From C and Python to Chapel as my main Programming Language
CHIUW 2022

Nelson Luís Dias[1]

[1]Professor, Environmental Engineering Department, Federal University of Paraná (Brazil), e-mail: nldias@ufpr.br url: https://nldias.github.io

June 10 2022

Interests, Research Focus, *My* Languages

About me: I am a Civil Engineer, and my research areas are Hydrology and Atmospheric Turbulence. *Most* of my research involves data processing, and some of it involves numerical methods (like CFD).

Interests, Research Focus, My Languages

About me: I am a Civil Engineer, and my research areas are Hydrology and Atmospheric Turbulence. Most of my research involves data processing, and some of it involves numerical methods (like CFD).

Languages in Learning:

Fortran 66 First language, with punched cards, on a PDP-11.

Pascal (Jensen and Wirth, 1974) When I learned to program with discipline.

Interests, Research Focus, My Languages

About me: I am a Civil Engineer, and my research areas are Hydrology and Atmospheric Turbulence. Most of my research involves data processing, and some of it involves numerical methods (like CFD).

Languages in Learning:

Fortran 66 First language, with punched cards, on a PDP-11.

Pascal (Jensen and Wirth, 1974) When I learned to program with discipline.

Languages in Research:

Vax Fortran `IF-THEN-ELSE`, `DO-WHILE`, `END DO`, etc..
MODULA-2, Turbo Pascal On the desktop.
C, Fortran-9X On the desktop, for a long time.
Python On the desktop, since 2008, until 2020.
Chapel On the desktop, since 2020.

Interests, Research Focus, My Languages

About me: I am a Civil Engineer, and my research areas are Hydrology and Atmospheric Turbulence. Most of my research involves data processing, and some of it involves numerical methods (like CFD).

Languages in Learning:

Fortran 66 First language, with punched cards, on a PDP-11.

Pascal (Jensen and Wirth, 1974) When I learned to program with discipline.

Languages in Research:

Vax Fortran `IF-THEN-ELSE`, `DO-WHILE`, `END DO`, etc..
MODULA-2, Turbo Pascal On the desktop.
C, Fortran-9X On the desktop, for a long time.
Python On the desktop, since 2008, until 2020.
Chapel On the desktop, since 2020.

I have no experience with Clusters, supercomputers, etc..

Subjective evaluations of languages

- Dijkstra, E. W. (1968). Go to statement considered harmful. *Commun ACM*, 11(3):147–148
- "premature optimization is the root of all evil" Knuth, D. E. (1974). Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, 6(4):261–301
- Kernighan, B. W. (1981). Why pascal is not my favorite programming language. Computing Science Technical Report 100, AT&T Bell Laboratories, Murray Hill, New Jersey 07974

Caveats:

- I don't have any experience with C++ nor with OO features in any language.
- I am not a computer scientist (this is strictly a user's perspective).

This said, my desiderata [noun, pl: "something that is needed or wanted"] for a "best language" are

1. Relatively easy.
2. Enough constructs to handle any algorithm and large chunks of data (array slicing).
3. Fast.

A quick (subjective) assessment of Chapel

Comparison with my other "main" languages

Language	Easy	Constructs& Array Sl.	Fast
Python	Yes	Yes (Numpy)	No (`for` is slow)
C	Yes (more or less)	No (too low level)	Yes
Fortran	No (too large)	Yes	Yes
Chapel	**Yes** (but large)	**Yes**	**Yes**

An opportunity: Chapel has the potential to be a "universal language" (a little bit of marketing here)

- For the desktop, not only for clusters (this is probably my main point here: the "common" person without access to supercomputers (me) can profit a lot from it too).
- It can replace C, Fortran and Python with advantages over all of them (table above)
- It needs to be advertised, and progressively made easier to install (Linux packaging? A Windows `setup.exe`?)
- Good overall tutorials (covering most of the language) are needed.

Chapel's Strengths

- Fast, painless parallelization: I can go a long way with `forall`.
- "Index-neutral": arrays can start at any value (0, 1, whatever). This also makes it easier to port legacy code from other languages.
- Modularity: I can build my own libraries and use them easily

```
export CHPL_MODULE_PATH=/home/nldias/Dropbox/nldchapel/modules
```

- Procedures with generic types

```
proc median(ref ax: [] ?at): at { ... }
```

- Domains (perhaps *the* distinctive feature of Chapel):
  - Declare arrays with a common domain, and manage their size together.
  - Sparse domains.
  - Associative domains (also work as sets) and arrays (like AWK arrays and Python dictionaries) are very useful and integrate seamlessly into array syntax.

Some of my libraries

angles.chpl Very basic operations with angles (degrees to radians, etc.)

atmgas.chpl Concentrations, densities, and other properties of atmospheric gases.

dgrow.chpl Grow a domain dynamically to accomodate an out-of-range index.

evap.chpl Evaporation formulas and methods in hydrology.

matrix.chpl Vector and matrix multiplication, tridiagonal matrix algorithm (with a sparse domain), Gauss-Jordan inversion of a matrix, solution of a system of linear equations with Gauss elimination.

nspectrum.chpl Spectral analysis.

nstat.chpl Basic statistics, linear regression, Lowess (Cleveland, 1981) low-pass filtering.

nstrings.chpl Simple string operations.

ssr.chpl Search, sort and replace procedures for arrays.

sunearth.chpl Astronomical formulas for the trajectory of the Earth around the Sun.

turbstat.chpl Processing of turbulence data.

water.chpl Thermodynamic properties of water.

Real examples I – Laplace's equation with Successive over-relaxation

With 12 logical cores (6 physical):

Grid size N_n, number of iterations to convergence n_c, estimated $\overline{u}$, MAD and runtime t_r for the serial and parallel versions of the solution of Laplace's equation with SOR.

	serial				parallel			
N_n	n_c	$\overline{u}$	MAD	t_r (s)	n_c	$\overline{u}$	MAD	t_r (s)
128	431	0.7500	2.5625×10^{-7}	0.0647	428	0.7500	2.5278×10^{-7}	0.0107
256	1934	0.7499	1.5653×10^{-6}	0.6914	1931	0.7499	1.5609×10^{-6}	0.1152
512	6947	0.7499	6.6456×10^{-6}	9.9662	6943	0.7499	6.6525×10^{-6}	1.3221
1024	23955	0.7499	2.7032×10^{-5}	137.6410	23952	0.7499	2.7028×10^{-5}	16.7878
2048	80310	0.7498	1.0864×10^{-4}	1867.7000	80306	0.7498	1.0865×10^{-4}	232.3660

Real examples II – Grow a domain as needed

Listing 1: dgrow.chpl — A procedure to grow the first dimension of a rectangular domain by a specified factor.

```
// ------------------------------------------------------------------
// --> dgrow: grow the first dimension of a domain d by fr if the
// index i is beyond the last element of the first dimension
// ------------------------------------------------------------------
proc dgrow(
   i: int,          // an index of the first dimension
   ref d:domain,    // the domain to be grown
   fr: real=1.5     // the growth factor (1.2 grows by 20%, etc.)
) {
  assert(fr >= 0.0);                // just in case
  var dranges = d.dims();           // the ranges that constitute d
  var nfirst = dranges(0).first;    // the first index of the first dim
  var nlast =  dranges(0).last;     // the last index of the first dim
  var dsize: real = d.shape(0);     // current size of first dimension
// if i in valid range, do nothing; only grow if i is
//  next-to-last in range
  if ( i == nlast + 1 ) then {
     dsize *= fr;                      // grow it
     var nsize = dsize:int;            // back to int
     nlast = nfirst + nsize - 1;       // calculate the new last
     dranges(0) = nfirst..nlast;       // the new range of the first dim
     d = dranges;                      // resize the whole d
  }
}
```

Wish list and last words

Wish list:

- Automatic re-indexing at procedure declaration:

```
proc vmax(ref a: [1..?n] real) { ... }
```

- Procedures as arguments to procedures:

```
proc trap(in n: int, in a: real, in b: real, f: proc(x: real): real): real { ... }
```

- Faster compilation, smaller executables.

Last words of praise for Chapel:

- Elegant,
- easy to catch errors,
- generates fast and parallell code effortlessly.

Wish list and last words

Wish list:

- Automatic re-indexing at procedure declaration:

```
proc vmax(ref a: [1..?n] real) { ... }
```

- Procedures as arguments to procedures:

```
proc trap(in n: int, in a: real, in b: real, f: proc(x: real): real): real { ... }
```

- Faster compilation, smaller executables.

Last words of praise for Chapel:

- Elegant,
- easy to catch errors,
- generates fast and parallell code effortlessly.

Thank you for your attention!

References

Cleveland, W. S. (1981). Lowess: A program for smoothing scatterplots by robust locally weighted regression. *Am Stat*, 35(1):54.

Dijkstra, E. W. (1968). Go to statement considered harmful. *Commun ACM*, 11(3):147–148.

Jensen, K. and Wirth, N. (1974). *Pascal user manual and report*. Springer-Verlag, New York, 1st edition.

Kernighan, B. W. (1981). Why pascal is not my favorite programming language. Computing Science Technical Report 100, AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

Knuth, D. E. (1974). Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, 6(4):261–301.