Parallel programming languages for micrometeorological research: the case for Chapel.

Nelson Luís Dias

Department of Environmental Engineering, Federal University of Paraná, Brazil
: nelsonluisdias@gmail.com : www.nldias.github.io

XIV Brazilian Micrometeorology Workshop, 21-24 Oct Nov 2025

Acknowledgements

This topic is the result of collaboration with many people: Livia Freire Grion, Anna Jesus Felix, Willian Lesinhovski, Paulo Henrique Laba, Marcelo Chamecki, Greg Torkelson, Henrique Ferro Duarte, …

The Chapel development team, led by Brad Chamberlain, is gratefully acknowledged for answering many questions on my part about the language, as well as giving permission to slides in this presentation.

Thanks

for the invitation:

It is an honor.

Contents

Introduction

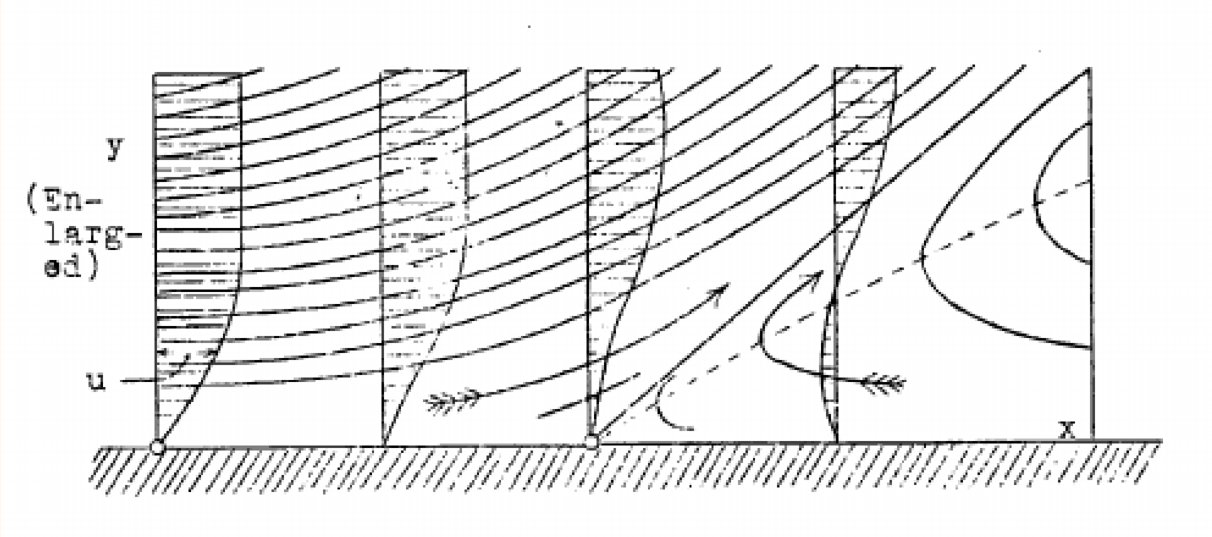
The boundary-layer approximation

Atmospheric turbulence studies started by borrowing concepts from Mechanical Engineering and the flow in channels and in pipes. In many cases these flows are almost homogeneous along the x direction, and viscosity effects are felt in a thin layer close to the wall (Prandtl, 1905, 1928; Tani, 1977; Anderson, 2005). Generalization to turbulent boundary layers generally implies

$$\frac{\partial \overline{u}}{\partial z} \gg \frac{\partial \overline{u}}{\partial x},$$
$$\frac{\partial \overline{T}}{\partial z} \gg \frac{\partial \overline{T}}{\partial x},$$

and this leads to the well-known Prandtl boundary-layer equations in different forms.

Figure from Prandtl (1928)



BL "eliminates" scales and boundary conditions

- Downstream boundary conditions no longer needed: elliptic → hyperbolic.
- Dimensionless quantities can now be defined.
- A dimensionless ODE can now be solved (Blasius' solution).

A simpler example: instant injection of a tracer in an infinite domain

$$
\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2}, \qquad c(x,0) = \frac{M}{A}\delta(x), \qquad c(\pm\infty, t) = 0,
$$
$$
\int_{-\infty}^{+\infty} c(x,t)\,\mathrm{d}x = \frac{M}{A}, \qquad \forall t.
$$

In principle we need to find a function of ***two*** variables, $c(x,t)$. However, because there is no explicit x scale (because the domain is infinite in x), we find two dimensionless variables only,

$$
\xi = \frac{x}{\sqrt{4Dt}}, \qquad \chi = \frac{c(x,t)A\sqrt{4Dt}}{M},
$$
$$
\chi = f(\xi), \qquad \ldots
$$

…reducing the problem to …

$$
\frac{\mathrm{d}}{\mathrm{d}\xi}\left[\frac{\mathrm{d}f}{\mathrm{d}\xi}+2\xi f\right]=0, \qquad f(\pm\infty)=0, \qquad \Rightarrow f(\xi)=\frac{1}{\sqrt{\pi}}\mathrm{e}^{-\xi^2},
$$

$$
c(x,t)=\frac{M}{A\sqrt{4Dt}}\exp\left[-\frac{x^2}{4Dt}\right].
$$

If you add scales, the solution depends on more variables:

$$
\frac{\partial c}{\partial t}=D\frac{\partial^2 c}{\partial x^2}, \qquad c(x,0)=c_0 f(x), \qquad c(0,t)=c(L,t)=0,
$$

$$
c(x,t)=c_0\sum_{n=1}^{\infty}A_n\mathrm{e}^{-\frac{n^2\pi^2 D}{L^2}t}\sin\frac{n\pi x}{L}, \qquad A_n=\frac{2}{L}\int_0^L f(x)\sin\frac{n\pi x}{L}\,\mathrm{d}x,
$$

and now there are 3 dimensionless variables:

$$
\frac{c}{c_0}, \qquad \frac{x}{L}, \qquad \frac{Dt}{L^2}.
$$

Few scales leads to (semi)analytical success

The combination of ***dimensionless variables*** obtained from relatively ***few length scales*** in prototypical problems has led to many sucesses in Fluid Mechanics:

- In laminar boundary-layer theory, to the Blasius's solution.
- In flow in ducts under pressure, to the log velocity profile and the Moody diagram for head loss calculations.
- ⋮
- In Monin-Obukhov Similarity Theory, we no longer have fully closed equations, but we still assume a few "infinities" (homogeneity in x, no influence of the height of the atmospheric boundary-layer), and hope that a flat terrain and the existence of an inertial sublayer will help.

But will them?

From: https://www.attoproject.org/adjustments-to-the-law-of-the-wall-above-an-amazon-forest/

What is going on? TKE budget estimated from LES

$$\frac{T^h + \Pi + A^h + P_u^h + P_v^h + P_w^h}{\varepsilon_e} \equiv \frac{N}{\varepsilon_e},$$
$$-\frac{S}{\varepsilon_e} + \frac{P_u^z + P_w^z}{\varepsilon_e} + \frac{B}{\varepsilon_e} + \frac{T^z + A^z}{\varepsilon_e} + \frac{N}{\varepsilon_e} = 1.$$

Normalized TKE budget terms derived from the LES The last column is the sum of all terms: if the subgrid-scale stresses and pressure terms are sufficiently small, the columns should sum to 1

Laba et al. (2025): LES TKE for the ATTO site

Height (m)	P_u^z/ε_e	P_w^z/ε_e	$P_{u\&w}^h/\varepsilon_e$	T^z/ε_e	T^h/ε_e	A^z/ε_e	A^h/ε_e	S/ε_e	$\sum$
LES averaged over 3 stencil points (6 m) in the vertical									
35	0.9069	−0.0235	0.0489	0.1906	0.1117	−0.1366	0.0592	−0.0274	1.1298
42	1.7069	−0.0206	0.0708	−0.2342	0.0928	−0.1755	0.0814	−0.0487	1.4730
50	1.8621	0.0082	0.0644	−0.6071	0.0469	−0.0361	−0.0220	−0.0491	1.2673
81	0.5208	−0.0569	0.1168	0.0809	0.0596	−0.0004	0.3518	−0.0796	0.9929

Upshot

- We are now facing complicated geometries due to topography, surface features, urban environments, ...
- This may introduce ***many*** new length scales, velocity scales, scalar scales, etc..
- The elegant approach of dimensionless solutions and dimensionless empirical functions may not work anymore.

Upshot

- We are now facing complicated geometries due to topography, surface features, urban environments, …
- This may introduce ***many*** new length scales, velocity scales, scalar scales, etc..
- The elegant approach of dimensionless solutions and dimensionless empirical functions may not work anymore.

We need (many more) numerical simulations:

- They should be easy to develop and deploy.
- They should be possible to run in our desk computers.
- Note that high-end personal computers with 8, 16, 32, …***cores*** are becoming increasingly common and more affordable.
- Several ***nodes*** and/or several ***cores*** means parallel programming.

The Chapel programming language

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

From Kayraklioglu, Laurendeau & Zayni, Adv Model and Simul Seminar Series, NASA Ames Research Center Feb 20th 2025: https://www.nas.nasa.gov/assets/nas/pdf/ams/2025/AMS_20250220_Kayraklioglu.pdf

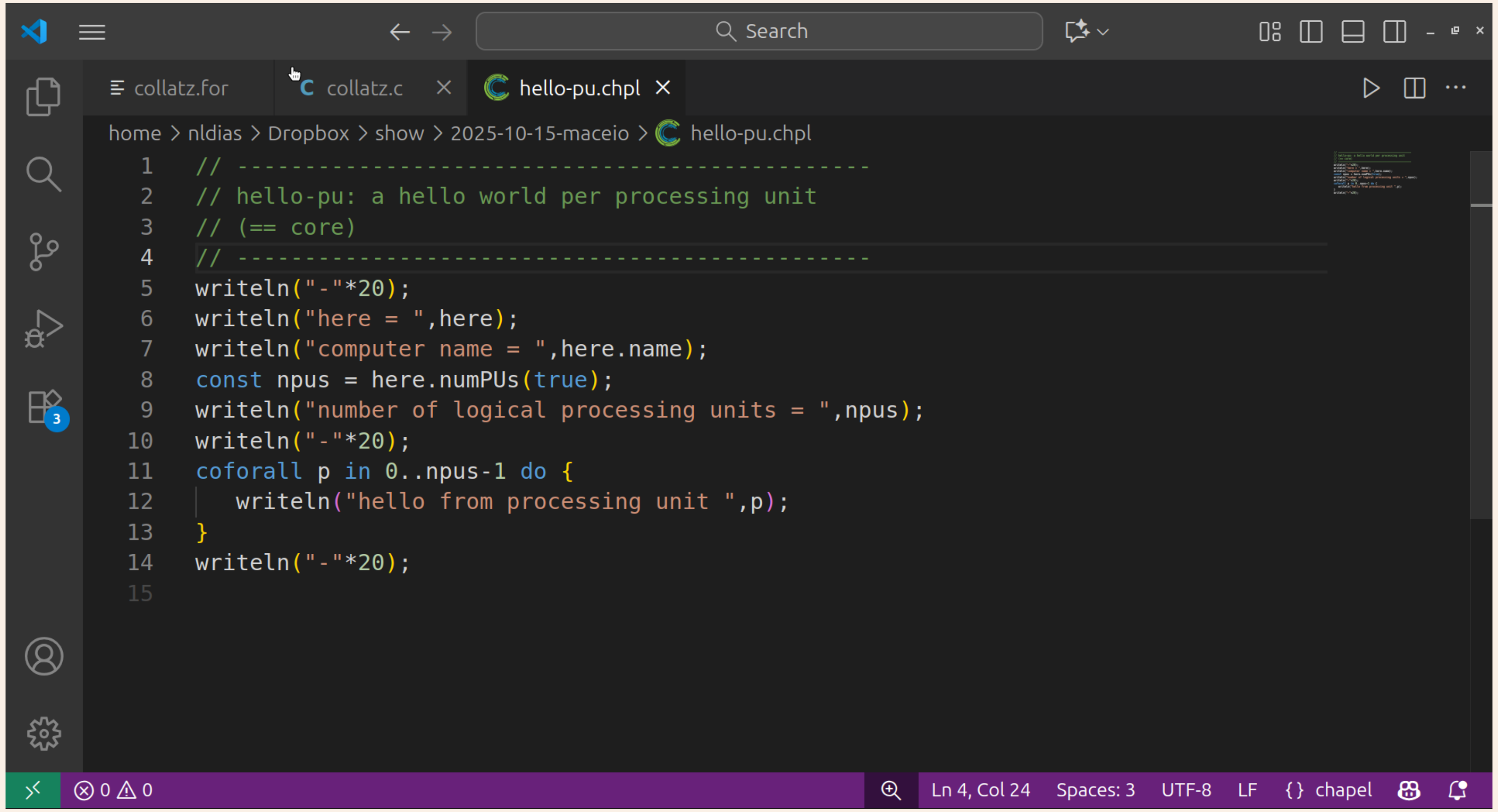
4

A short list of nice features (personal view)

- Easy to start using.
- Easy to translate an existing [Fortran or C or Python or …] code to Chapel.
- Interoperability: Can link a Fortran, or C, or Python library with Chapel. This allows reuse of existing software libraries without the need to re-program everything from scratch.
- Modular, Python-style: every Chapel file (extension `chpl`) is a module: it can be a program or it can be a library. There is no separate compilation of Chapel modules: this is ***simpler*** than the separate compilation schemes of C and Fortran.
- Powerful arrays, Fortran-style. Arbitrary lower and upper bounds; slicing possible (but not as efficient as Fortran).
- Memory-safe, except for some uses of classes (`https://chapel-lang.org/blog/posts/memory-safety/`).
- Runs anywhere, from notebooks to supercomputers. Runs in CPUs and GPUs.
- Powerful but simple commands for parallelization. ***Much easier*** than OpenMP and MPI.
- Simple and straightforward distribution of arrays and tasks among cores in a single node, and among many nodes (of a cluster).

Good resources to program in Chapel

- Excellent community suport
- Good on-line documentation https://chapel-lang.org/
- Syntax highlighting for emacs, vi, vs-code



Chapel's hello world

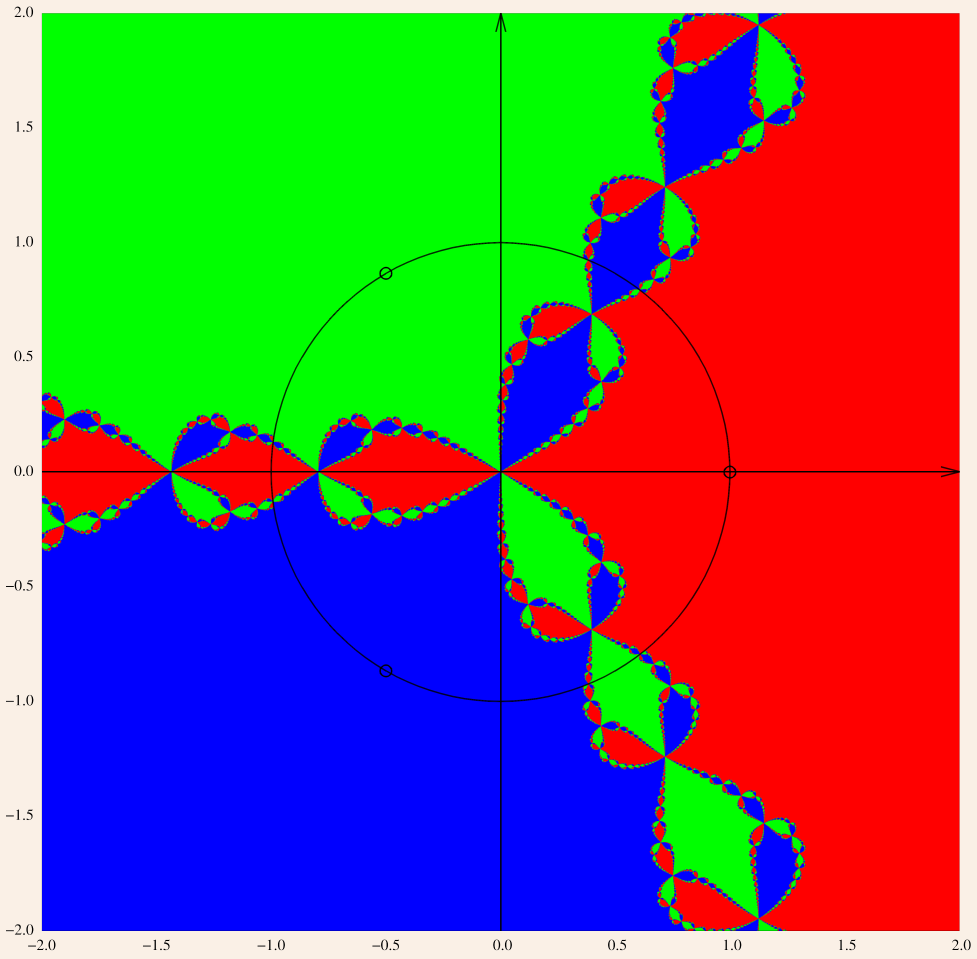
```
// ------------------------------------------------
// hello-pu: a hello world per processing unit
// (== core)
// ------------------------------------------------
writeln("-"*20);
writeln("here␣=␣",here);
writeln("computer␣name␣=␣",here.name);
const npus = here.numPUs(true);
writeln("number␣of␣logical␣processing␣units␣=␣",npus);
writeln("-"*20);
coforall p in 0..npus-1 do {
   writeln("hello␣from␣processing␣unit␣",p);
}
writeln("-"*20);
```

```
$chpl hello-pu.chpl
$./hello-pu
--------------------
here = LOCALE0
computer name = athome
number of logical processing units = 16
--------------------
hello from processing unit 1
hello from processing unit 3
hello from processing unit 10
hello from processing unit 4
hello from processing unit 11
hello from processing unit 2
hello from processing unit 13
hello from processing unit 12
hello from processing unit 5
hello from processing unit 15
hello from processing unit 9
hello from processing unit 8
hello from processing unit 6
hello from processing unit 7
hello from processing unit 14
hello from processing unit 0
--------------------
```

In Chapel, ***locales*** are the nodes of a cluster; ***processing units*** are the cores in each locale.
This program runs on a single locale, and parallelizes over 16 cores.

Example: find a root in the complex plane

Obtain the root of $z^3 - 1 = 0$ with initial guess from each pixel of a 4×4 square in the complex plane and color-code it according to the root found: $1 + 0\mathrm{i}$ = red, $-1/2 + \mathrm{i}\sqrt{3}/2$ = green, $-1/2 - \mathrm{i}\sqrt{3}/2$ = blue.



Chapel code: zxynewton-p.chpl

```
use IO only openWriter;
config const N = 2048;
const colors = {0,1,2};
var Ncol: [colors] int = 0;
var xycolor: [-N..N,-N..N] int(8);
var zinit: [-N..N,-N..N] complex;
const eps = 1.0e-6;
const fou = openWriter("zxynewton-p.out");
var dxy = 2.0/N;
for i in -N..N {
   forall j in -N..N {
      zinit[i,j] = i*dxy + (j*dxy)*1i;
      var z0 = zinit[i,j];
      iterate(z0);
      choosecolor(z0,i,j);
      Ncol[xycolor[i,j]] += 1;
   }
}
writeln(Ncol);
fouwrite;
fou.close();
proc f(const in zin: complex): complex {
   return  (zin**3 - 1.0);
}
proc fl(const in zin:complex): complex {
   return (3*zin**2);
}
proc iterate(ref z0: complex) {
   var z, fz0: complex;
   var delta = eps;
   while (delta >= eps) {
      fz0 = f(z0);
      z = z0 - fz0/fl(z0);
      delta = abs(z - z0);
      z0 = z;
   }
}
proc choosecolor(
   const in z0: complex,
   const in i: int,
   const in j: int) {
   if isClose(z0.im,0.0,absTol=1.0e-5) {
      xycolor[i,j] = 0; // red
   }
   else if isClose(z0.im,sqrt(3.0)/2.0,absTol=1.0e-5) {
      xycolor[i,j] = 1; // green
   }
   else {
      xycolor[i,j] = 2; // blue
   }
}
```

fouwrite code not shown.

What do I get form parallelizing?

program	time (s)
`zxynewton-p`	0m8.738s
`zxynewton`	0m29.076s

Finally, a case where Chapel parallelizes very efficiently

Consider the two-dimensional diffusion equation

$$\frac{\partial \phi}{\partial t} = D\left[\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right]$$

with simple initial and boundary conditions

$$\begin{aligned}
\phi(0,x,y) &= 1, & 0 < x < L_x, && 0 < y < L_y,\\
\phi(t,0,y) = \phi(t,L_x,y) &= 0, & t > 0, && 0 \le y \le M,\\
\phi(t,x,0) = \phi(t,x,L_y) &= 0, & t > 0, && 0 \le x \le L.
\end{aligned}$$

This has an analytical solution (modified from Kreider et al. (1966, section 13–7)) of the form

$$\phi(t,x,y) = \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} A_{mn} \sin\left(\frac{(2m+1)\pi x}{L_x}\right) \sin\left(\frac{(2n+1)\pi y}{L_y}\right) \times \exp\left\{-\pi^2 D\left[((2m+1)/L_x)^2 + ((2n+1)/L_y)^2\right] t\right\},$$

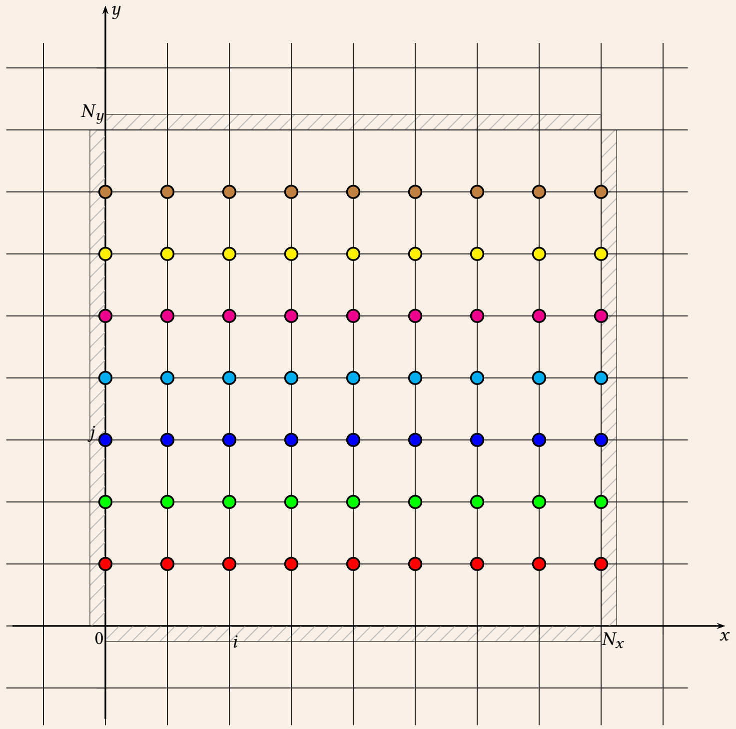
where

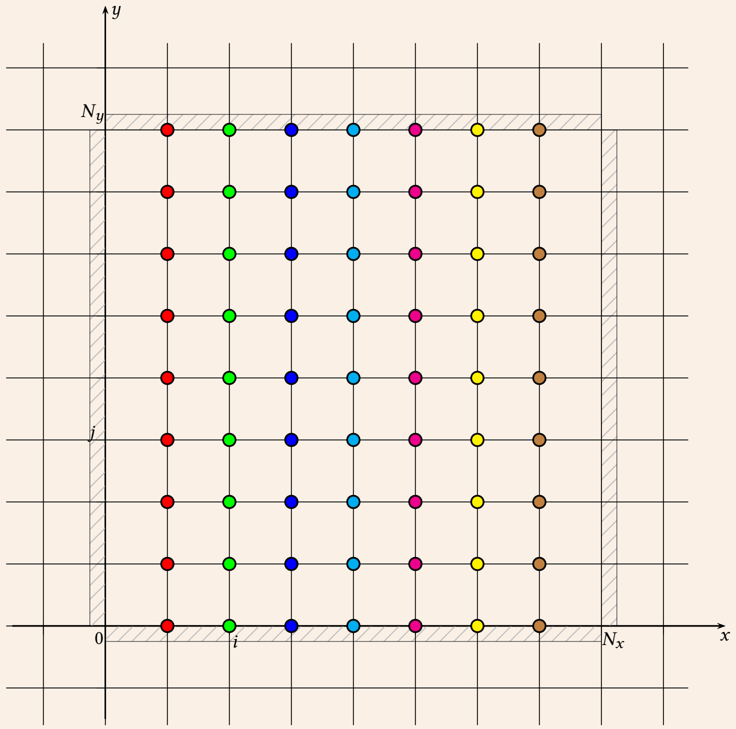
$$A_{mn} = \frac{16}{\pi^2(2m+1)(2n+1)}.$$

Solution with the clever ADI method

$$\frac{\phi_{n+1,i,j} - \phi_{n,i,j}}{\Delta t} = D\left(\frac{\phi_{n+1,i+1,j} - 2\phi_{n+1,i,j} + \phi_{n+1,i-1,j}}{\Delta x^2} + \frac{\phi_{n,i,j+1} - 2\phi_{n,i,j} + \phi_{n,i,j-1}}{\Delta y^2}\right),$$

$$\frac{\phi_{n+2,i,j} - \phi_{n+1,i,j}}{\Delta t} = D\left(\frac{\phi_{n+1,i+1,j} - 2\phi_{n+1,i,j} + \phi_{n+1,i-1,j}}{\Delta x^2} + \frac{\phi_{n+2,i,j+1} - 2\phi_{n+2,i,j} + \phi_{n+2,i,j-1}}{\Delta y^2}\right).$$





Parallelization gain & speedup

```
for j in 1..Ny-1 do {
   D[1..Nx-1] = Fon*phi[iold,1..Nx-1,j-1]
      + (1.0 - 2*Fon)*phi[iold,1..Nx-1,j]
      + Fon*phi[iold,1..Nx-1,j+1];
   D[1] += Fon*phi[inew,0,j];        // left BC
   D[Nx-1] += Fon*phi[inew,Nx,j];    // right BC
   tridiag(A,B,C,D,phi[inew,1..Nx-1,j]);
}
```

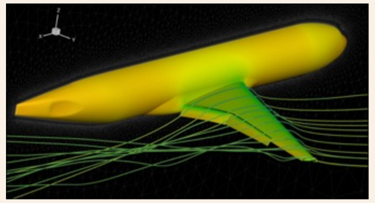
```
forall j in 1..Ny-1 do {
   var D: [1..Nx-1] real;
   D[1..Nx-1] = Fon*phi[iold,1..Nx-1,j-1]
      + (1.0 - 2*Fon)*phi[iold,1..Nx-1,j]
      + Fon*phi[iold,1..Nx-1,j+1];
   D[1] += Fon*phi[inew,0,j];
   D[Nx-1] += Fon*phi[inew,Nx,j];
   tridiag(A,B,C,D,phi[inew,1..Nx-1,j]);
}
```

N_n	serial	parallel	speedup
128	1.0299	0.0415	24.79
256	2.2347	0.0980	22.80
512	5.3224	0.2537	20.98
1024	13.9097	0.8833	15.75
2048	41.6813	3.6751	11.34
4096	171.7800	18.1090	9.48

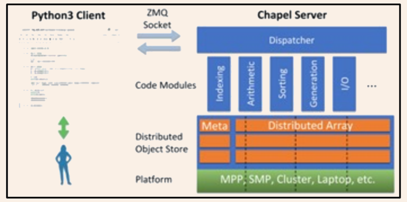
On 8 physical cores, 16 logical cores with hyperthreading.

Real Applicatons

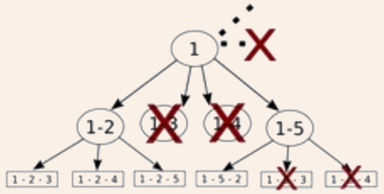
Applications of Chapel



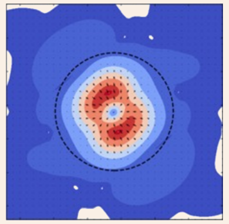
CHAMPS: 3D Unstructured CFD
aurendeau, Bourgault-Côté, Parenteau, Plante, et a
École Polytechnique Montréal



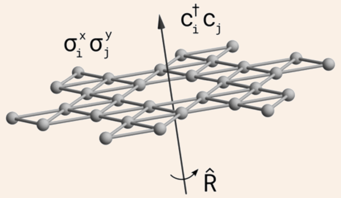
rkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



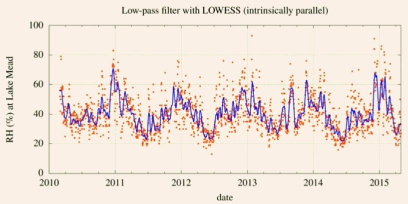
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



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

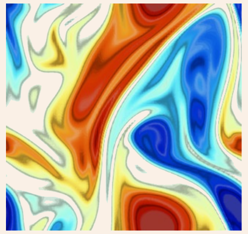


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

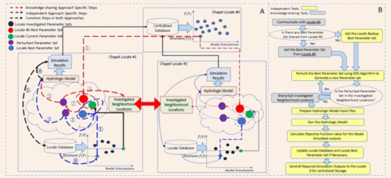


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

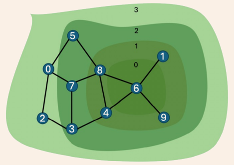
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



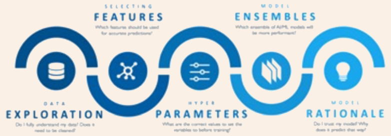
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



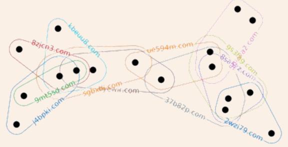
Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



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] 5

From Kayraklioglu, Laurendeau & Zayni, Adv Model and Simul Seminar Series, NASA Ames Research Center Feb 20th 2025: https://www.nas.nasa.gov/assets/nas/pdf/ams/2025/AMS_20250220_Kayraklioglu.pdf

Progress towards our own Chapel use

Parallel implementation in Chapel for the numerical solution of the 3D Poisson problem

Anna Caroline Felix Santos de Jesus
carolinefelix@usp.br
Instituto de Ciências Matemáticas e de Computação, University of São Paulo
São Carlos, Brazil

Willian Carlos Lesinhovski
wlesin@yahoo.com.br
Department of Environmental Engineering, Federal University of Paraná
Curitiba, Brazil

Livia S. Freire
liviafreire@usp.br
Instituto de Ciências Matemáticas e de Computação, University of São Paulo
São Carlos, Brazil

Nelson Luis Dias
nldias@ufpr.br
Department of Environmental Engineering, Federal University of Paraná
Curitiba, Brazil

From de Jesus et al. (2023)

Trends in Computational and Applied Mathematics
Trends in Computational and Applied Mathematics, **26** (2025), e01833
Sociedade Brasileira de Matemática Aplicada e Computacional
Online version ISSN 2676-0029 www.scielo.br/tcam
ORIGINAL ARTICLE
doi: 10.5540/tcam.2025.026.e01833

On the relative merits of interpolation schemes for the immersed boundary method: a case study with the heat equation

W. C. LESINHOVSKI[1*], N. L. DIAS [2], L. S. FREIRE [3] and A. C. F. S. JESUS[4]

From Lesinhovski et al. (2025)

A parallel implementation of the immersed boundary method in the Chapel programming language

Willian Carlos Lesinhovski
Postgraduate Program in Environmental Engineering, Federal University of Paraná
Curitiba, Brazil
wlesin@yahoo.com.br

Nelson Luís Dias
Department of Environmental Engineering, Federal University of Paraná
Curitiba, Brazil
nelsonluisdias@gmail.com

Abstract
In this work a parallelizable numerical scheme for the Navier-Stokes equations in two dimensions using the Immersed Boundary Method is presented. The algorithm was implemented in the Chapel programming language, providing a speed-up factor close to 6 on a personal computer.

Keywords
Numerical analysis, Nonlinear equations, Parallel programming languages.

ACM Reference Format:
Willian Carlos Lesinhovski and Nelson Luís Dias. 2025. A parallel implementation of the immersed boundary method in the Chapel programming language. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3731599.3767503

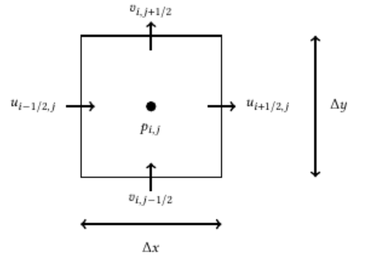
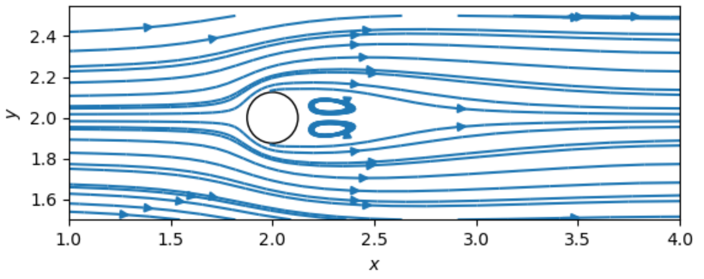


Figure 1: Illustration of one grid cell and the position of the variables in the staggered grid.



From Lesinhovski and Dias (2025)

Boundary-Layer Meteorology (2025) 191:36
https://doi.org/10.1007/s10546-025-00932-x
RESEARCH ARTICLE

Topography-Induced TKE Budget Behavior Over an Amazon Forest

Paulo Henrique Laba[1] · Nelson Luís Dias[2] · Marcelo Chamecki[3] · Cléo Quaresma Dias-Júnior[4] · Gregory Torkelson[3]

Received: 21 January 2025 / Accepted: 17 July 2025

From Laba et al. (2025)

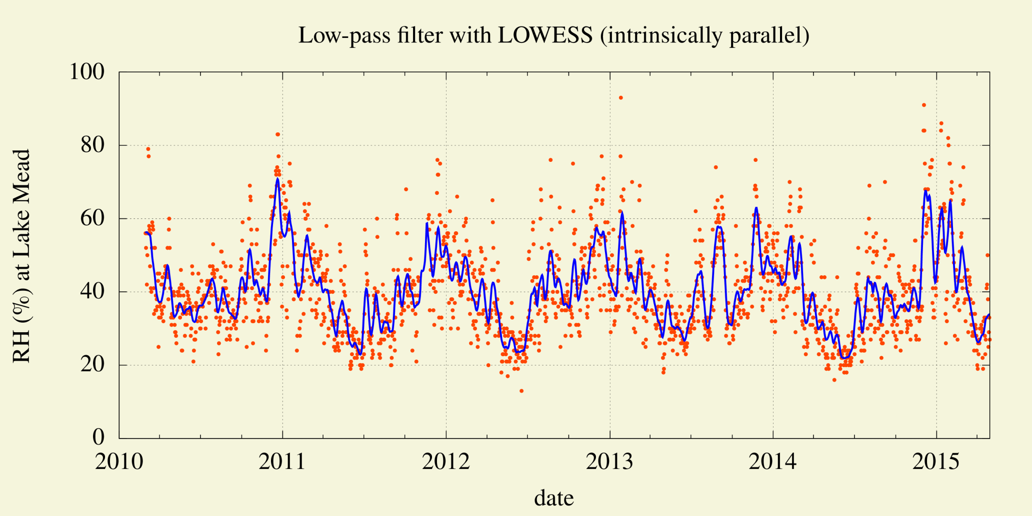
Open Access
Full Paper
Journal of Agricultural Meteorology 81(2): 73–89, 2025

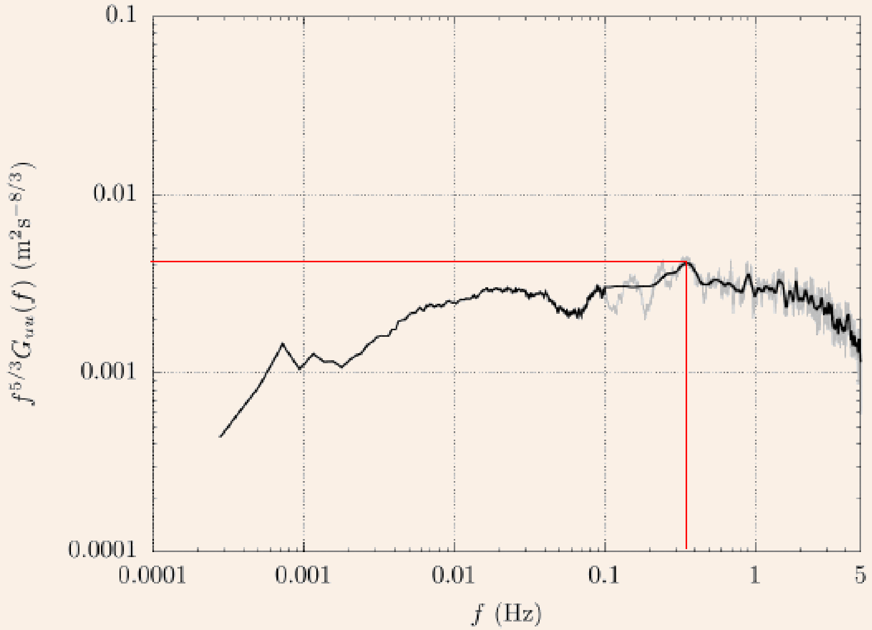
Validation of the STAEBLE lake evaporation model in mountainous terrain

Henrique F. Duarte[a,c,†], Nelson Luís Dias[a,b] and Hiroki Iwata[c,d]
[a]Graduate Program in Environmental Engineering, Federal University of Paraná, Centro Politécnico UFPR, Curitiba, PR, 81531-980-PO Box 19011, Brazil
[b]Department of Environmental Engineering, Federal University of Paraná, Centro Politécnico UFPR, Curitiba, PR, 81531-980-PO Box 19011, Brazil
[c]Department of Environmental Science, Shinshu University, 3-1-1 Asahi, Matsumoto, Nagano, 390-8621, Japan
[d]Institute for Mountain Science, Shinshu University, 3-1-1 Asahi, Matsumoto, Nagano, 390-8621, Japan
[e]Institute for the Study of Earth, Oceans, and Space, University of New Hampshire, Morse Hall, 8 College Road, Durham, NH, 03824, USA.

From Duarte et al. (2025)

Also, visit my page: https://nldias.github.io/software.html





ada "attached domain arrays" can be used as arguments in first-class functions

angles from radians to decimals and the other way around:

atmgas atmospheric gas properties and manipulation

dgrow grow an array to fit an index

evap procedures to calculate evaporation in hydrology

nspectrum the spectrum from a single FFT

nstat statistical stuff (including Lowess and Levenberg-Marquardt!)

nstrings additional string manipulation

planfit the planar fit method for micrometeorology

pmatrix parallel matrix operations

smatrix serial matrix operations

ssr sort, search, etc. in an array

sunearth simple astronomical calculations useful in hydrology and meteorology

water properties of liquid water

Conclusions

Conclusions

- Desktops and notebooks are now up to the task of performing useful and detailed Fluid Mechanics simulations.
- They will get even more powerful in the coming years.
- Leveraging their power however, requires parallel programming

Pros:

- Chapel is modern, powerful, elegant, generates fast code, and is easy to learn.
- A single language can be used to most tasks, like statiscal processing of data, fluid mechanics simulations, various file manipulations, etc..
- Chapel makes parallel programming easy.
- Chapel runs from notebooks to supercomputers, and is very portable.

Cons:

- Not interactive.
- Compilation can take up to 3-5 s.
- Not a large application base yet (but remember you can use Python, C and Fortran libraries inside Chapel).

Thanks for the attention!

References

Anderson, J. D. (2005, Dec). Ludwig prandtl's boundary layer. *Phys. Today 58*(12), 42–48.

de Jesus, A. C. F. S., W. C. Lesinhovski, L. S. Freire, and N. L. Dias (2023). Parallel implementation in chapel for the numerical solution of the 3d poisson problem. In *CHIUW 2023: 10th annual Chapel Implementers and Users Workshop, a free, online event for the Chapel programming language community held on June 1-2, 2023*, Online conference.

Duarte, H. F., N. L. Dias, and H. Iwata (2025). Validation of the staeble lake evaporation model in mountainous terrain. *Journal of Agricultural Meteorology 81*(2), 73–89.

Kreider, D. L., R. G. Kuller, D. R. Ostberg, and F. W. Perkins (1966). *An Introduction to Linear Analysis.* Reading, Massachusetts: Addison-Wesley Publishing Company.

Laba, P., N. Dias, M. Chamecki, C. Dias-Júnior, and G. Torkelson (2025). Topography-induced tke budget behavior over an amazon forest. *Boundary-Layer Meteorol 191*, 36.

Lesinhovski, W. C. and N. L. Dias (2025). A parallel implementation of the immersed boundary method in the chapel programming language. In *The 8th Annual Parallel Applications Workshop, Alternatives To MPI+X.*

Lesinhovski, W. C., N. L. Dias, L. S. Freire, and A. C. F. S. d. Jesus (2025). On the relative merits of interpolation schemes for the immersed boundary method: a case study with the heat equation. *Trends in Computational and Applied Mathematics 26*, e01833.

Prandtl, L. (1905). Presentation in congress; title not known. In A. Krazer (Ed.), *Verhandlungen des dritten internationalen Mathematiker-Kongresses in Heidelberg 1904 (Proceedings of the Third International Congress of Mathematicians in Heidelberg 1904)*, pp. 484–491.

Prandtl, L. (1928). Motion of fluids with very little viscosity. Technical Memorandum 452, NACA. translated from Ueber Flussigkeitsbeegung bei sehr kleiner Reibung, Vier Abhandlungen zur Hydrodynamik und Aerodynamik, pp. 1–8, Gottingen, 1927.

Tani, I. (1977). History of boundary layer theory. *Annual review of fluid mechanics 9*(1), 87–111.