kokkos …

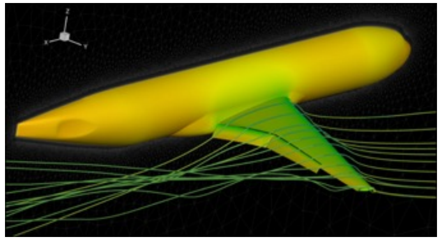
…**portable** as C

…**fun** as [your favorite programming language]

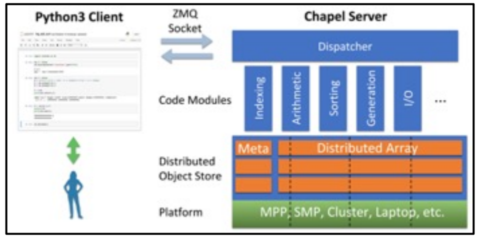
chapel-lang.org

This is our motivation for Chapel

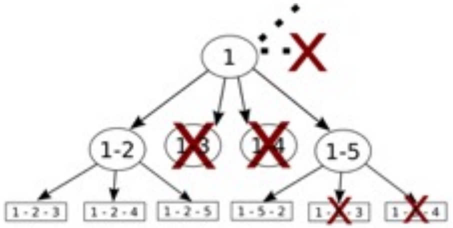
Applications of Chapel



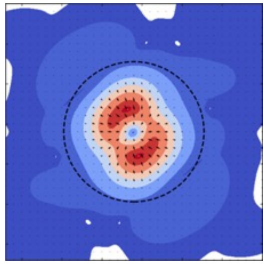
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



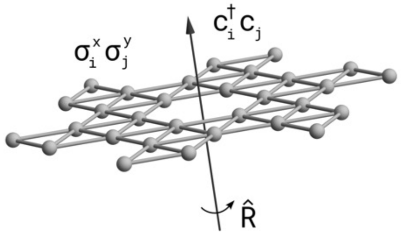
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



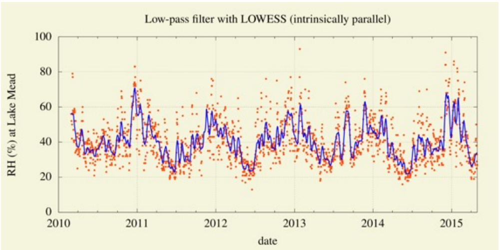
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

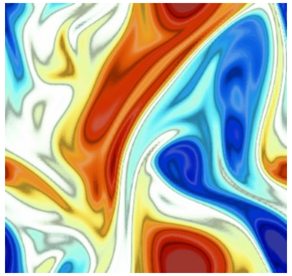


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

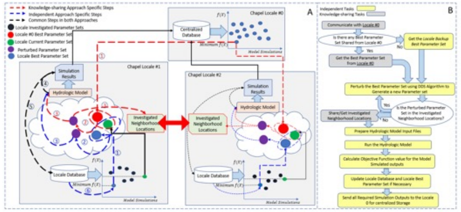


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

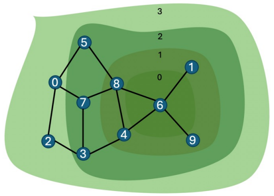
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



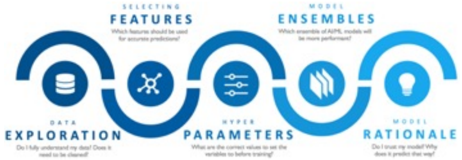
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



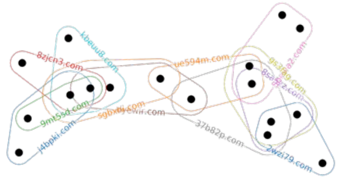
Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

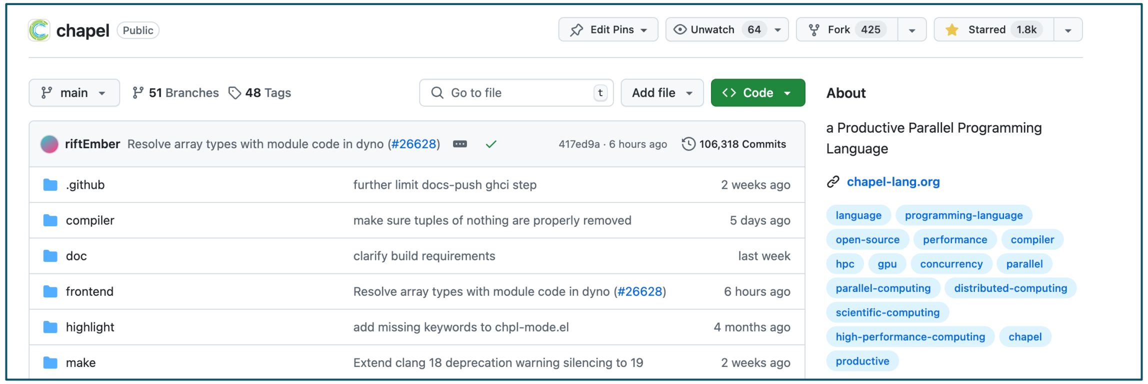


CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

[images provided by their respective teams and used with permission]

Chapel Community

Chapel is Open Source



github.com/chapel-lang/chapel

Community Events

- Office hours, live coding sessions, teaching meetups, language design discussions (new!)
- Annual conference ChapelCon

chapel-lang.org/community/

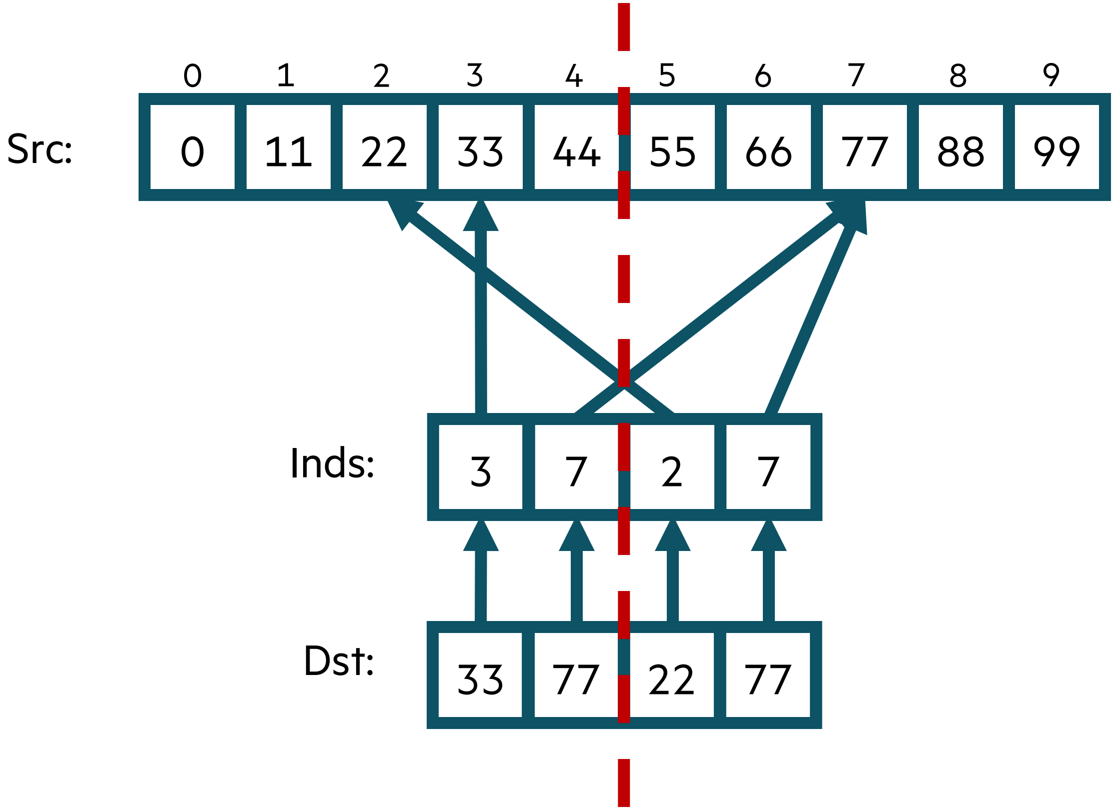
Chapel is joining the High Performance Software Foundation

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

THE LINUX FOUNDATION

What does Chapel code look like?

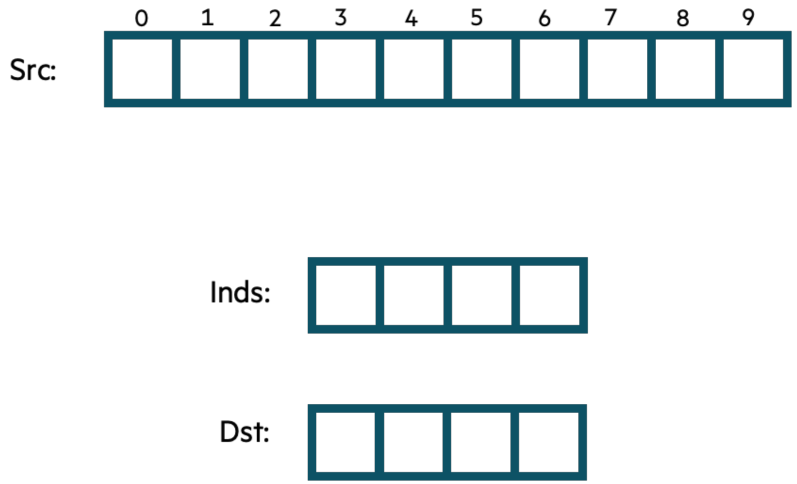
Bale IndexGather (IG): In Pictures



Bale IG in Chapel: Array Declarations

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

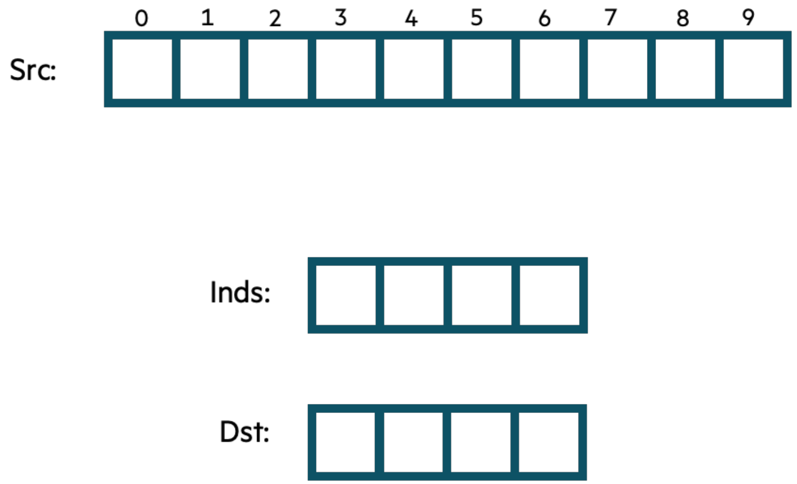
```
$
```



Bale IG in Chapel: Compiling

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

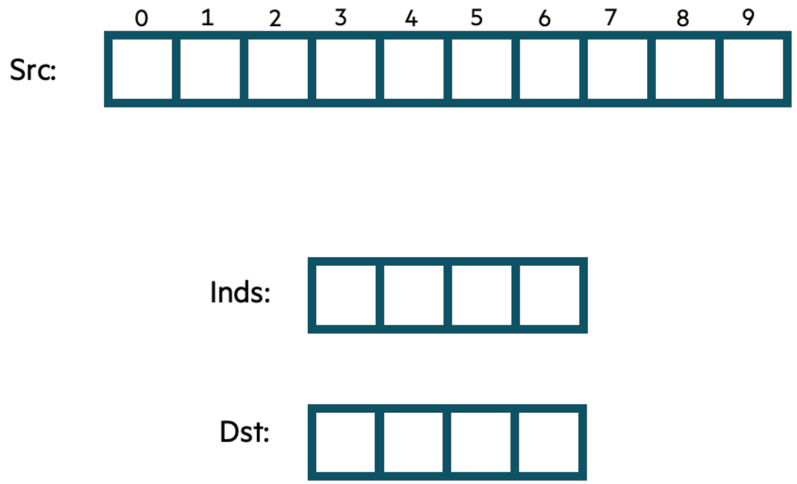
```
$ chpl bale-ig.chpl
$
```



Bale IG in Chapel: Executing

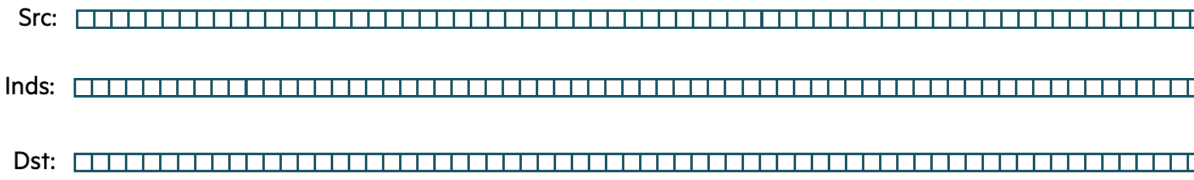
```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Executing, Overriding Configs

```
config const n = 10,
             m = 4;


var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

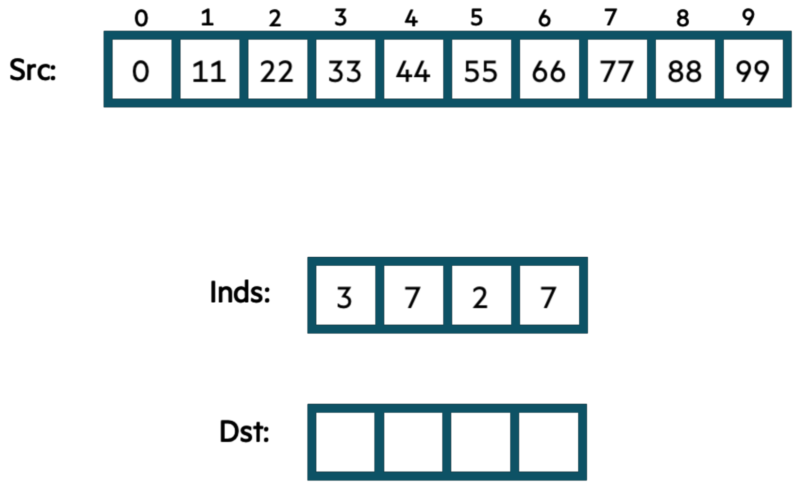



```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```

Bale IG in Chapel: Array Initialization

```
use Random;

config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;

Src = [i in 0..<n] i*11;
fillRandom(Inds, min=0, max=n-1);
```

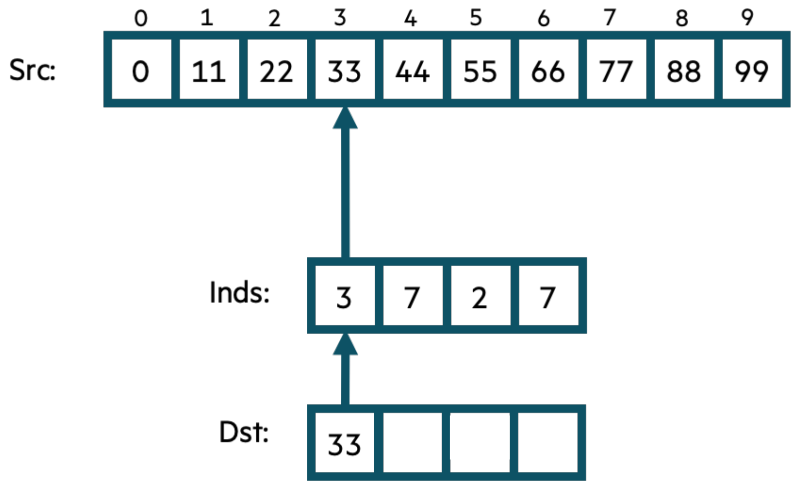
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Serial Version

```
config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
for i in 0..<m do
  Dst[i] = Src[Inds[i]];
```

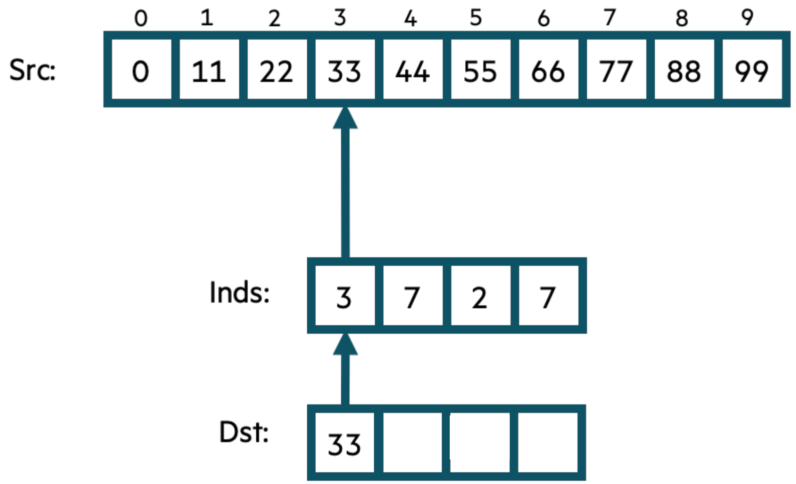
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Serial, Zippered Version

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
for (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

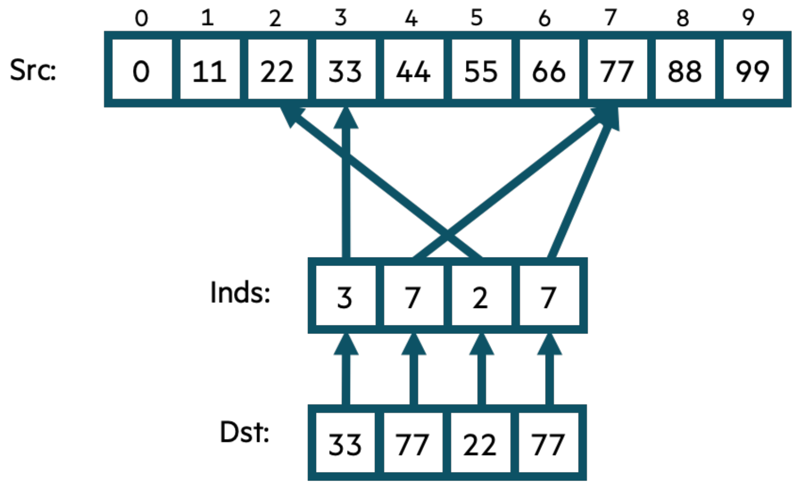
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel, Zippered Version (Vectorized)

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
foreach (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

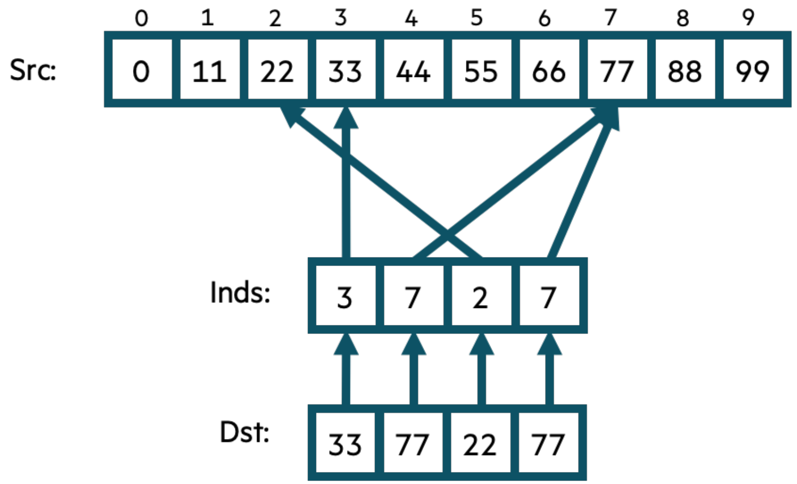
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



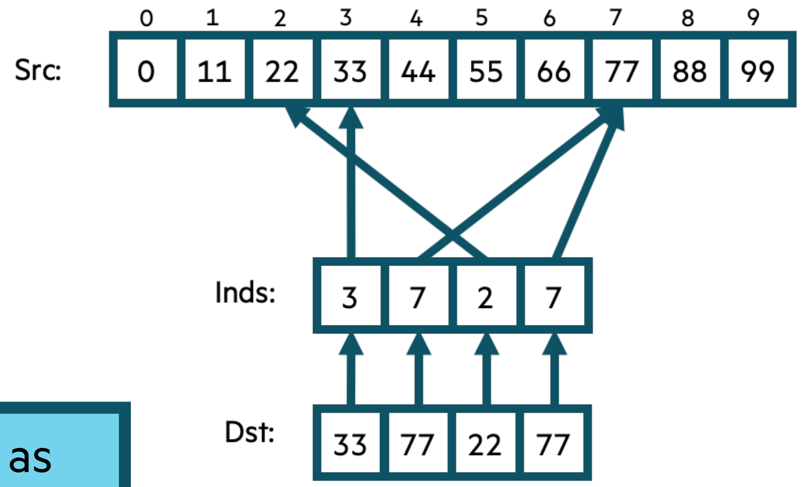
Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
config const n = 10,
             m = 4;

var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

Can also be written as

```
Dst = Src[Inds];
```

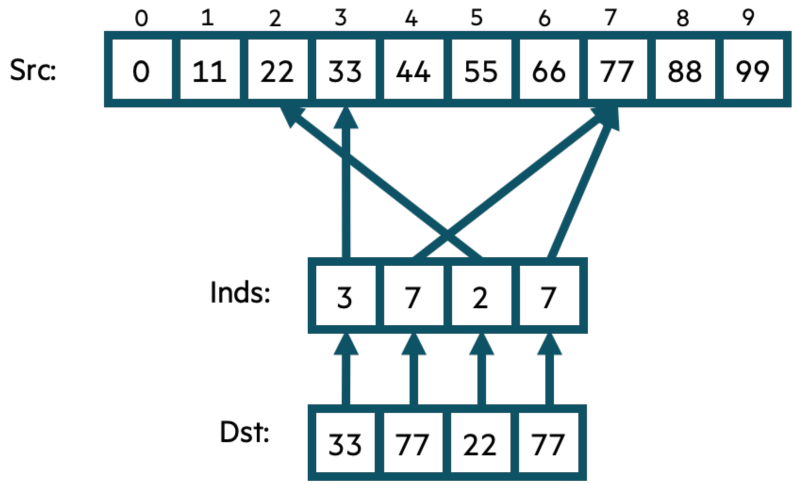
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

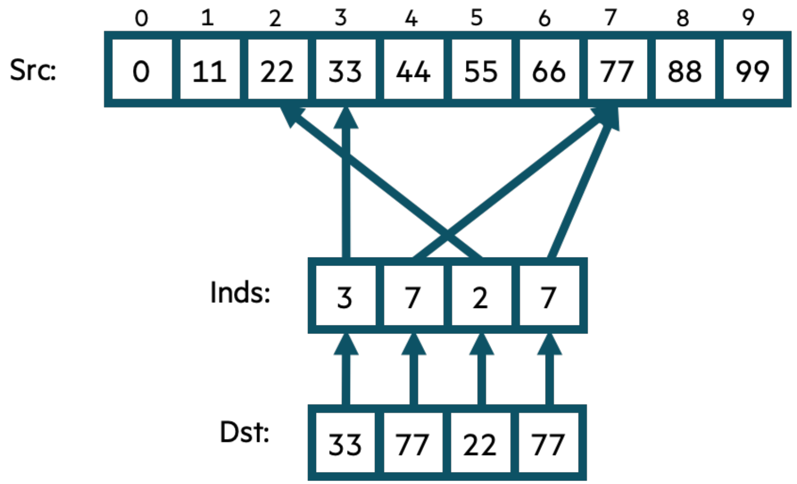
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel, Zippered Version for a GPU

```
config const n = 10,
             m = 4;



on here.gpus[0] {
  var Src: [0..<n] int,
      Inds, Dst: [0..<m] int;
  …
  forall (d, i) in zip(Dst, Inds) do
    d = Src[i];
}
```

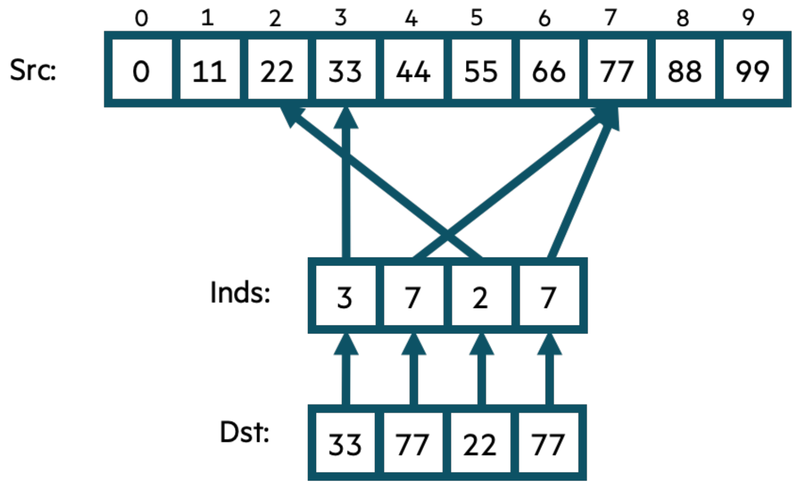
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel, Zippered Version (Multicore)

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

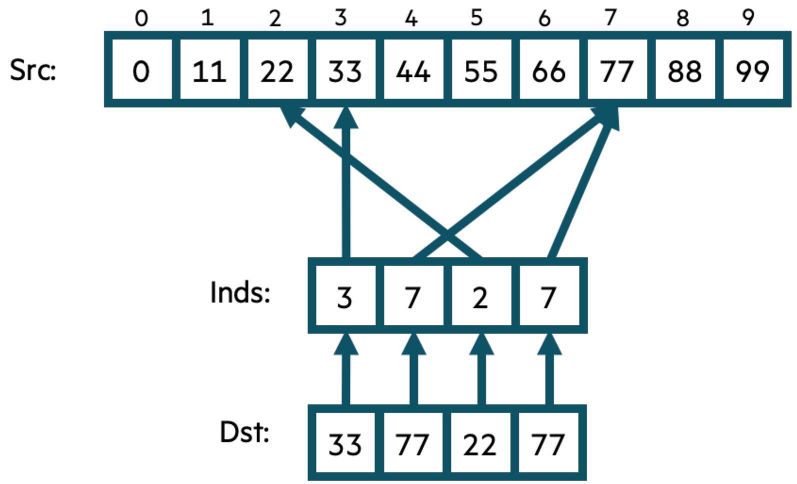
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Parallel , Zippered Version with Named Domains (Multicore)

```
config const n = 10,
             m = 4;

const SrcInds = {0..<n},
      DstInds = {0..<m};

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

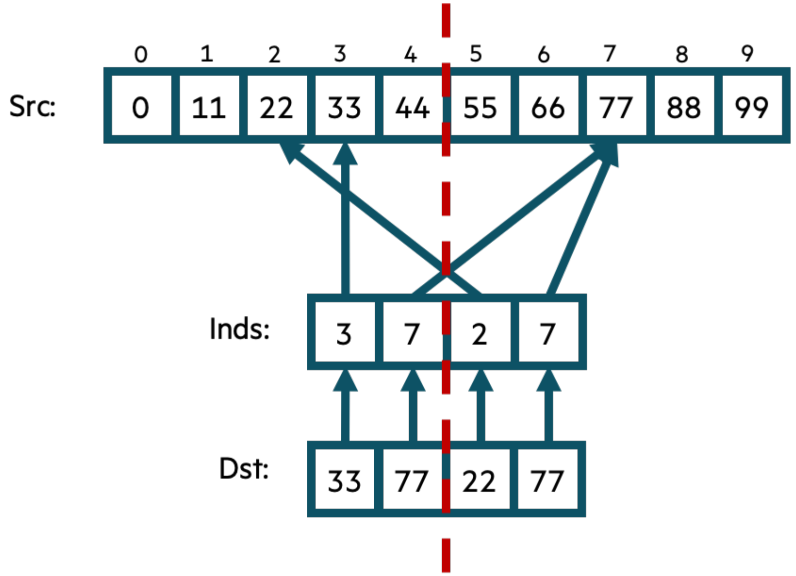
```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```



Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

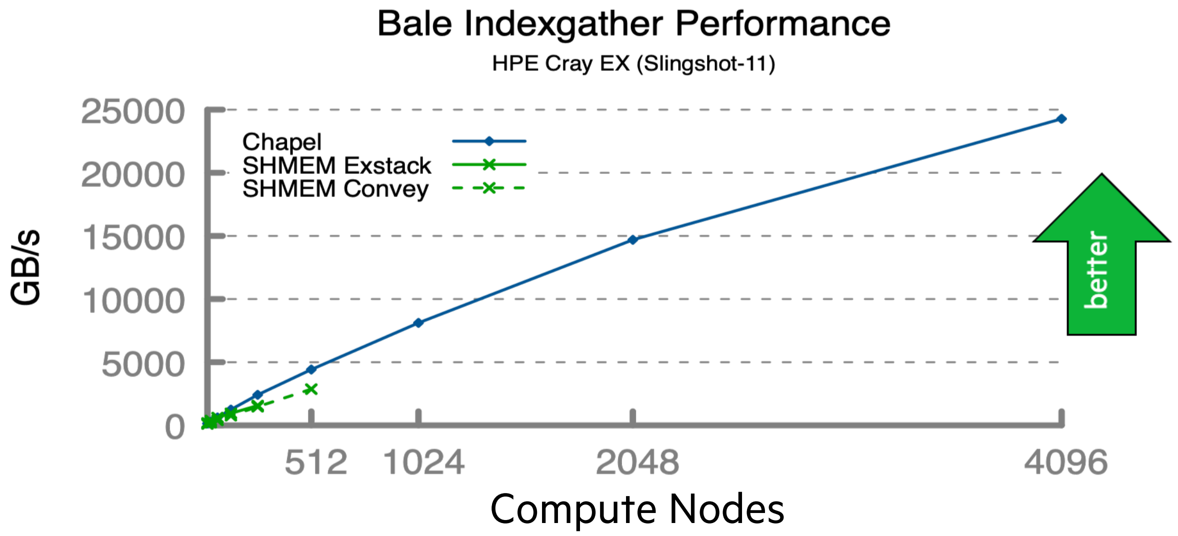
```
$ chpl bale-ig.chpl
$ ./bale-ig -nl 4096
$
```

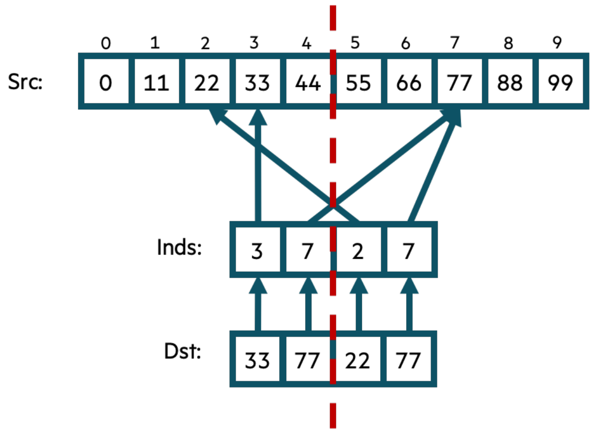


Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl --fast --auto-aggregation
$ ./bale-ig -nl 4096
$
```





CHAMPS: CHApel Multi-Physics Software

Computational Aerodynamics R&D efforts within Academia

Vision for transonic aerodynamic modeling

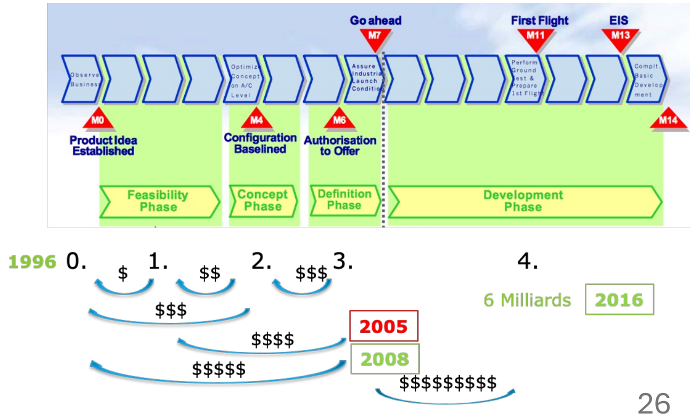
- CFD via RANS has matured and is fully integrated within industrial workflows
  - Including adjoint-based optimization
  - Full-flight envelope remains elusive (e.g. unsteady flows)
- CFD's integration within multidisciplinary applications has yet to reach maturity
  - Static-Aeroelasticity, including adjoint, has been thoroughly studied
  - More holistic problems can only be achieved with multi-fidelity approach

Table 1 MDO levels and tool sets[a]

MDO Level	Fidelity	Aerodynamics	Structures	Propulsion
CMDO	L0	Knowledge-based aerodynamics	Knowledge-based weight prediction	Fixed architecture, scaled engine model
	L1	Quasi-3D methods (3D VLM / Panel method + 2D High-Fidelity CFD)	Beam or thin-shell models	Variable architecture, generic rubber engine
	L1.5	Disciplinary L2 Surrogate Models		Surrogate model(s) from Engine supplier(s)
PMDO	L2	Mid-to-High Fidelity CFD (3D TSD to RANS)	Global FEM	Real engine model (fixed)
	L2.5	Disciplinary L3 Surrogate Models		
DMDO	L3	RANS	Detail FEM	

Source:
Piperni et al., Development of a Multilevel Multidisciplinary-Optimization Capability for an Industrial Environment, AIAA J, 2013.

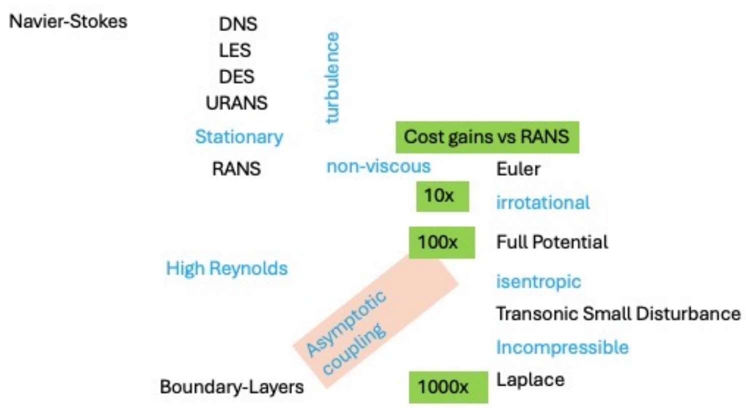
- Case study: C-Series/A220

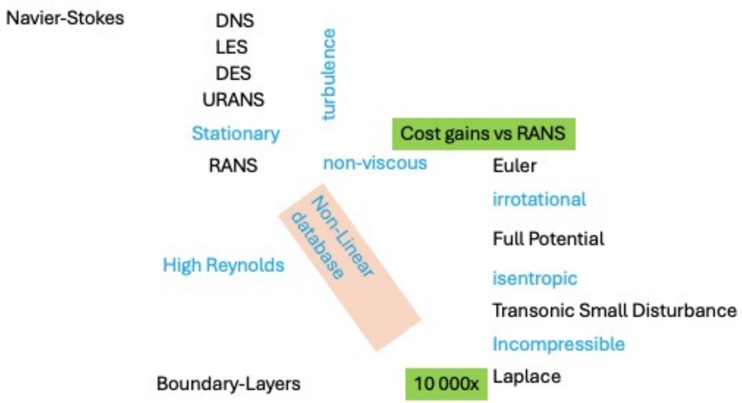


Sources:
Flaig, Axel. Airbus 380: Solutions to the aerodynamic Challenges of Designing the World's Largest Passenger Aircraft, 2008

Wikipedia for cost+years

Aerodynamic modeling choices





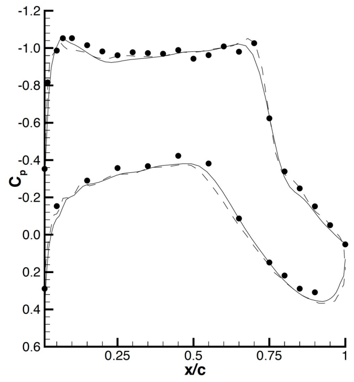
Boundary-Layer Coupling Schemes

- Captures shock-waves + flow separation

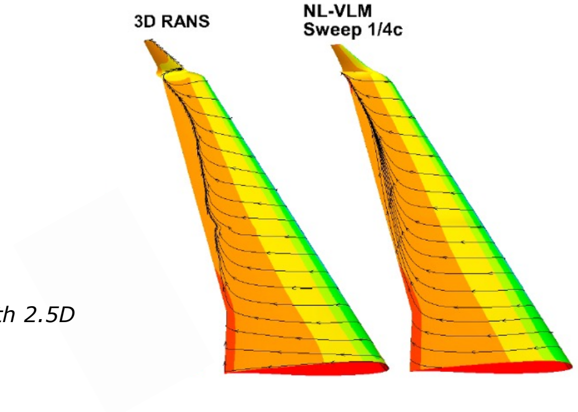
High Fidelity Database

- Allows easy AI/ML treatment

Boudreau, J. & Laurendeau, E., drag Prediction Using the Euler/Navier-Stokes Code FANSC, SAE 2003-01-3022



Parenteau et al., VLM Coupled with 2.5D RANS Sectional Data for High-Lift Design, AIAA 2018-1049



Academic Research Constraints

- Very High Turnaround of High-Quality Personnel (HQP)
  - Undergrad Summer Research (0.3 years), MSc (2 years), PhD (3-4 years), Post-Docs (2-years)

- Industrial Research Contracts and needs for 3D Full Aircraft Aerodynamic Modeling
  - Master complete workflow: geometry, mesh generation, flow solver,post-processing
  - Acquire multi-disciplinary and multi-fidelity knowledge

- High-Performance Computing 'barrier'
  - Complex Source Codes, despite great advances in computer sciences
  - Computational efficiency, a nice-to-have is now a must-have to perform analysis or optimization
  - OPEN-MP + MPI paradigms makes for O(Million) lines of code

Academic Laboratory Solutions

- Cascading complexity stream of problems
  - Fundamental to applied problems
  - MSc (2D, single disciplinary), PhD (3D, multidisciplinary), Post-doc (high TRL levels)

- Large laboratory
  - Flat governance structure
  - Collaboration with single-disciplinary specialists (e.g. optimization, AI, etc.)
  - International collaboration, great also for training HQP

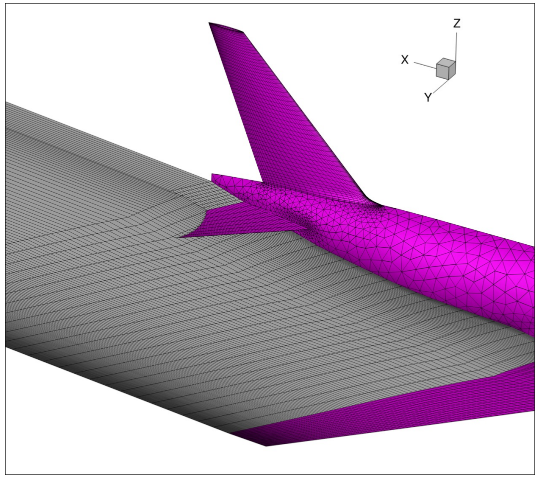
- High-Performance Computing
  - New unstructured RANS-based software using Chapel language: CHAMPS

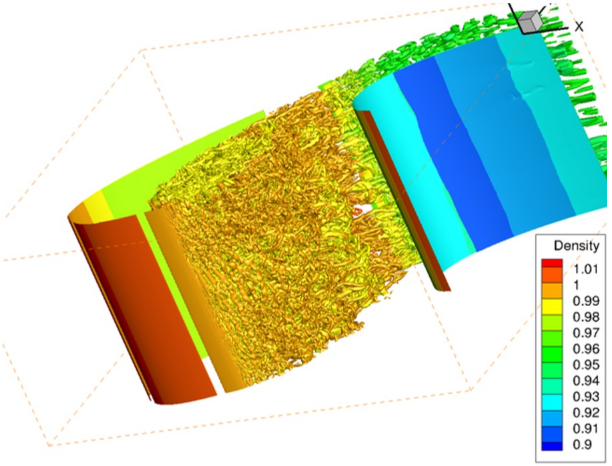
CHAMPS: Advanced 2D-3D CFD Solver

Overview of CHAMPS (Chapel Multi-Physics Software)

- Cutting-edge Computational Fluid Dynamics (CFD) solver
- 2D and 3D simulations using unstructured meshes.
- Three levels of fidelity for complex aerodynamic and multi-physics problems.
- Solves the Reynolds-Averaged Navier-Stokes (RANS) equations using second-order finite volume methods.
- Supports advanced convective flux schemes like Roe and AUSM.
- Includes Spalart-Allmaras (SA), k-ω SST-V, and Langtry-Menter transition models.
- Explicit Runge-Kutta solver, Implicit solvers including SGS and GMRES for enhanced stability.
- Linked with external libraries such as MKL, CGNS, METIS, MMG, CGAL and PETSc.
- Simulates icing phenomena using both deterministic and stochastic approaches.
- Handles fluid-structure interactions for advanced aerodynamics studies.

CHAMPS: Multi-Fidelity Transonic Viscous Flows





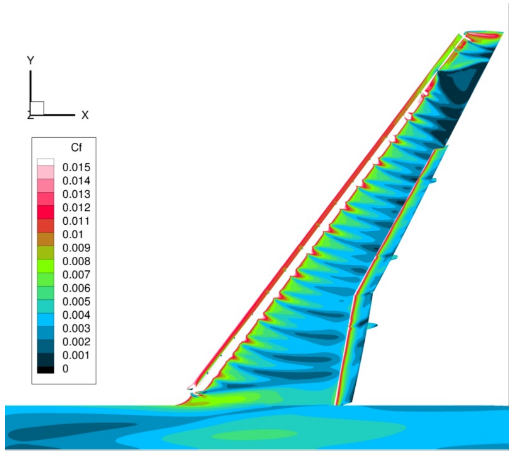
Medium Fidelity ~O(mins):

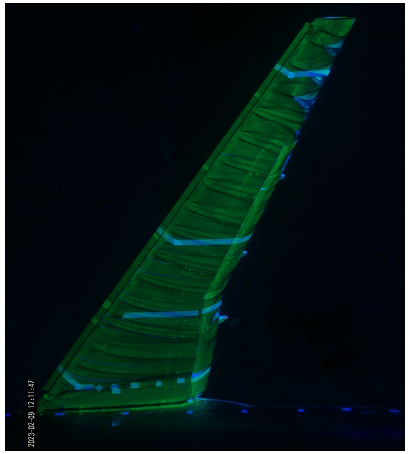
- Euler (Coupling with Boundary Layer)
- Non Linear Vortex Lattice Method (NL-VLM)

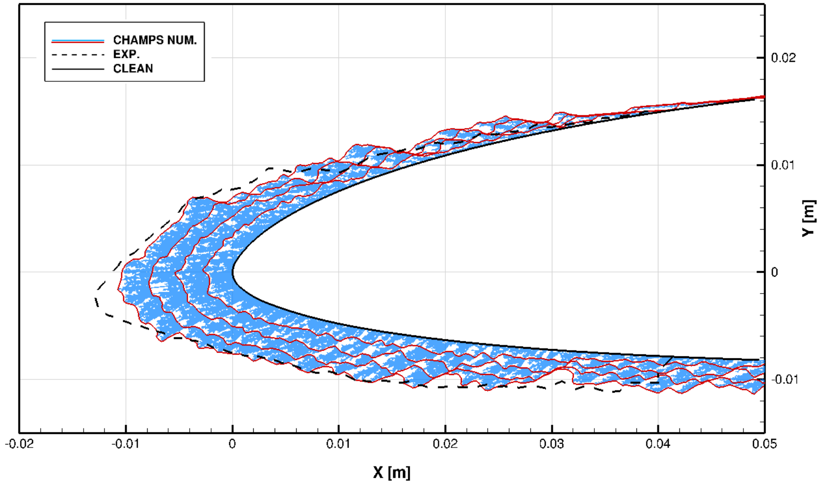
High Fidelity ~O(Hours/Days):

- Unsteady Reynolds Averaged Navier Stokes (U-RANS)
- Wall Model Large Eddy Simulation (WMLES)
- Detached Eddy Simulation (DES)

CHAMPS: Expertise & Global Impact







- Current Development team consists of 6 PhD and 3 MSc students
- Used at Université Strasbourg (France), Prof. Hoarau
- Used in Polytechnique Montréal's graduate class in Computational Aerodynamics
- CHAMPS has contributed to over 50 publications, including 10 journal articles
- Workshops Participation:
  - High Lift Prediction Workshops (HLPW): 4 and 5th Editions
  - Drag Prediction Workshops (DPW): 6, 7 and 8th Editions
  - Ice Prediction Workshops (IPW) : 1st and 2nd Editions

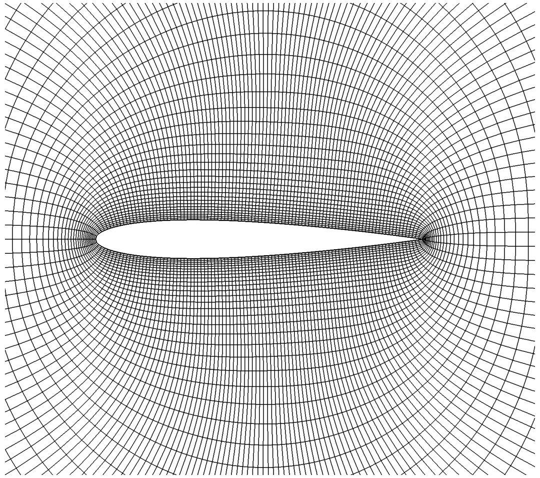
CHAMPS: Advanced 2D-3D CFD Solver

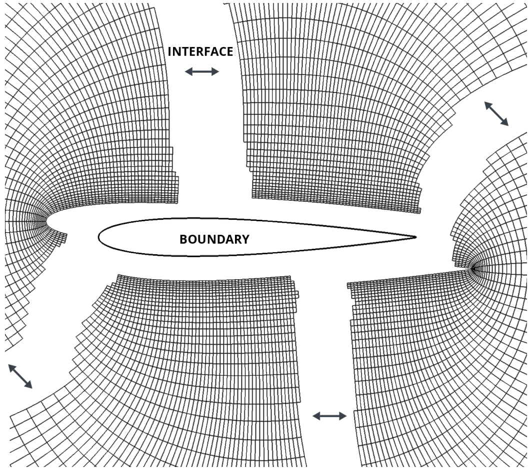
Codebase Statistics:

- **CHAMPS** : ~150K lines of code
- **Pre-Processor**: ~17K lines
- **Flow Solver**: ~15K lines
- **Turbulence Solver**: ~13K lines
- **Droplet Solver (Eulerian + Lagrangian)**: ~24K lines
- **Post-Processor**: ~2K lines
- **Smaller Solvers**: ~5K lines (each)
- **Shared Structure & APIs** : ~50K lines

Parallel CFD for HPC

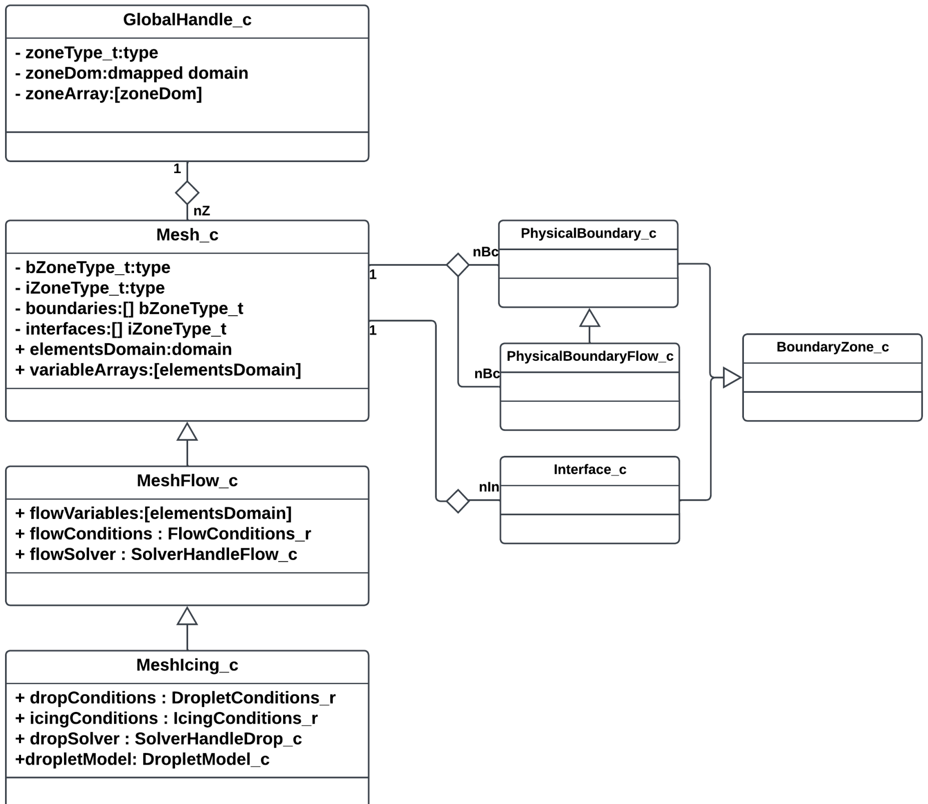
- Volumetric Meshing around complex geometries
- 2D Meshes : Ranging from 0.5 to 1.0 Million Unknowns
- 3D Meshes : Handling up to 1 Billion Unknowns
- Leveraging HPC to significantly reduce computation time
- Problem is partitioned into smaller sub-problems interconnected via interfaces
- Each sub-problem runs independently on dedicated tasks
- Minimizing communication overhead to maximize overall efficiency





Software Structure :
Framework

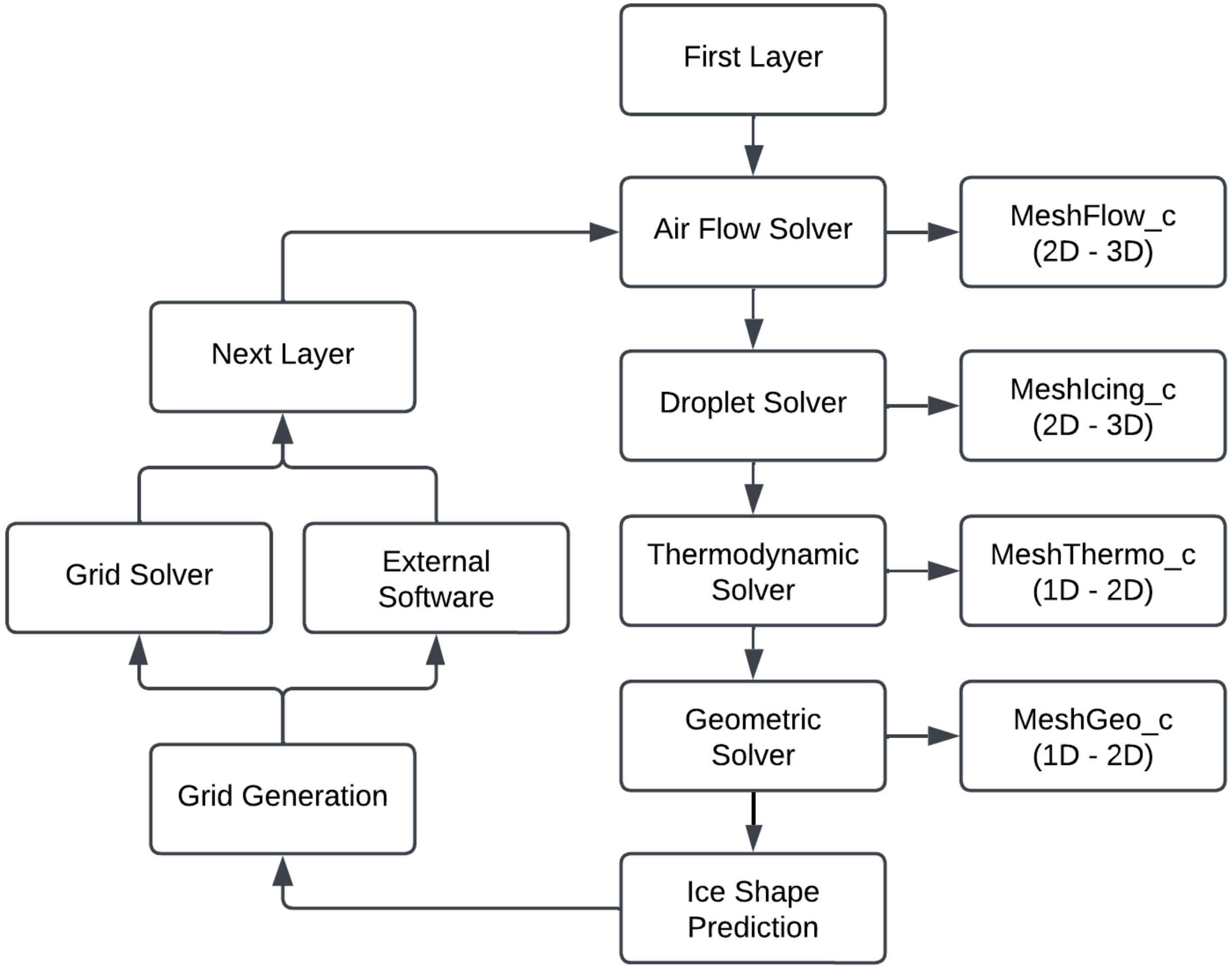
- Multiphysics problems require different computational domains grid (*Mesh_c*)
- *Type* aliases in *Chapel* are used to define computational domains at the start of each simulations
- Supports Generic Programming and Improve flexibility
- Distributed *domains* enable efficient handling of large-scale simulations across multiple computational nodes




```
/** Type of mesh objects **/
    type zoneType_t;

    /** Range associated with the distribution of the Locales. **/
    const localeSpace_ = {0..#numLocales};

    /** Distributed domain on the Locales **/
    const localeDomain_ : domain(1) dmapped blockDist(boundingBox=localeSpace_) = localeSpace_;

    /** Number of zones in the grid file **/
    const numZones_ : int = 0;

    /** Range associated with the grid domain as bounding box **/
    const zoneSpace_ = {0..#numZones_};

    /** Distributed domain for the grid **/
    const zoneDomain_ : domain(1) dmapped  blockDist(boundingBox=zoneSpace_) = zoneSpace_;

    /** Distributed array of mesh objects **/
    var zones_ : [zoneDomain_] zoneType_t = [zoneDomain_] new zoneType_t();
```

Software Structure :
Icing Framework

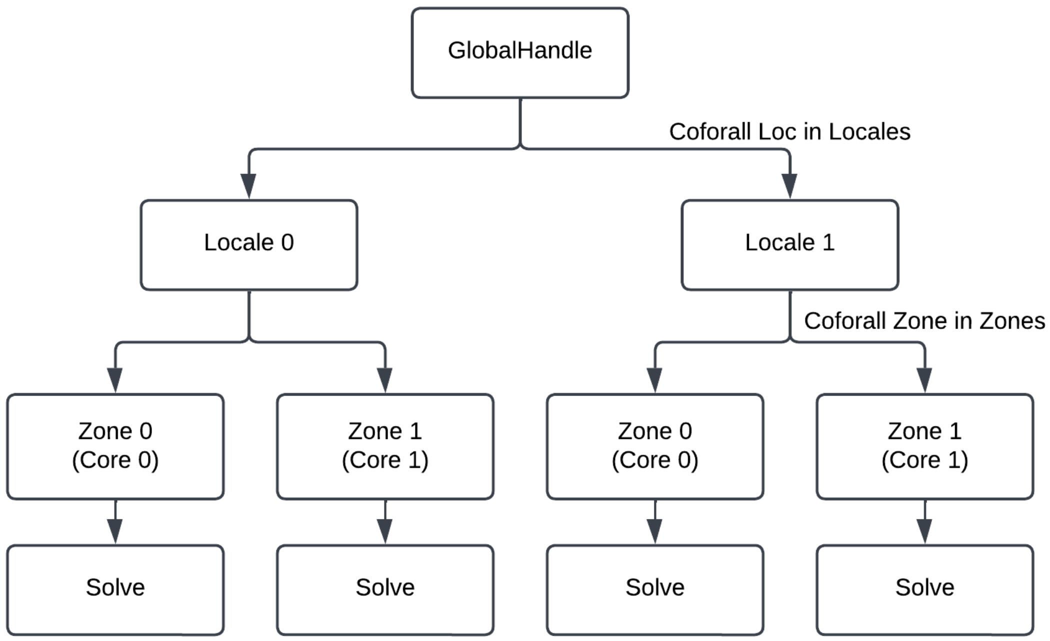
- Typical icing simulations involve four distinct computational domains
- Some domains solve the volume field (3D), while others focus on surface interactions (2D)
- Each computational domain has its own specific characteristics, variables and requirements



Software Structure :

Top-level Overview

- Each simulation runs in a single execution (using a *GlobalHandle*), ensuring efficiency and consistency.
- Tasks are distributed hierarchically using *coforall* statements, first at the Locale (node) level, then further subdivided at the Zone (core) level to maximize parallelism.
- Grid partitioning through METIS guarantees optimal load balancing across all cores, enhancing computational efficiency and minimizing idle time.



```
proc main(args : [] string)
{
    // Fetching user inputs for the flow
    var flowInputs  : FlowInputs_r;
    var turbInputs  : TurbInputs_r;

    ref commonInputs = flowInputs.commonInputs_;

    var globalHandle : GlobalHandleFlow_t = initializeComputationalDomain(GlobalHandleFlow_t, commonInputs);

    try! writeln("Elapsed time after reading grid: ", clock.elapsed());

    runFlowSimulation(globalHandle.borrow(), clock, flowInputs, turbInputs);
}
```

- Type *GlobalHandleFlow_t* will initialize *MeshFlow_t* computational domains
- One proc to run : runFlowSimulation()

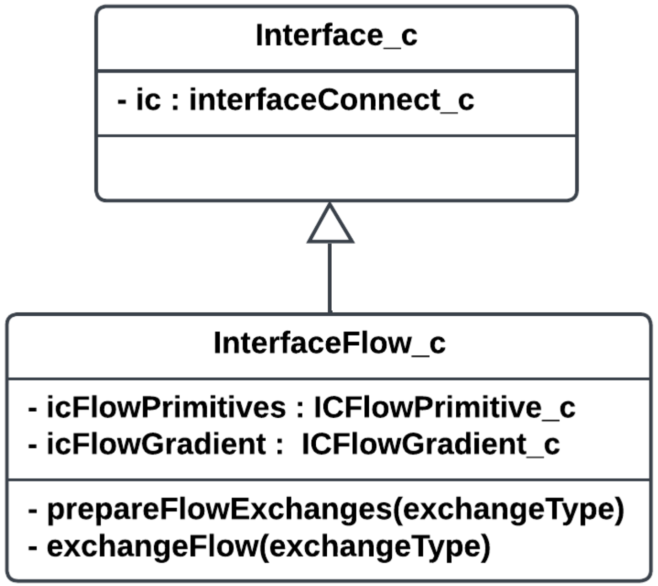
```
coforall loc in Locales do
on loc
{
    const localZonesIndices = globalHandle.zones_.localSubdomain();
    ref   localZones        = globalHandle.zones_.localSlice(localZonesIndices);

    coforall zone in localZones.borrow()
    {
        zone.flowModel_.solve(zone, locID, zone.localZoneIndex_);
    }
}
```

- First *coforall* will loop over Locales
- Second *coforall* will loop over cores to execute Solve()
- Provides greater control of each computational domain's operations

Software Structure :
Data Exchanges

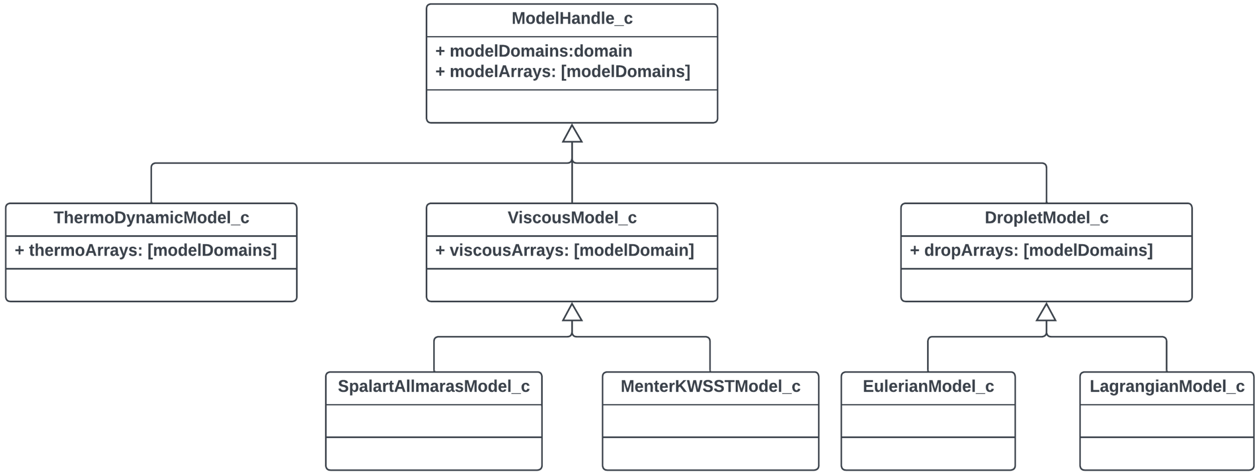
- Each *interface* is equipped with an *interfaceConnect* to facilitate seamless communication with adjacent zones.
- Each zone prepares for data exchanges by populating its respective buffer arrays, ensuring that all necessary information is readily available for transfer.
- Custom synchronization barriers are implemented to maximize efficiency.
- Once synchronization is achieved, all data exchanges are executed simultaneously, minimizing communication overhead and maximizing throughput.
- The global namespace support provided by *Chapel* ensures that any task can access the necessary buffers, regardless of its location across Locales.

```
proc performInterfaceExchanges(zone, exchangeType : ExchangeType_t)
    {
        use CustomAllLocalesBarriers;

        // Fill buffers
        zone.prepareExchange(exchangeType);

        customAllLocalesBarrier.barrier();

        // Read buffers
        zone.exchangeInterfaces(exchangeType);

        customAllLocalesBarrier.barrier();
    }
```



Software Structure :
Generics & Modularity

- All models inherit from a base *ModelHandle_c* Class
- Maximize code reusability, leading to faster implementation and enhance readability
- *Where* statements are needed to prevent compilation errors.
- This ensures compatibility when fields or methods are not present in the parent class.
- *Where* statements also prevent conflicts with sibling classes (other children of the same parent).



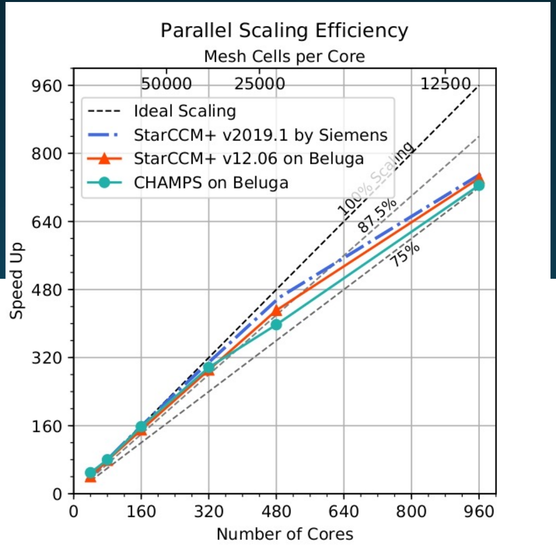

```
override proc solve(zone, locId : int, localTaskID : int)
where isProperSubtype(zone.type, MeshThermo_c)
```

Software Structure :
Model Implementation

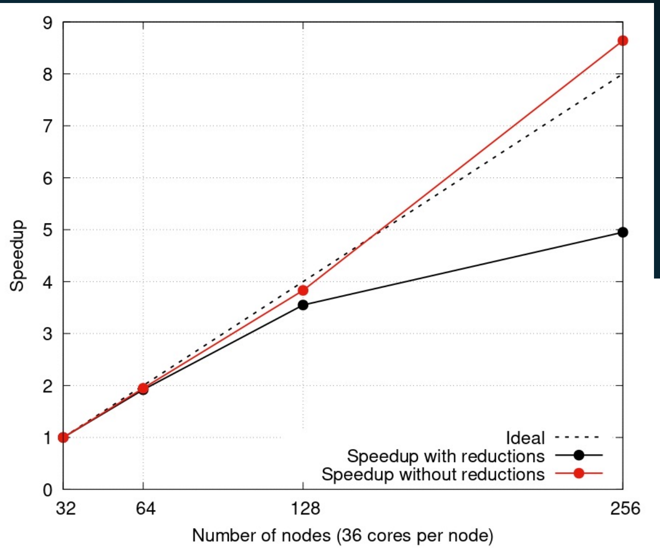
- Viscous models are decided based on user input at the start of runFlowSimulation()
- Leads to the instantiation of a new *viscousModel_c* object in each zone

```
proc initializeViscousModelAndSolver(globalHandle, zone, flowInputs : FlowInputs_r, turbInputs : TurbInputs_r, t
{
    select flowInputs.FLOW_REGIME_
    {
        when FlowRegime_t.INVISCID
        {
            zone.viscousModel_ = new owned InviscidModel_c();
        }
        when FlowRegime_t.LAMINAR
        {
            zone.viscousModel_ = new owned LaminarModel_c();
        }
        when FlowRegime_t.TURBULENT
        {
            select turbInputs.TURB_MODEL_
            {
                when TurbulenceModel_t.SA
                {
                    zone.viscousModel_ = new owned SpalartAllmarasModel_c(zone, turbInputs);
                }
                when TurbulenceModel_t.KW
                {
                    zone.viscousModel_ = new owned MenterKWSSTModel_c(zone, turbInputs);
                }
            }

            zone.viscousModel_.initializeConditionsAndSolvers(globalHandle, zone);
        }
    }
}
```

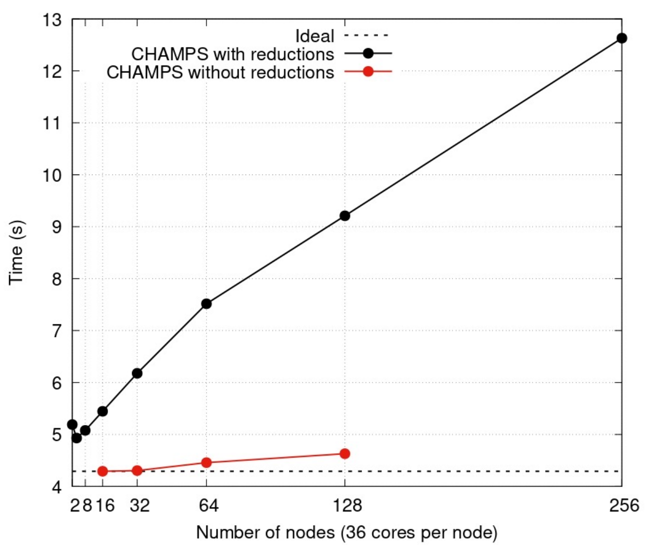
Scalability Analysis



Civil Airliner : Strong Scaling



Simple Box : Strong Scaling

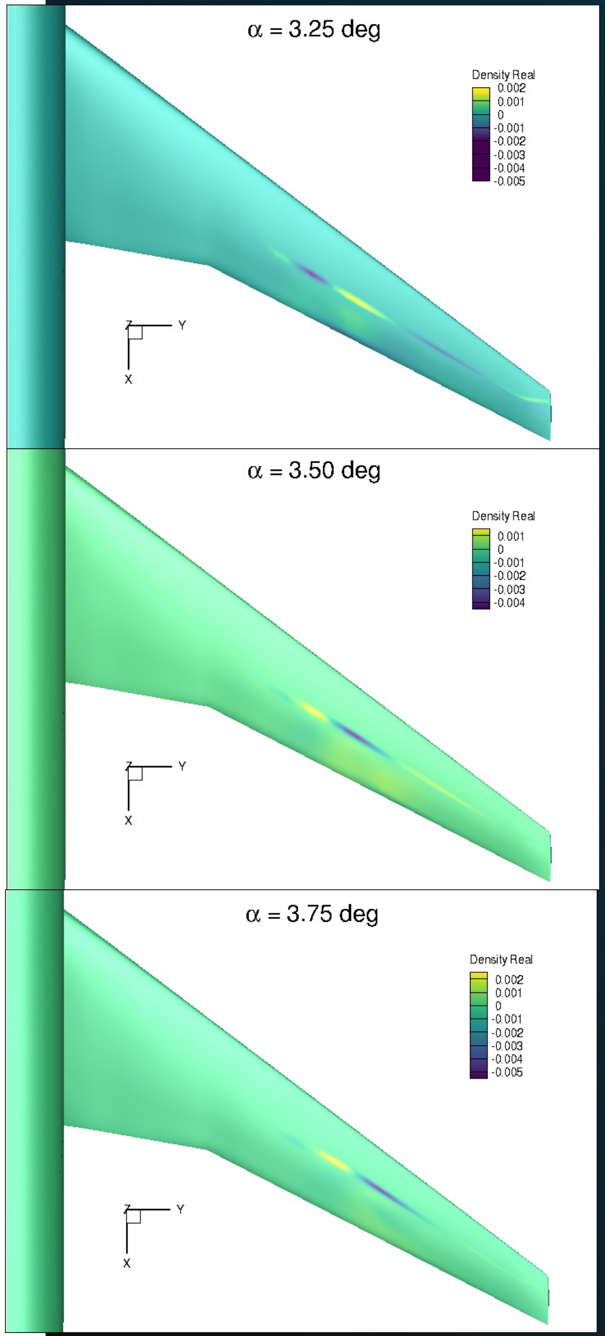


Simple Box : Weak Scaling

- **Civil Airlines Model :**
  - CHAMPS' performance were similar to other softwares available industrially
- **Plain Box Model:**
  - Conducted on an **HPE HPC cluster** this test used a high number of cores (9216 cores in total) to explore CHAMPS's scalability.
  - Tests with and without **reduction operations** revealed **super scalability** in the absence of reductions.

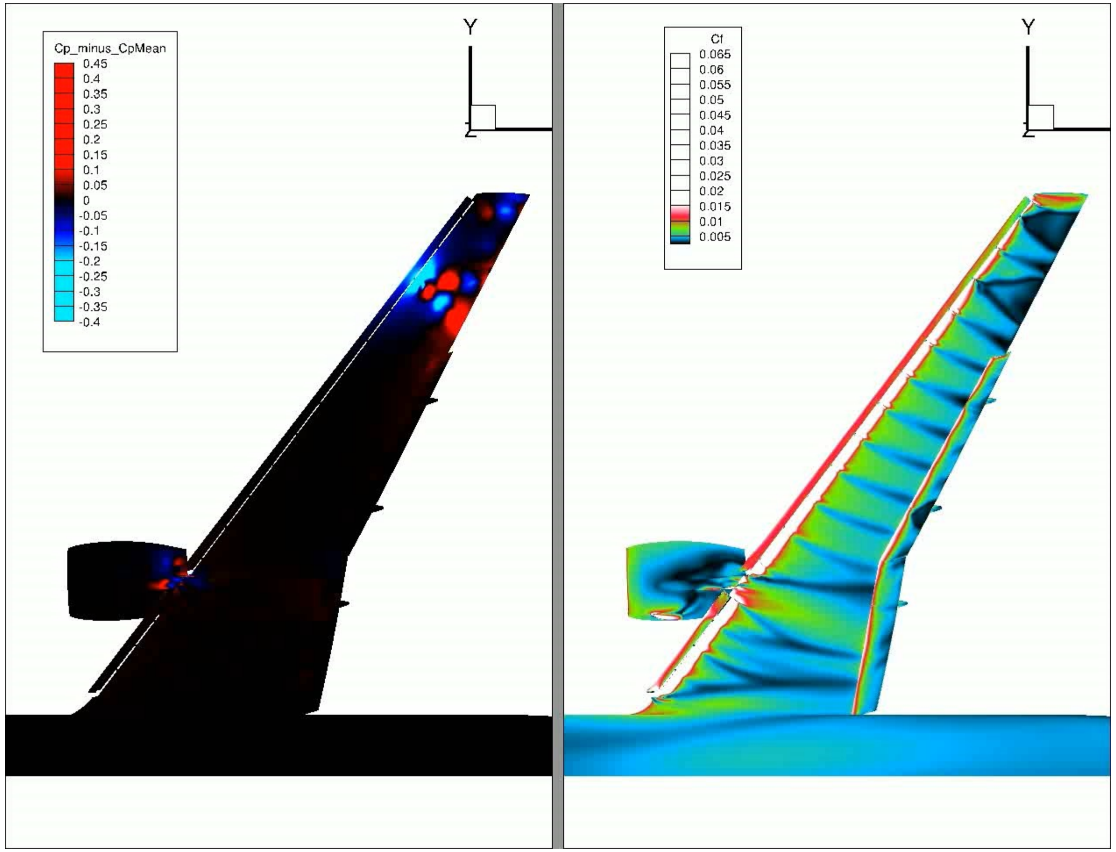
CHAMPS :
Recent Results & Capabilities

- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



CHAMPS :
Recent Results & Capabilities

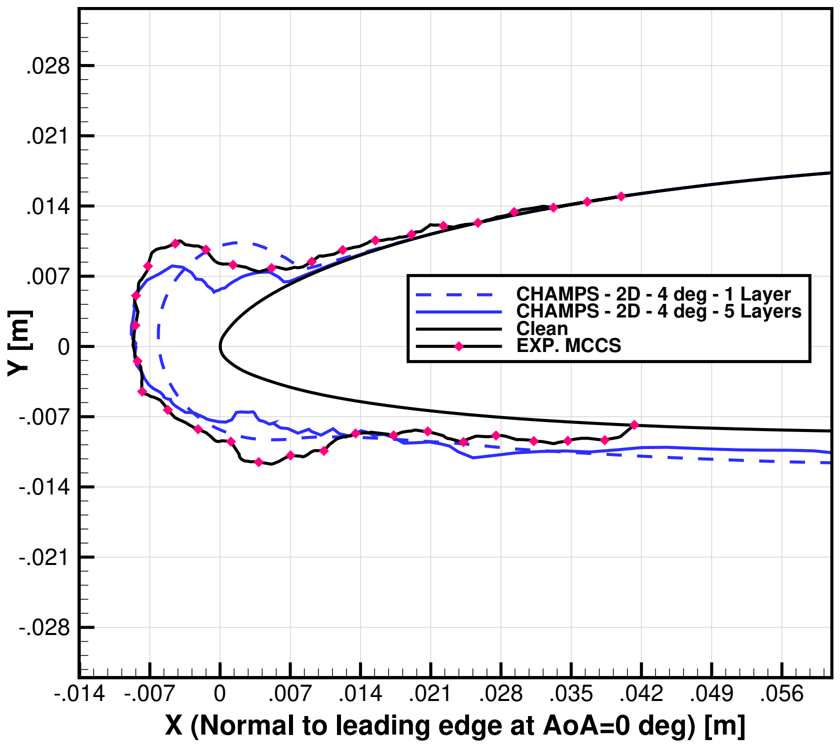
- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



CHAMPS :

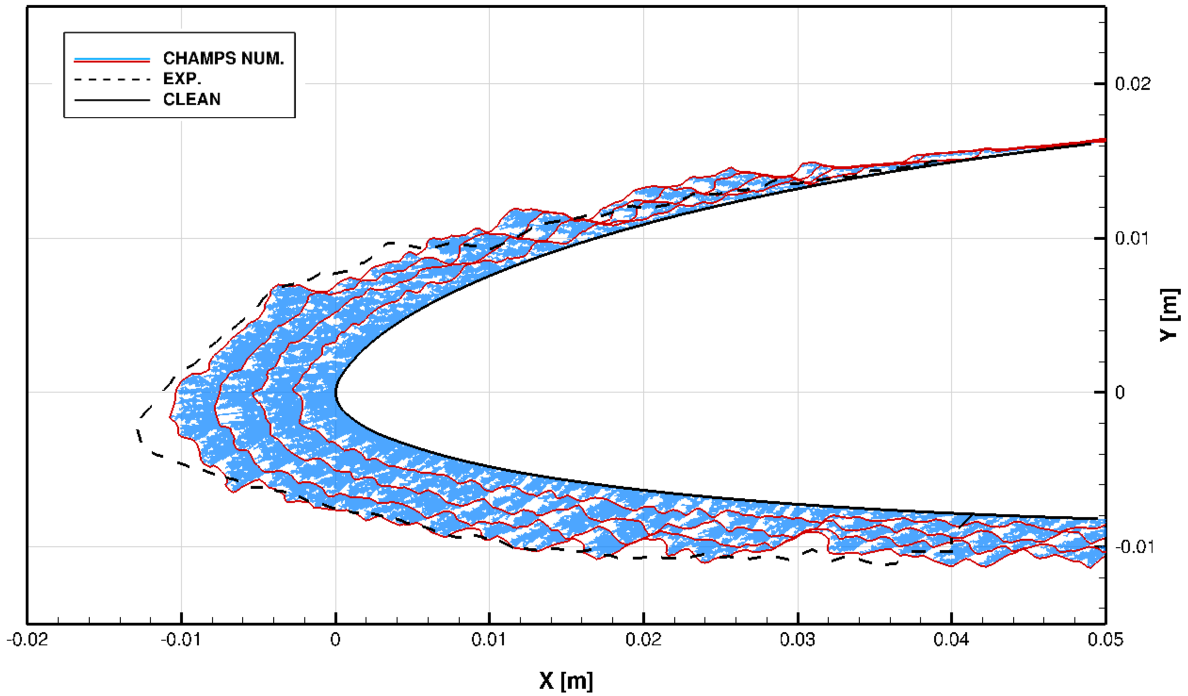
Recent Results & Capabilities

- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



CHAMPS :
Recent Results & Capabilities

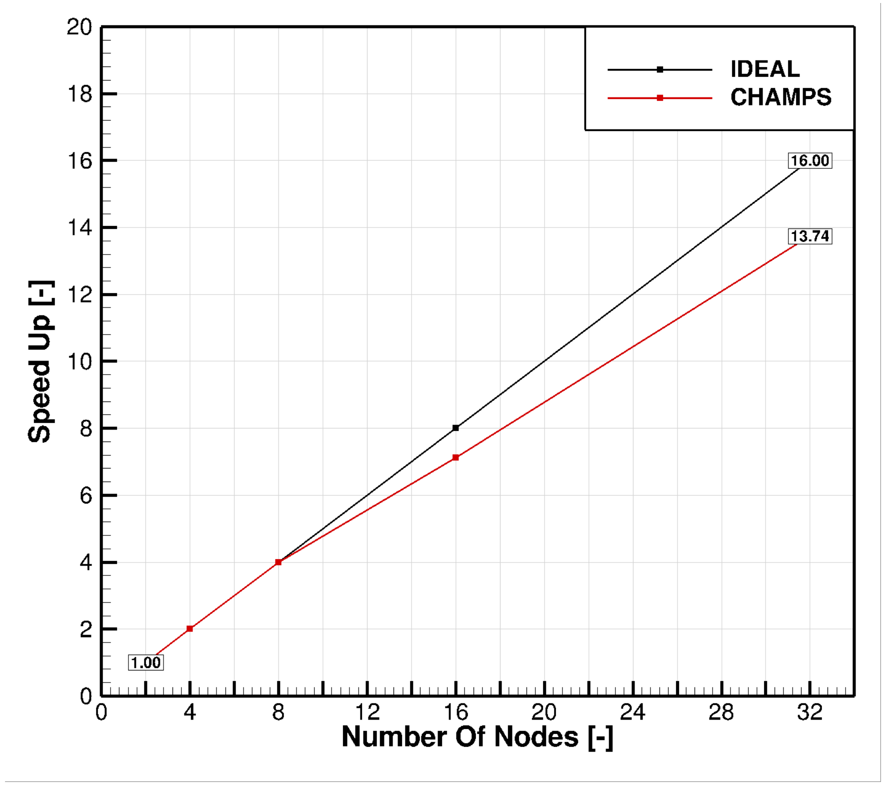
- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



CHAMPS :

Recent Results & Capabilities

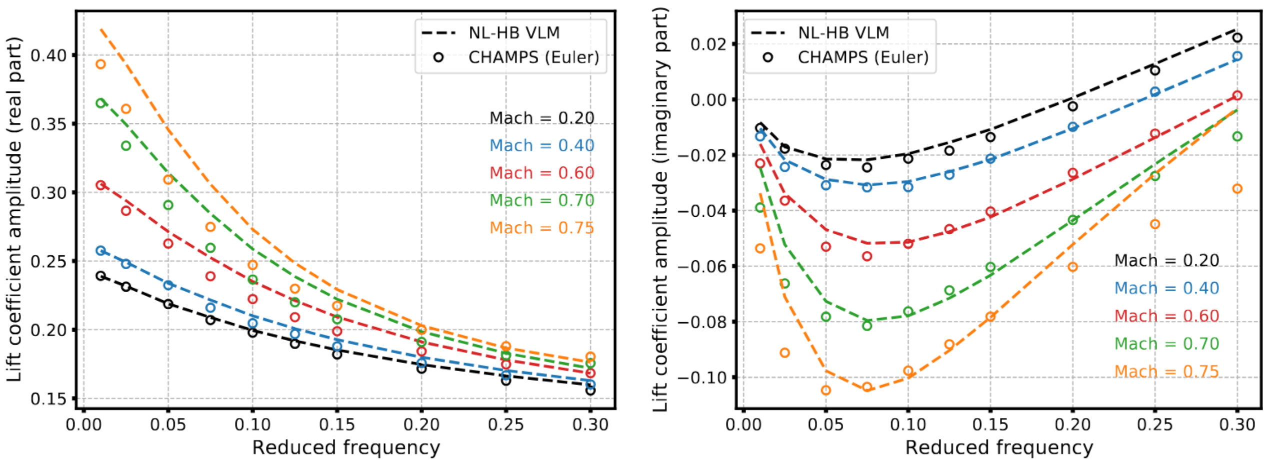
- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



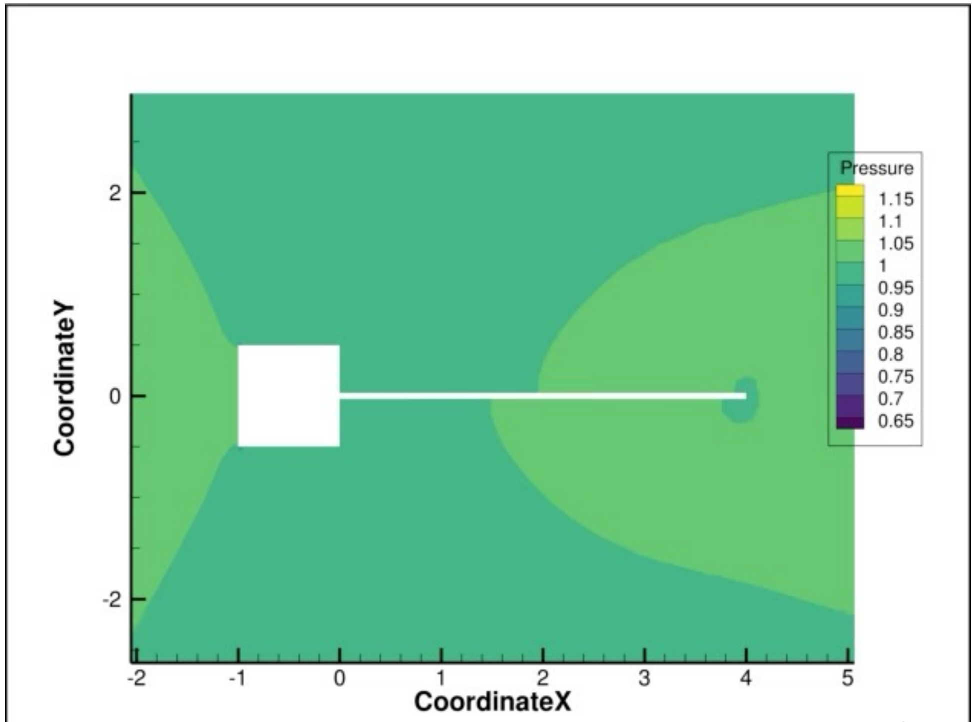
1 Node = 40 Cores
Total = 1280 Cores

CHAMPS :
Recent Results & Capabilities

- Global Stability Analysis
- Studying High lift configurations (HLPW5)
- Predicting Ice shapes in 2D and 3D (IPW2)
- Multilayer Stochastic Ice accretion
- Lagrangian Model Scalability
- Aero-Elasticity



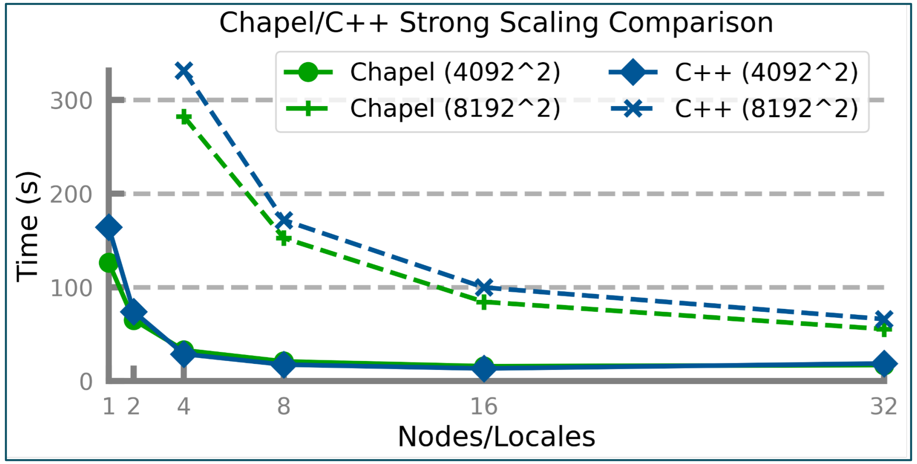
(a) Real part (b) Imaginary part

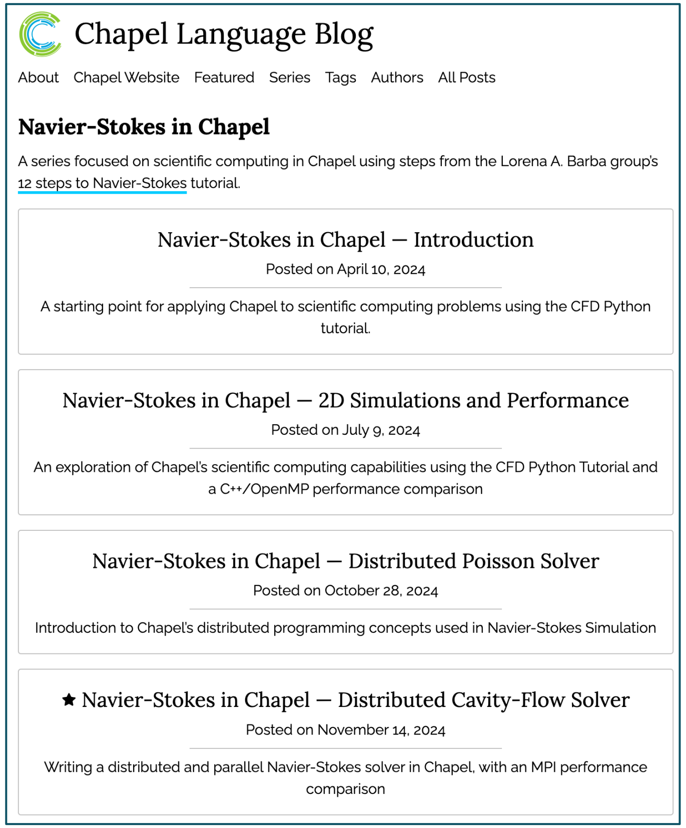


More on Chapel...

Navier-Stokes in Chapel

- Four introductory articles using Navier-Stokes as use-case
  - Based on an existing Python example
  - Chapel concepts are gradually introduced
    – **with side-by-side comparisons to Python**
  - Basics of Chapel
  - Single-node parallelism
  - Introduction to distributed programming concepts
  - Ending with **scalability and performance comparison with C++ / MPI**




Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

Navier-Stokes in Chapel

A series focused on scientific computing in Chapel using steps from the Lorena A. Barba group's 12 steps to Navier-Stokes tutorial.

Navier-Stokes in Chapel — Introduction
Posted on April 10, 2024
A starting point for applying Chapel to scientific computing problems using the CFD Python tutorial.

Navier-Stokes in Chapel — 2D Simulations and Performance
Posted on July 9, 2024
An exploration of Chapel's scientific computing capabilities using the CFD Python Tutorial and a C++/OpenMP performance comparison

Navier-Stokes in Chapel — Distributed Poisson Solver
Posted on October 28, 2024
Introduction to Chapel's distributed programming concepts used in Navier-Stokes Simulation

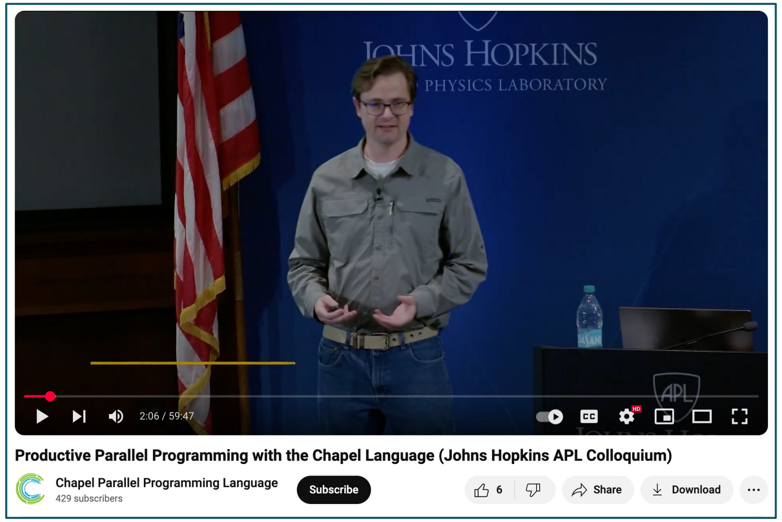
★ Navier-Stokes in Chapel — Distributed Cavity-Flow Solver
Posted on November 14, 2024
Writing a distributed and parallel Navier-Stokes solver in Chapel, with an MPI performance comparison

chapel-lang.org/blog/series/navier-stokes-in-chapel/

A Pair of Previous Talks

- Michael Ferguson (HPE) gave a talk at NASA Goddard
  - *Productive Parallel Programming with the Chapel Language*
  - A lot of performance comparisons to other languages
  - At-scale performance results using sorting
  - **This talk may be available internally to you, as well**

- A similar version from a Johns Hopkins University Applied Physics Lab Colloquium is available



www.youtube.com/watch?v=SuZckfFF_pE

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

7 Questions for Chapel Users

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Highly recommend Eric's interview on Chapel blog

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

Using Chapel in satellite image analysis for coral reef biodiversity analysis

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

Tags: User Experiences | Interviews | Data Analysis | Computational Fluid Dynamics

By: Engin Kayraklioglu, Brad Chamberlain

Using Chapel in data analytics for atmospheric turbulence research in the Amazon

Other success stories on graph processing and data analytics:

chapel-lang.org/blog/series/7-questions-for-chapel-users/

… stay tuned for more!

Ways to Engage with the Chapel Community

Live/Virtual Events

- ChapelCon (formerly CHIUW), annually
- Office Hours, monthly
- Live Demo Sessions, monthly

Electronic Broadcasts

- Chapel Blog, ~biweekly
- Community Newsletter, quarterly
- Announcement Emails, around big events

Community / User Forums

- Discord — Discord
- Discourse — Discourse
- Email Contact Alias — `chapel+qs@discoursemail.com`
- GitHub Issues
- Gitter — GITTER
- Reddit — reddit
- Stack Overflow — stackoverflow

Social Media

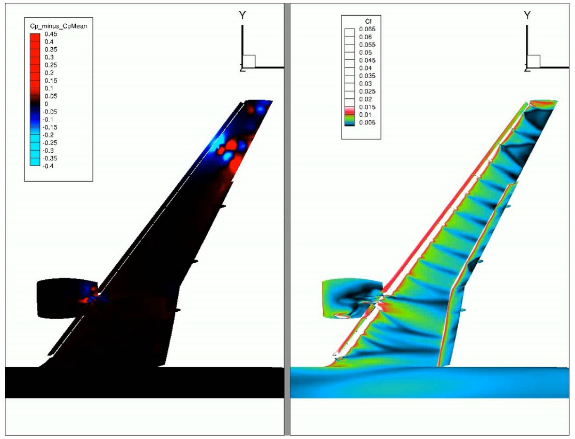
- Bluesky
- Facebook
- LinkedIn — LinkedIn
- Mastodon — mastodon
- X / Twitter
- YouTube — YouTube

Closing Thoughts

- Chapel is
  - productive,
  - parallel,
  - fast,
  - scalable,
  - open-source,
  - *flight-proven* ☺

chapel-lang.org

- Powered by Chapel, CHAMPS
  - is being developed very rapidly to increase its capabilities
  - can run on multiple nodes efficiently
  - produces high-fidelity results



Both teams are excited to hear comments, questions, and collaboration opportunities!

Thank you