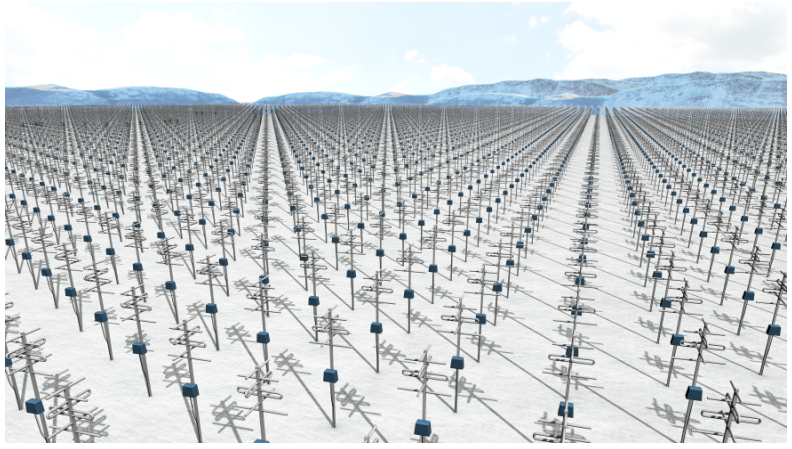
Rewriting IS data analysis

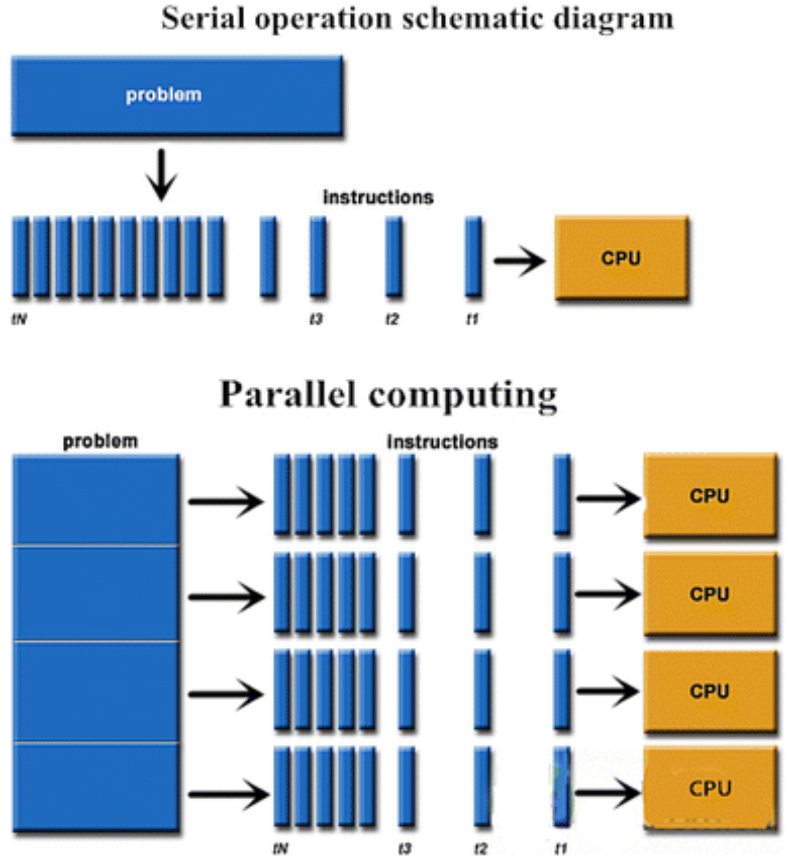
Stephan C. Buchert (scb@irfu.se)

Swedish Institute of Space Physics, Uppsala

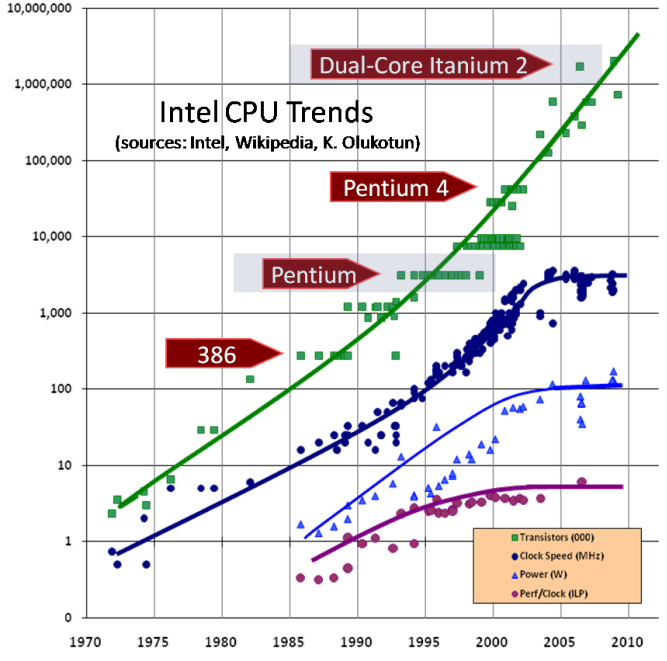
12^{th} EISCAT_3D User Meeting 2021-03-05, Online

E3D data analysis





- E3D: many beams, tristatic volume scattering, ...
- $\sim 100 - 1000$ more data to analyse!
- Moore's law seems not valid since ~ 2005:



- way to go: parallel computing (?)
- first practical steps:

The main Special Function for IS analysis

Numerical calculation of the IS spectrum for Maxwellian distributions rests on either

- the plasma dispersion function, code name `friedconte(z)`: B.D. Fried, S.D. Conte, The plasma dispersion function. New York Academic Press, 1961.
- complex error function `erfc(z)`:

$$\texttt{friedconte}(z) = i\sqrt{\pi}\exp\left(-z^2\right)\left(1+\texttt{erfc}(iz)\right) \tag{1}$$

- the Faddeeva function

$$\texttt{faddeeva}(z) = \exp\left(-z^2\right)\texttt{erfc}(-\texttt{iz}) \tag{2}$$

- the Dawson integral

$$D(x) = \exp\left(-x^2\right)\int_0^x t^2 dt \tag{3}$$

Any one of the four functions needs to be evaluated (zillions of times).

Fried-Conte

```
      SUBROUTINE PLASMA
C*THIS ROUTINE COMPUTES THE COMPLEX PLASMA DISPERSION FUNCTION
C GIVEN BY:
C             Z(S)=I*SQRT(PIE)*EXPC(-S**2)*(1.+ERFC(I*S)
C WHERE:
C         I=SQRT(-1.) ;  S=X+I*Y=COMPLEX ARGUMENT
C FOR ABS(Y).GT.1.0, THE CONTINUED FRACTION EXPANSION GIVEN BY FRIED
C AND CONTE (1961) IS USED; WHILE FOR ABS(Y).LE.1.0, THE FOLLOWING
C DIFFERENTIAL EQUATION IS SOLVED:
C           D Z(S)
C           ------ = -2.*(1.+S*Z(S))
C            D S
C SUBJECT TO Z(0)=I*SQRT(PIE)
C
C     "F(K)"=TRUE FREQUENCY.
C     "X(K)"=NORMALIZED FREQUENCY.
C     "SCALEF"=FREQUENCY SCALING FACTOR FOR NORMALIZATION.
C ----------------------------------------------------------------------
C BY WES SWARTZ
```

- Solving the diff. equation is not easy to parallelize :-(

The shortest code

Reference:
[1] J.A.C. Weideman, "Computation of the Complex Error Function," SIAM J. Numerical Analysis, pp. 1497-1518, No. 5, Vol. 31, Oct., 1994
Available Online: http://www.jstor.org/stable/2158232

TABLE 1
Matlab program for computing $w(z) = e^{-z^2}\,erfc(-iz)$ *using the series* (I).

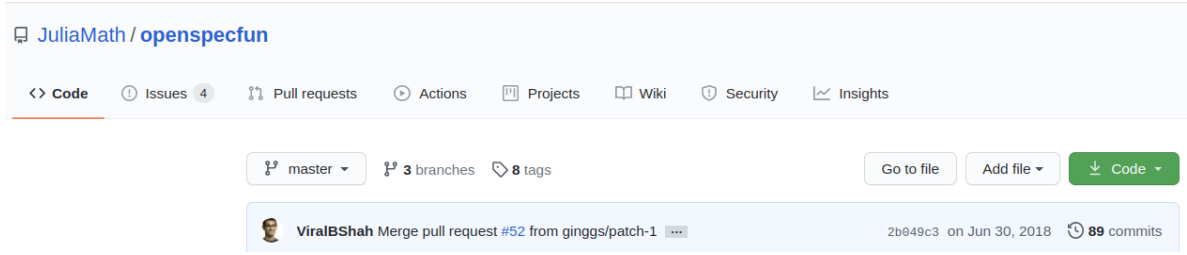
```
function w = cef(z,N)

% Computes the function w(z) = exp(-z^2) erfc(-iz) using a rational
% series with N terms. It is assumed that Im(z) > 0 or Im(z) = 0.

M = 2*N; M2 = 2*M; k = [-M+1:1:M-1]';   % M2 = no. of sampling points.
L = sqrt(N/sqrt(2));                    % Optimal choice of L.
theta = k*pi/M; t = L*tan(theta/2);     % Variables theta and t.
f = exp(-t.^2).*(L^2+t.^2); f = [0; f]; % Function to be transformed.
a = real(fft(fftshift(f)))/M2;          % Coefficients of transform.
a = flipud(a(2:N+1));                   % Reorder coefficients.
Z = (L+i*z)./(L-i*z); p = polyval(a,Z); % Polynomial evaluation.
w = 2*p./(L-i*z).^2+(1/sqrt(pi))./(L-i*z); % Evaluate w(z).
```

- evaluates at each frequency (`polyval`) $\rightarrow$ can be done in parallel;
- not very fast,
- $N = 16$ gives ~ 12 significnat digits
- probably more than good enough for IS radar.

The State of the Art




```
/* erfcx(x) = exp(x^2) erfc(x) function, for real x, written by
   Steven G. Johnson, October 2012.

   This function combines a few different ideas.

   First, for x > 50, it uses a continued-fraction expansion (same as
   for the Faddeeva function, but with algebraic simplifications for z=i*x).

   Second, for 0 <= x <= 50, it uses Chebyshev polynomial approximations,
   but with two twists:

      a) It maps x to y = 4 / (4+x) in [0,1].  This simple transformation,
         inspired by a similar transformation in the octave-forge/specfun
         erfcx by Soren Hauberg, results in much faster Chebyshev convergence
         than other simple transformations I have examined.

      b) Instead of using a single Chebyshev polynomial for the entire
         [0,1] y interval, we break the interval up into 100 equal
         subintervals, with a switch/lookup table, and use much lower
         degree Chebyshev polynomials in each subinterval. This greatly
         improves performance in my tests.
```

- fast and very accurate according to a review;
- accessible from, e.g., Python via SciPy
- written by Steven G. Johnson, MIT, in C/C++ (2012)
- (co-authored FFTW, “photonic crystals”)
- involved in the Julia programming language

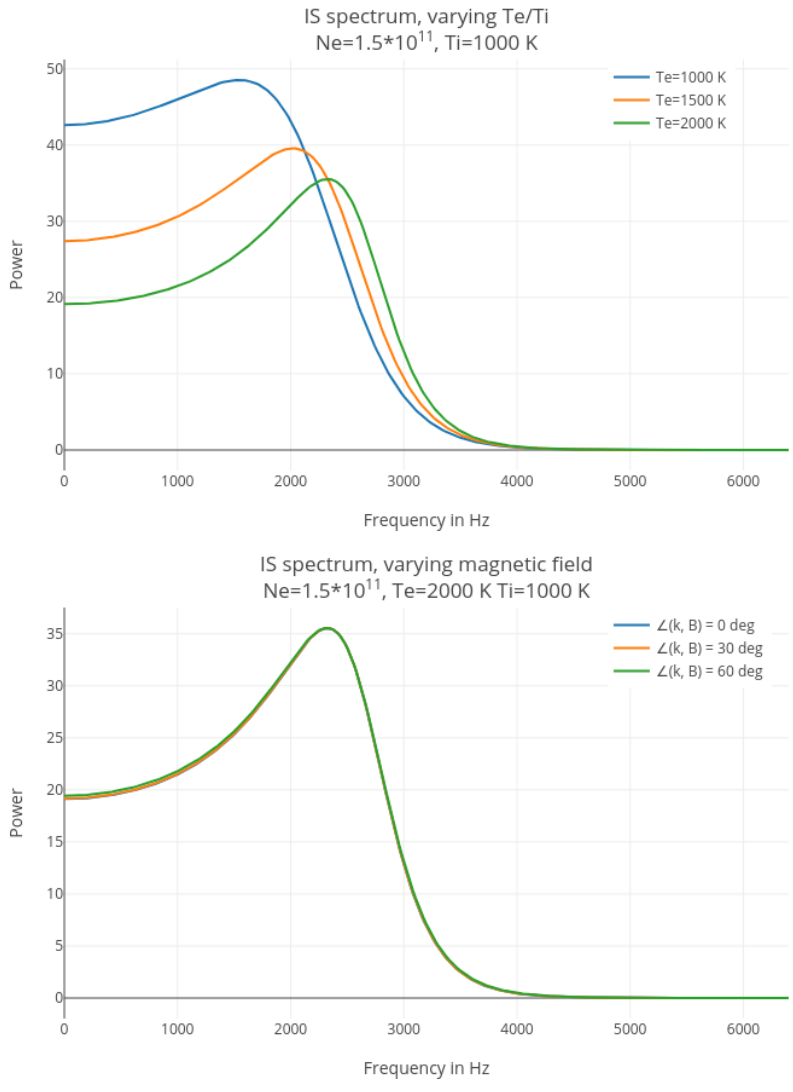
www.julialang.org

Fast

Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

- generally reaches the speed of C/C++/Fortran code
- without the need to actually program at this “low level”;
- easy to access C/C++/Fortran libraries (compared to Matlab Mex);
- easy to make use multi-core CPUs, SIMD, and AVX2/AVX512;
- also NVidia GPU (no hassle for me).
- can run in a Jupyther environment (IJulia, not tested by me).

IS data analysis in Julia



- uses multicore for loop over Fadeeva frequencies
- multicore speedup 2–3 (on 12 core i7 laptop)
- but GPU not used (C++ function);
- Fourier transform Spectrum $\rightarrow$ ACF:
- implemented as matrix multiplication $\sim 256x32$
- that seems to happen in the GPU without hassle.
- still a long way to go. . .