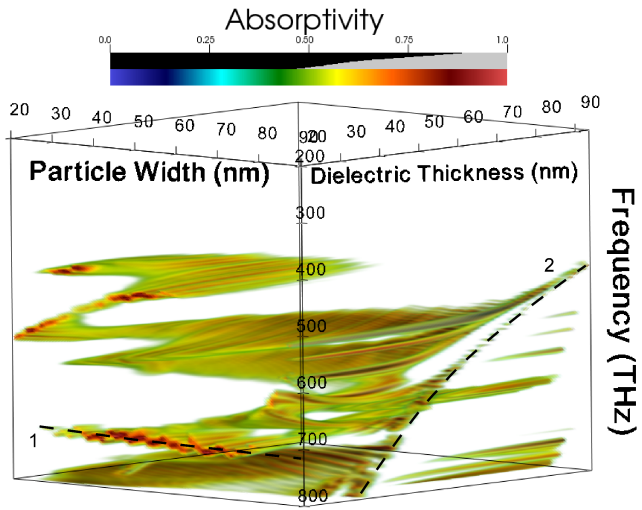
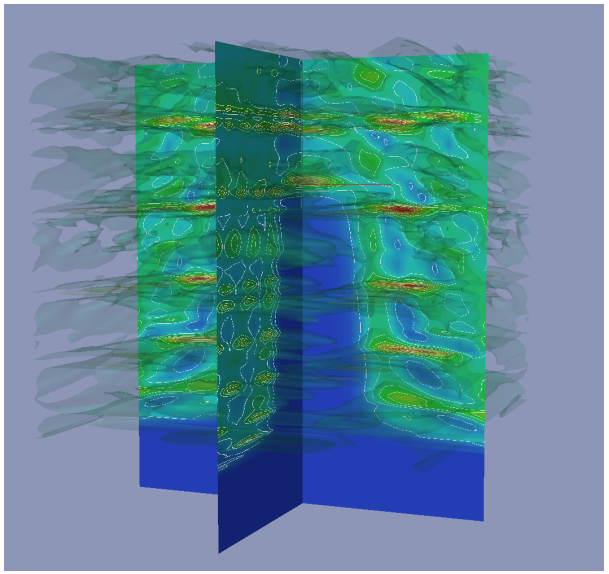
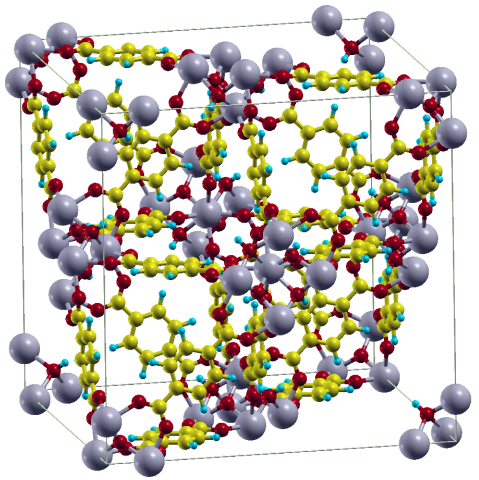
Predicting the Electrical and Thermal Transport Properties in Direct Energy Conversion Material: A Practical HPC Perspective



Plasmonic Thermal Absorber



Nanostructure Thermoelectrics



MOF Photocatalyst

Dr. Terry Musho
Assistant Professor
Department of Mechanical
and Aerospace Engineering

Outline

- What are direct energy conversion materials?
- Why are transport properties important?
- How do you calculate the electrical transport properties?
- How do you calculate the thermal transport properties?

10min

- HPC Techniques
  - Back to 1970: Using AWK to parse large data files
  - Parallelization in 2min: Using OpenMP
  - GPGPU Power: Writing OpenCL code
  - Visualizing Your Data: Writing Data in VTK format

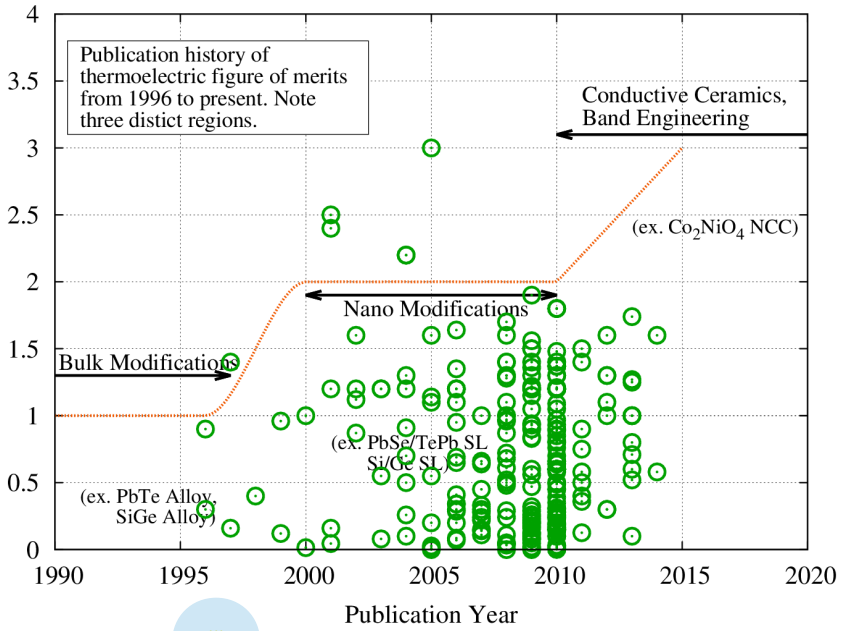
20min

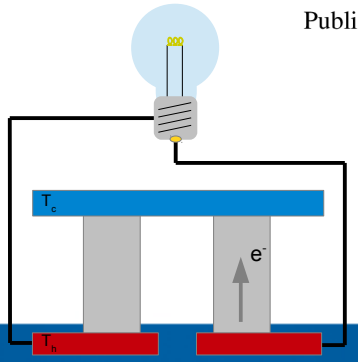
Download Examples: https://db.tt/hEIxB0tO

What are direct energy conversion materials?

Materials that convert energy directly from one form of energy to another. An example of a conversion that is not direct is a Bryton cycle (combustion->steam->turbine->generator)

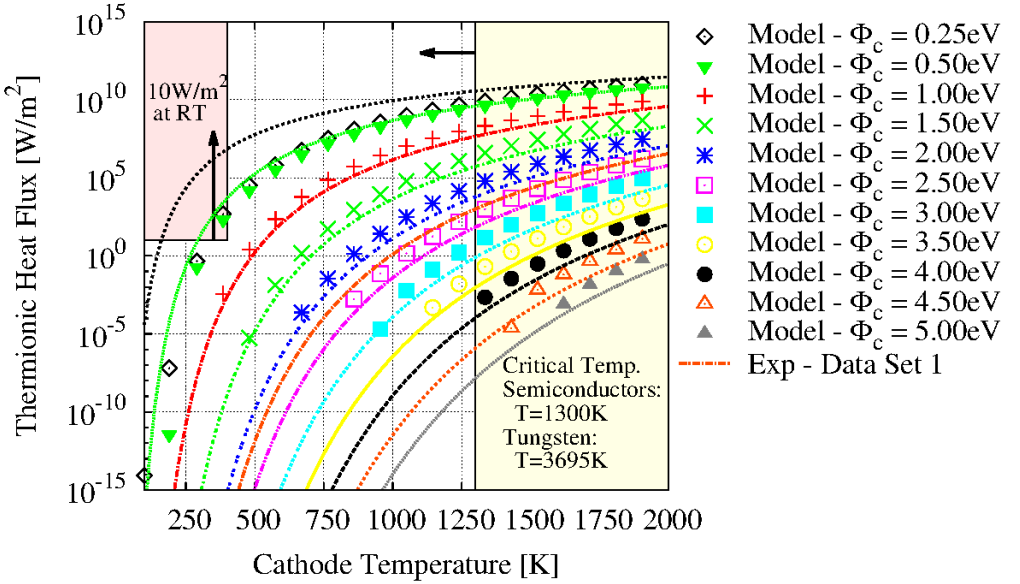
Thermoelectric Materials

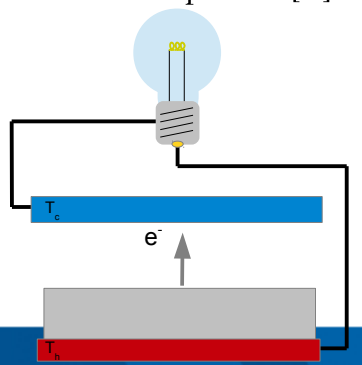




$$ZT = \frac{S^2 \sigma \bar{T}}{k_e + k_p}$$

Thermionic Materials

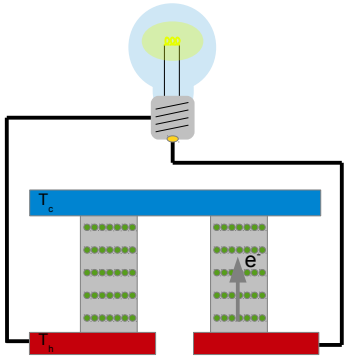




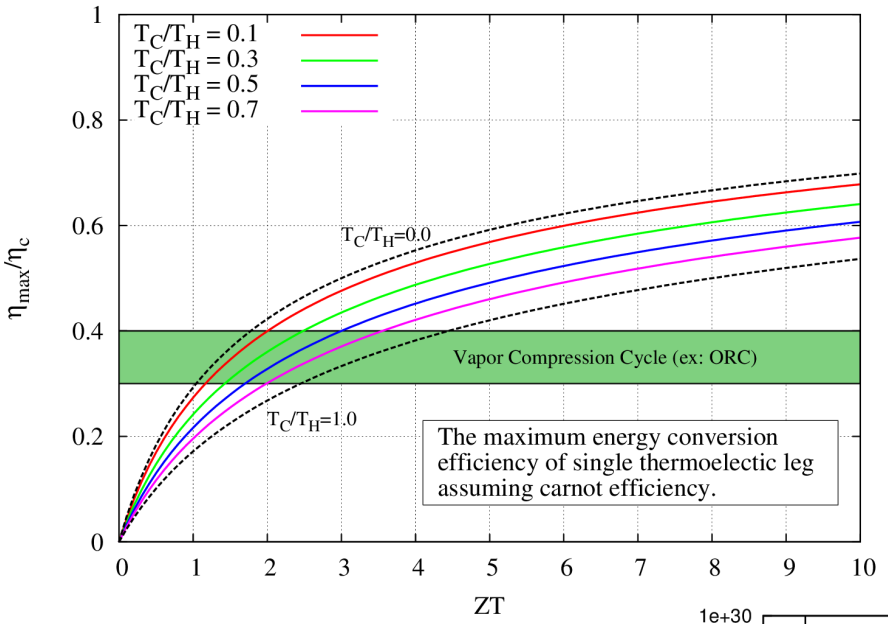
$$J = A_G T^2 \exp(\Phi / k_B T)$$

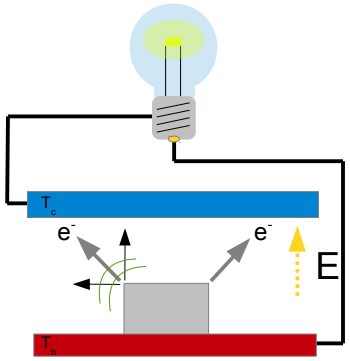
Why are transport properties important?

Transport properties govern the efficiency of the conversion process.

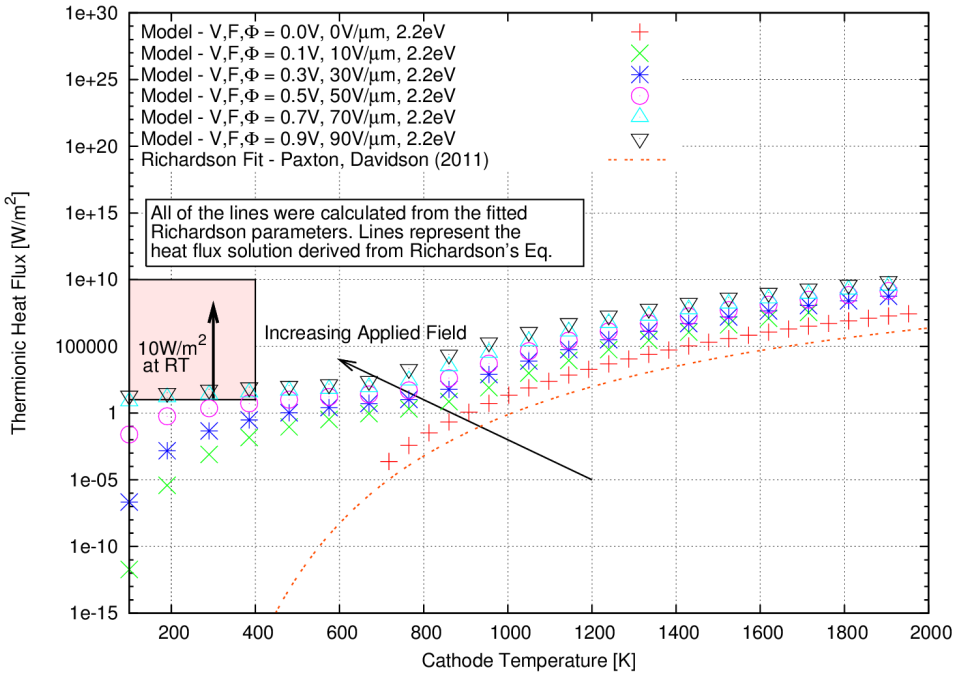


Nanostructure Thermoelectric





Nanotip Thermionic Emitter



How do you calculate the electrical transport properties?

DFT

- Effective mass
  CB and VB Edge
  Permittivity → Non-equilibrium Green's Function (NEGF) Electronic Model
- Relaxed UC
  Elastic Constants → Non-equilibrium Green's Function (NEGF) Thermal Model

$$ZT = \frac{S^2 \sigma \bar{T}}{k_e + k_p}$$

NEGF Electronic Model (2D Effective Mass Hamiltonian)

$$G = EI - H + V_{ee} + \Sigma_1 + \Sigma_2 + \Sigma_{ep} \qquad k = (k_x, k_y)$$

$$I = \int_0^{\infty} \frac{2e^2}{h} \Xi (f_2 - f_1)\, dE$$

Electron-Phonon Scattering Term

$$\Sigma_i f_s = \Gamma_i = \int_{\hbar\omega_{min}}^{\hbar\omega_{max}} D_{odp,adp}[(f_{BE}+1) D^n(E+\hbar\omega) + f_{BE} D^n(E-\hbar\omega)]$$

Couple

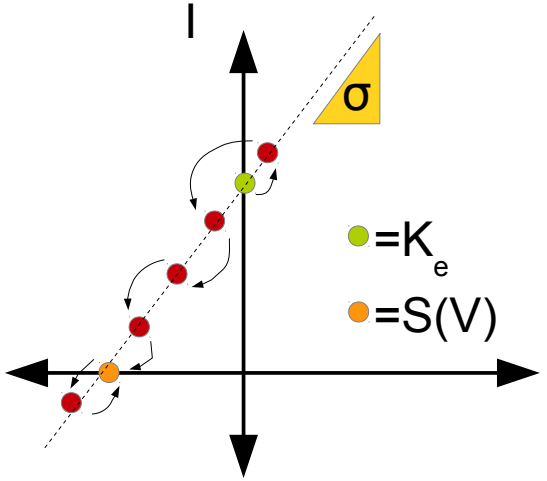
NEGF Thermal Model (Parabolic Atomistic Hamiltonian)

$$G = \omega^2 I - H + \Sigma_1 + \Sigma_2 + \Sigma_{ep} + \Sigma_{pp}$$

$$q = \int_0^{\omega_{max}} \frac{\hbar\omega}{(2\pi)^3} \Xi (f_2 - f_1)\, d\omega$$

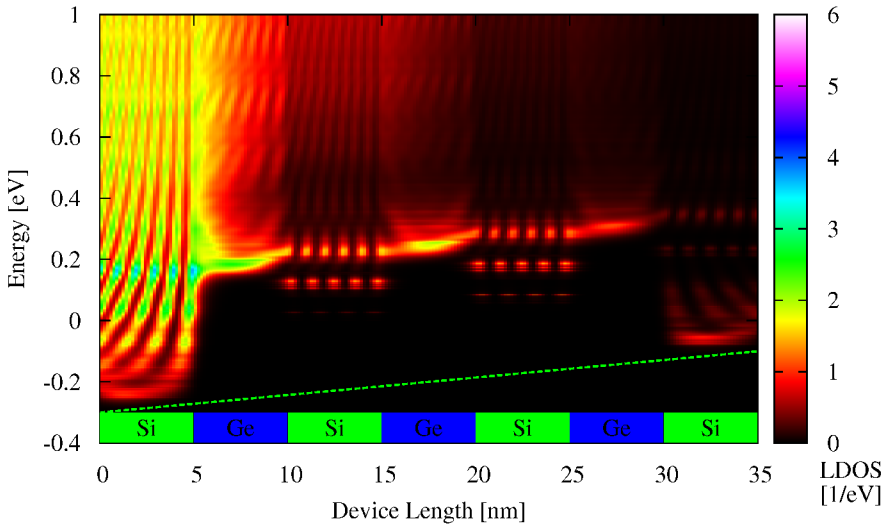
Phonon-Phonon Scattering Term

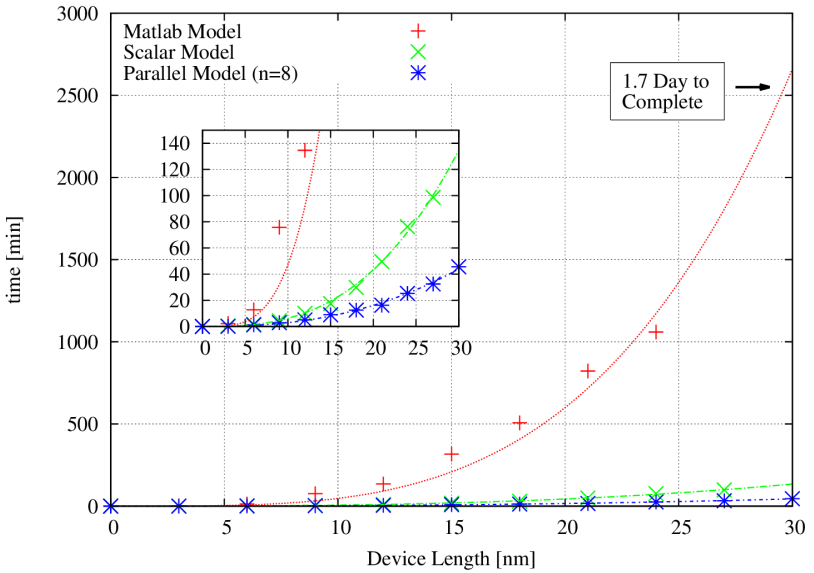
$$\Sigma_i f_s = \Gamma_i = \int_{\hbar\omega_{min}}^{\hbar\omega_{max}} f_o[(f_{BE}+1) D^n(E+\hbar\omega) + f_{BE} D^n(E-\hbar\omega)]$$

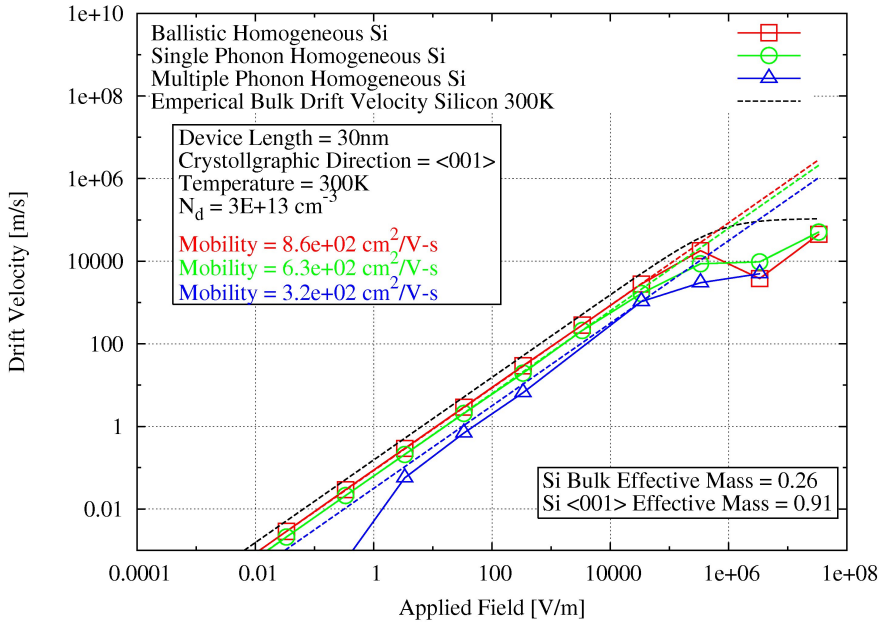


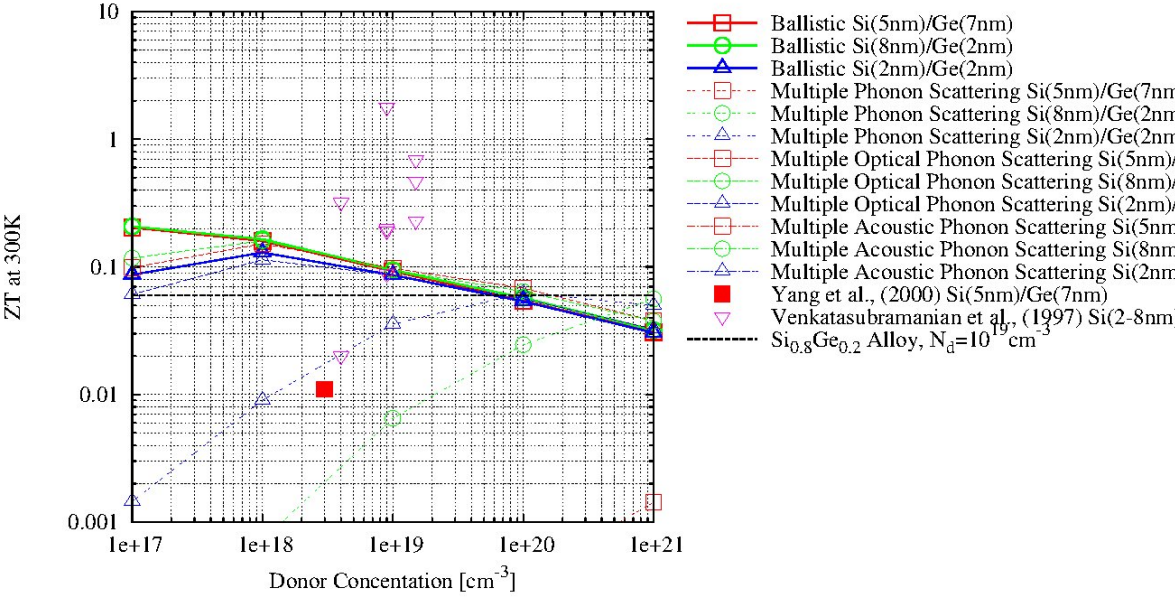
Iteratively solve for Seebeck Voltage

How do you calculate the electrical transport properties?









Back to 1970: Using AWK to parse large data files

AWK is a scripting language that is typically used for data extraction. Was very popular in the 1970s and 1980s. It can out of Bell Labs in 1970's and is named after three people: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan. Can parse data files in the tens of Gigabytes efficiently.

One Liners

```
# Read the first two columns of the data file and print to screen
awk '{x=$1;y=$2;print y,x}' christy_data.dat

# Read the first two columns and output the row number and two columns times two to a file
awk '{n=NR;x=$1*2.;y=$2*2.;print n,x,y}' christy_data.dat >output1.dat

# Read all columns but only print if column 1 is greater than 500. Print to screen
awk '{c1=$1;c2=$2;c3=$3;c4=$4;c5=$5;if(c1>500) {print c1 c2 c3 c4 c5}}' christy_data.dat
```

Multi-Liner

```
BEGIN { i = 1}
# Between these two brackets each line will be read similar to the one liners
{
  #read all values in column 3 into an array
  if(NR>1) { #skip first row
     output_N0[i] = $3
     i++ #increment
  }
}
#once all the lines have been read in the inputfile this will be executed
END{
  #you can print array to outputfile
  for(nn=1;nn<=i;nn++){
    print nn,output_N0[nn] > "output2.dat"
  }
}
```

Back to 1970: Using AWK to parse large data files

We can integrate awk into gnuplot to make plotting data quick and efficient.

```
#define variables
einf = 1.000
wpf = 1.34e16/(2*3.14159) #plasma frequency - fit
wtf = 6.2e13/(2*3.14159) #collision frequency – fit

#function
e1_fit(w) = einf – wpf**2/((w*10**12)**2+wtf**2)

#least squares fit data to function, skip first two rows, print col 1 and 4.
fit e1_fit(x) "<awk '{if(NR > 2){x=$1;y=$4;print x,y}}' ./christy_data.dat" using 1:2 via wpf,wtf

#plot data
plot "<awk '{if(NR > 2){x=$1;y=$4;print x,y}}' ./christy_data.dat" u 1:2 axis x1y1 title "Real Epsilon
({/Symbol e}_1) - Exp, Christy et al. 1972" w p ls 53 pt 1,\
     "<awk '{if(NR > 2){x=$1;y=$5;print x,y}}' ./christy_data.dat" u 1:2 axis x1y2 title "Imag Epsilon
({/Symbol e}_2) - Exp, Christy et al. 1972" w p ls 54 pt 2,\
      e1_fit(x) w l ls 74 axis x1y2 title "Imag Epsilon ({/Symbol e}_1) - Least Squares Fit to Exp"
```

Parallelization in 2min: Using OpenMP

OpenMP is a shared memory multiprocessing method for C, C++, and Fortran. To implement OpenMP all you have to do is a directive above loops. OpenMP can be used in a hybrid method with MPI to further increase parallel efficiency

set environmental variable

```
export OMP_NUM_THREADS=4
```

Original Loop

```
! Matxopt_mcopy - Matrix copy
!
subroutine Matxopt_dmcopy(Go,Gi,ns,nf)

   implicit none
   integer(kind=4) :: n
   integer(kind=4), intent(in) :: ns, nf
   real(kind=8), dimension(:), intent(in) :: Gi1,Gi2
   real(kind=8), dimension(:), intent(out) :: Go

   do n = 1, ns*nf
       Go(n) = Go(n)+Gi1(n)*Gi2(n)
   end do

end subroutine Matxopt_dmcopy
```

Parallized Loop

```
! Matxopt_mcopy - Matrix copy
!
subroutine Matxopt_dmcopy(Go,Gi,ns,nf)

   integer(kind=4) :: n, r
   integer(kind=4), intent(in) :: ns, nf
   real(kind=8), dimension(:,:), intent(in) :: Gi
   real(kind=8), dimension(:,:), intent(out) :: Go

!$OMP PARALLEL DO
   do n = 1, ns
      do r = 1, nf
        Go(n) = Go(n)+Gi1(n)*Gi2(n)
      end do
   end do
!$OMP END PARALLEL DO

end subroutine Matxopt_dmcopy
```

GPGPU Power: Writing OpenCL code

OpenCL is a platform independent means of writing code for both CPU and GPUs. OpenCL code will run on both NVIDIA and ATI GPUs.

CPU Code

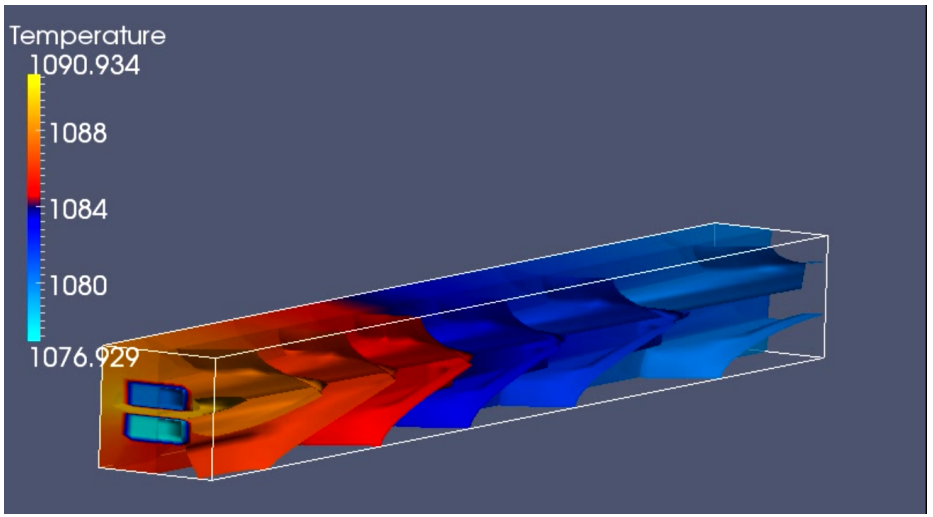
```
!..... CALCULATE BETA AND OMEGA
!
      BET=0.
      do K=2,NKM
        do I=2,NIM
          do J=2,NJM
            IJK=LK(K)+LI(I)+J
            BET=BET+RES(IJK)*RESO(IJK)
          end do
        end do
      end do
      OM=BET*GAM/(ALF*BETO+SMALL)
      BETO=BET
```

OpenCL Code

```
/CALCULATE BETA AND OMEGA
__kernel void clcgstab1D_cbo(int nxyz, int nxmax, int
nymax, int nzmax, int NIJ, int NJ, __global int* LK,
__global int* LI, __global double* BET,
                             __global double* RES, __global
double* RESO)
{

  int idx,bidx,bszx;
  int i;
  int ijk;

  idx = get_local_id(0); //thread id - i-dir
  bidx = get_group_id(0); //group id - i-dir
  bszx = get_local_size(0); //group size - i-dir

  i = bidx*bszx+idx;

  ijk = LK[i]; //must lie on grid

  if(ijk>=0) {
    BET[ijk] = RES[ijk]*RESO[ijk];
  }

}
```

Visualizing Your Data: Writing Data in VTK format

Free Viewer of VTK Format
- VisIT
- Paraview (preference)




```
write(19,103) '# vtk DataFile Version 2.0'
write(19,103) 'Field Emission Device - LDOS'
write(19,103) 'ASCII'
write(19,103) 'DATASET RECTILINEAR_GRID'
write(19,105) 'DIMENSIONS',Ndx,Ndy,NE

l = 0; w = 0

write(19,106) 'X_COORDINATES ',Ndx,' float'
do n=1,Ndx
  write(19,104) l !Points
  l = l + Gdx(n)
end do

write(19,106) 'Y_COORDINATES ',Ndy,' float'
do j=1,Ndy
  write(19,104) w !Points
  w = w + Gdy(j)
end do

Em = maxval(E)
write(19,106) 'Z_COORDINATES ',NE,' float'
do r=1,NE
  write(19,104) E(r)/Em*l !Points
end do

write(19,107) 'POINT_DATA', NE*Np
write(19,103) 'SCALARS Avail_States(LDOS) float 1'
write(19,103) 'LOOKUP_TABLE default'
do r=1, NE
   do k=1,Ndy
     do n=1,Ndx
     j = (k-1)*Ndx + n
       write(19,104) A(r,j)/(2.0*pi) !LDOS Contour Plot
     end do
   end do
end do
```

Questions?