Development of Parallel CFD Applications on Distributed Memory with Chapel

Matthieu Parenteau, Simon Bourgault-Cote, Frederic Plante and Éric Laurendeau

CRIAQ
CONSORTIUM FOR RESEARCH AND INNOVATION IN AEROSPACE IN QUÉBEC
NSERC CRSNG
POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY

Table of contents

Introduction

- Matthieu Parenteau: PhD Candidate, Member of Eric Laurendeau's research team
- Simon Bourgault-Cote: Research associate, Member of Eric Laurendeau's research team
- Frederic Plante: PhD Candidate, Member of Eric Laurendeau's research team
- Eric Laurendeau: Professor, Department of Mechanical Engineering

Main Research Activities

- Computational Fluid Dynamics (CFD)
- Multi-fidelity aerodynamics
- Multi-physics simulation (Aero-icing and Aero-elastic)

CFD

Complex Algorithms

- The Navier-Stokes equations are highly nonlinear PDEs
- Iterative solution of large sparse matrices

High Computational Cost

- The computational cost increases rapidly with fidelity (up to 600M elements for finest grids)
- The number of iterations increases with the fidelity

Programming Language

- C, C++ and Fortran are generally the main programming languages used in CFD to achieved adequate performances
- MPI/OpenMP are used to enable parallelism over distributed and shared memory respectively

What we want in a programming language for CFD

Productivity

- The research field of CFD evolves rapidly and is competitive
- Quick implementation of complex algorithms over distributed memory

Fast

- The inherent computational cost demands fast CFD software

Portable and Scalable

- 2D cases on a desktop
- Large scale 3D cases over 500+ cores
- 1 code portable to any hardware

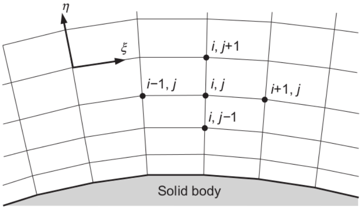
CFD Applications with Chapel

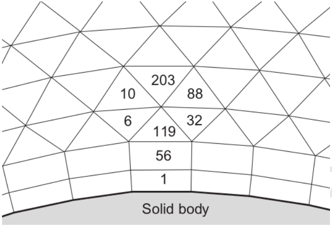
2D Euler Structured Code

As a first investigation, our in-house shared memory 2D Euler code written in C was converted into Chapel to compare directly the language performances.

3D RANS Unstructured Code

A new 3D unstructured Reynolds Average Navier-Stokes (RANS) flow solver was built from scratch with Chapel and with the aim to perform large scale simulations (3D) on distributed memory.





2D Structured Code

C code : NSCODE

- Shared memory only
- Parallelism performed with OpenMP on the partitioned mesh

Chapel code

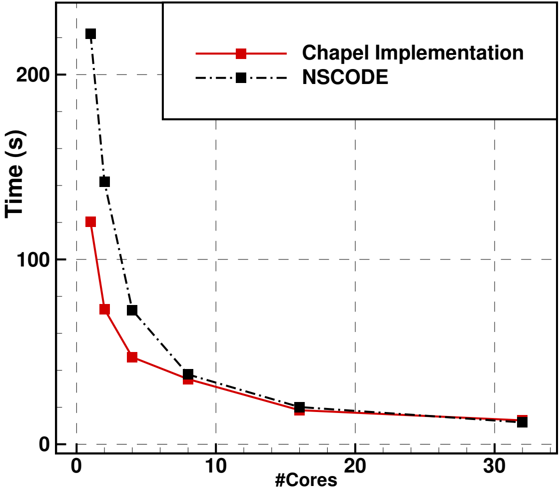
- Simple translation of the C code
- Compiled for single Locale only
- Parallelism applied like the C code with "forall"

Objective

For a simple flow solver, is Chapel as fast as C?

NSCODE vs Chapel Implementation

Real time in second to complete 100 iterations on a grid of 1M elements
→ Chapel code is faster or equal to NSCODE



3D Unstructured Code

Objective

Build a complete 3D unstructured RANS flow solver from scratch with Chapel for large scale simulation on distributed memory and compare overall performances with a traditional flow solver written in C++ (SU2).

CHapel Multi-Physics Simulation (CHAMPS)

- 3D Unstructured RANS flow solver.
- Second order finite volume.
- Convective flux schemes: Roe and AUSM
- Spalart Allmaras turbulence model.
- Explicit solver (Runge Kutta) and implicit solvers (SGS, GMRES, BCGSTAB).
- Interface with external libraries: MKL, CGNS, METIS and PETSC.

Parallelism over distributed memory

- Inspired by an SPMD approach with MPI
- 1 task per zone
- Efficient approach for finite volume schemes

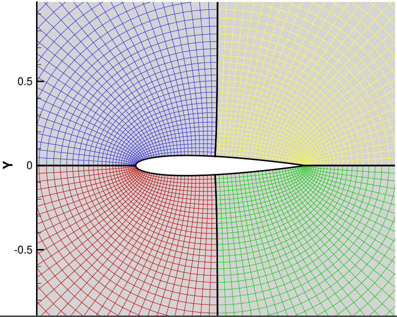
```
coforall loc in Locales do
on loc
{
    const localZonesIndices=globalHandle.zones_.
    localSubdomain();
    var localZones=globalHandle.zones_.localSlice(
    localZonesIndices);

    coforall (zone, localTaskID) in zip(localZones, 0..#
    localZones.size)
    {
        for i in 0..#maxIter
        {
            zone.flowSolver_.iterate(zone);
```

Communication

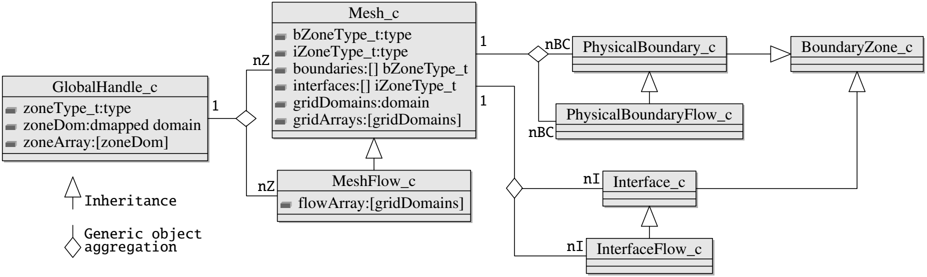
```
proc performInterfaceExchanges(zone, exchangeType)
{
    // Fill buffers
    local do zone.prepareExchange(exchangeType);
    allLocalesBarrier.barrier();

    // Read buffers
    zone.exchangeInterfaces(exchangeType);
    allLocalesBarrier.barrier();
}
```



CHAMPS

- A Multi-physics problem requires different computational grids
- Type aliases are used to define these various computational domains



Avoiding Performance Pitfalls

Implicit Parallelism

- Implicit parallelism embedded in some operations (whole array assignment)
- In CHAMPS, all the available cores on a Locale are used (1 task per zone)
- If placed inside the iterative process, implicit parallelism incurs serious overhead

Multi-Locale Overhead

- Noticeable overhead between Single-Locale and Multi-Locale mode
- The *local* statement is necessary to reduce this overhead

Overall Performances

Computational Time per Iteration

Depends on the harware, the programming language and the implementation.

Scalability

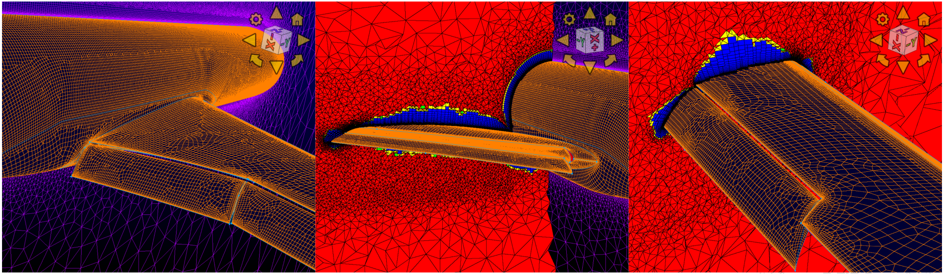
Depends on the harware, the programming language and the implementation.

Convergence Rate

Depends mainly on the implemented algorithms, flow conditions and mesh quality.

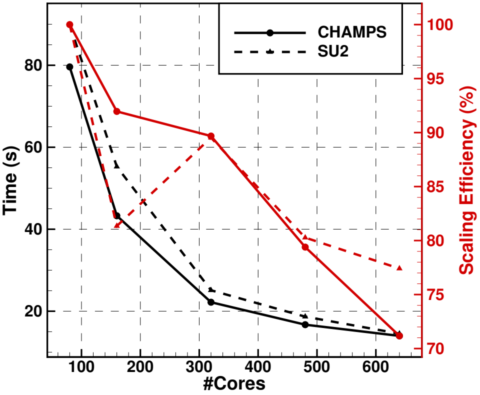
CRM - High-Lift Configuration

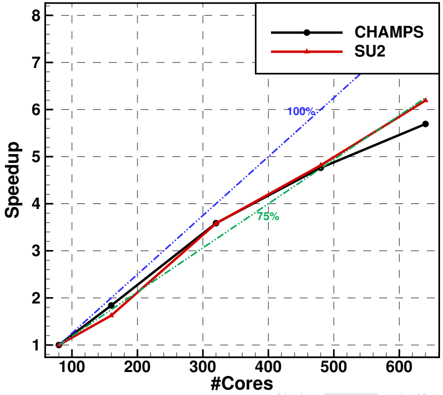
- Common Research Model (CRM) in high-lift configuration
- Mixed element grid: 22M cells and 10M nodes (coarse grid)
- Challenging conditions for a CFD software
- Chapel 1.19 configured with the infiniband conduit for gasnet
- Comparison against SU2 (C++/MPI)



Computational Time and Scalability

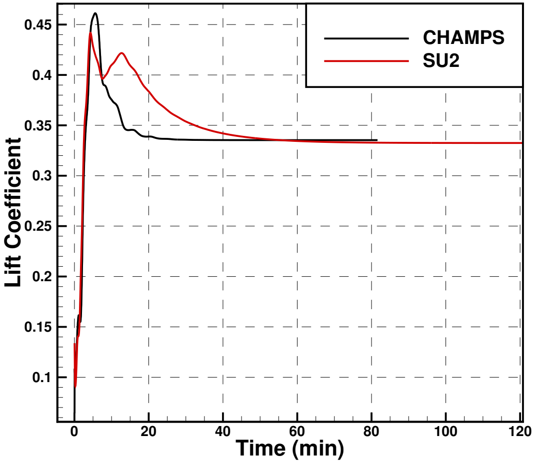
- CHAMPS is achieving similar performances than SU2.





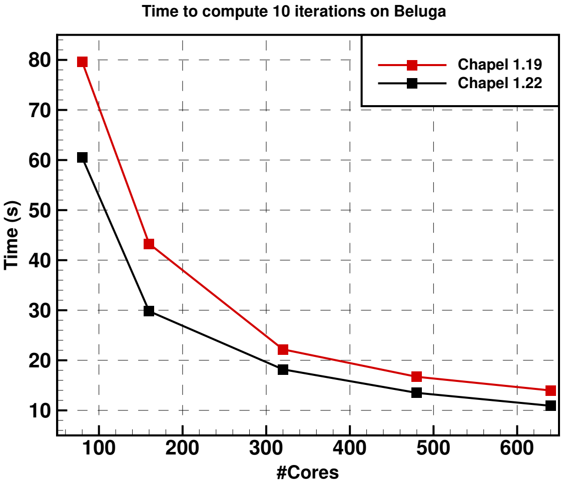
Convergence Rate

- Comparison of the lift force convergence for the clean configuration
- Flow conditions: Mach = 0.85, Angle of Attack = 1.0 and Reynolds number = 5E6



Chapel 1.22

- Optimization made for infiniband network
- Around 20% faster runs with Chapel 1.22



Conclusion

Distributed Memory Parallelism

- The development of distributed memory application is very efficient with Chapel
- Complex algorithms are easily portable to large computer clusters
- However, it must be done with care to avoid performance pitfalls

Productivity

- Language well accepted by the rest of the team (considering a C/C++/Python background)
- Additional modules are easily added by team members (other than original developers)

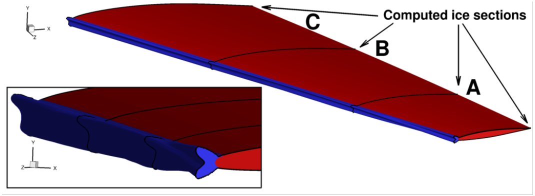
Performance

- The Chapel codes are performing similarly to other C/C++ applications.

Future Developments

New physical models

- Droplet model for ice accretion
- Structural model for aeroelastic simulations



Acknowledgements

- The authors would like to acknowledge the great support from the Chapel team at Cray.