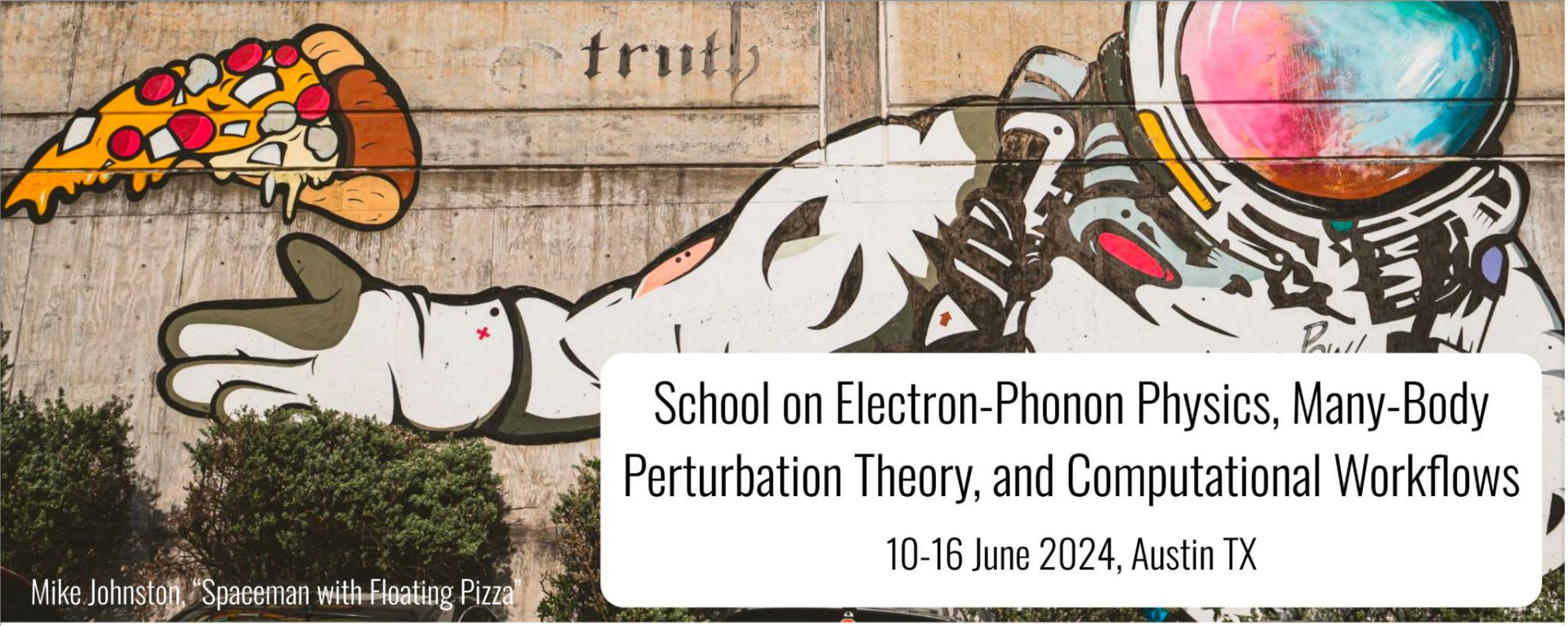


Mike Johnston, "Spaceman with Floating Pizza"

School on Electron-Phonon Physics, Many-Body Perturbation Theory, and Computational Workflows

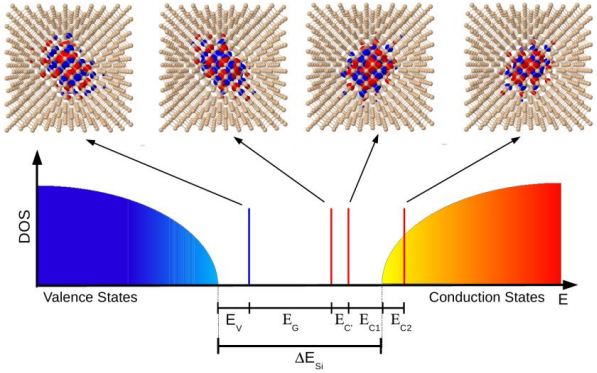
10-16 June 2024, Austin TX

U.S. DEPARTMENT OF ENERGY

NSF

TACC
TEXAS ADVANCED COMPUTING CENTER

ODEN



C2SEPEM

BerkeleyGW

EPW Summer School 2024

Hackathon Session

Mauro Del Ben, Zhenglu Li & Jack Deslippe (LBNL)

BerkeleyGW Overview

A massively parallel software package to study the excited state properties of electrons in materials via GW+BSE approaches and beyond

- Programming language: Mainly Fortran 2003 (over 100,000 lines of code)
- Parallelization:
  - Multi-node (MPI)
  - Multi-core (OpenMP)
  - GPU (OpenACC/OpenMP-target)
- Libraries:
  - Required: BLAS, LAPACK, FFTW
  - Optional (recommended): ScalaPACK, ELPA, HDF5
  - Additional: PRIMME, cuBLAS, cuFFT

BerkeleyGW

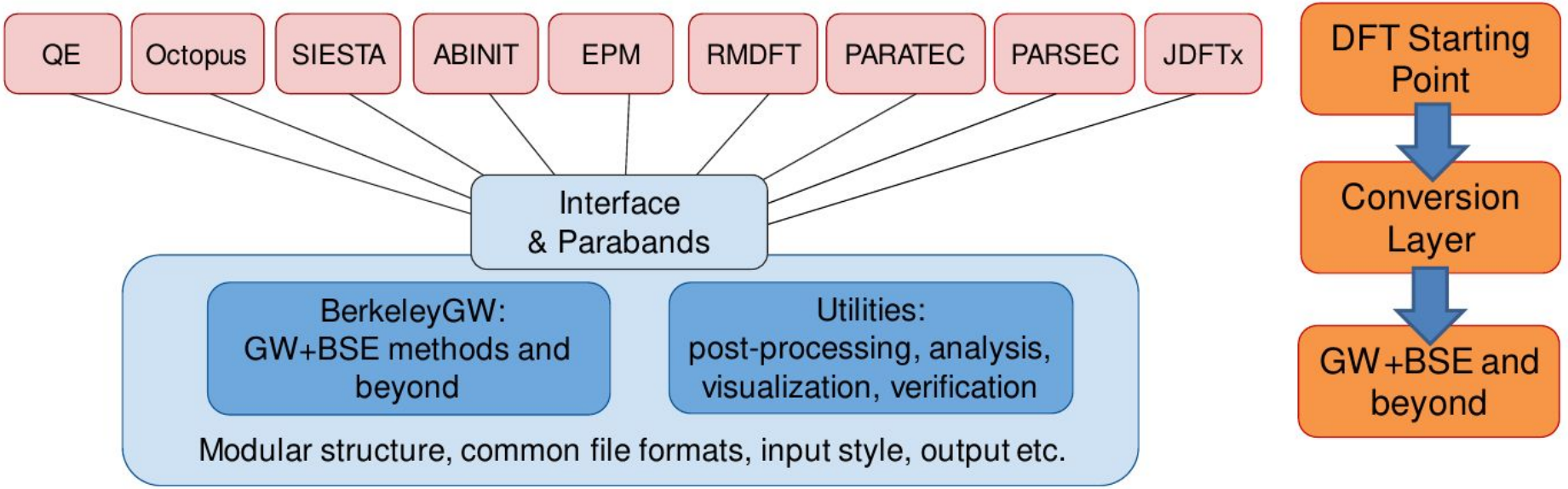
Overview

A massively parallel software package to study the excited state properties of electrons in materials via GW+BSE approaches and beyond

- Basic computational motifs implemented in BerkeleyGW:
  - Large distributed matrix-multiplication over short and fat matrices
  - Large distributed linear algebra: LU decomposition, matrix inversion, eigen-decomposition, etc...
  - Many, non-distributed fast Fourier transformations (FFT)
  - Dimensionality reduction and low-rank approximations
  - Parallel I/O of rank-2 -3 and -4 tensors

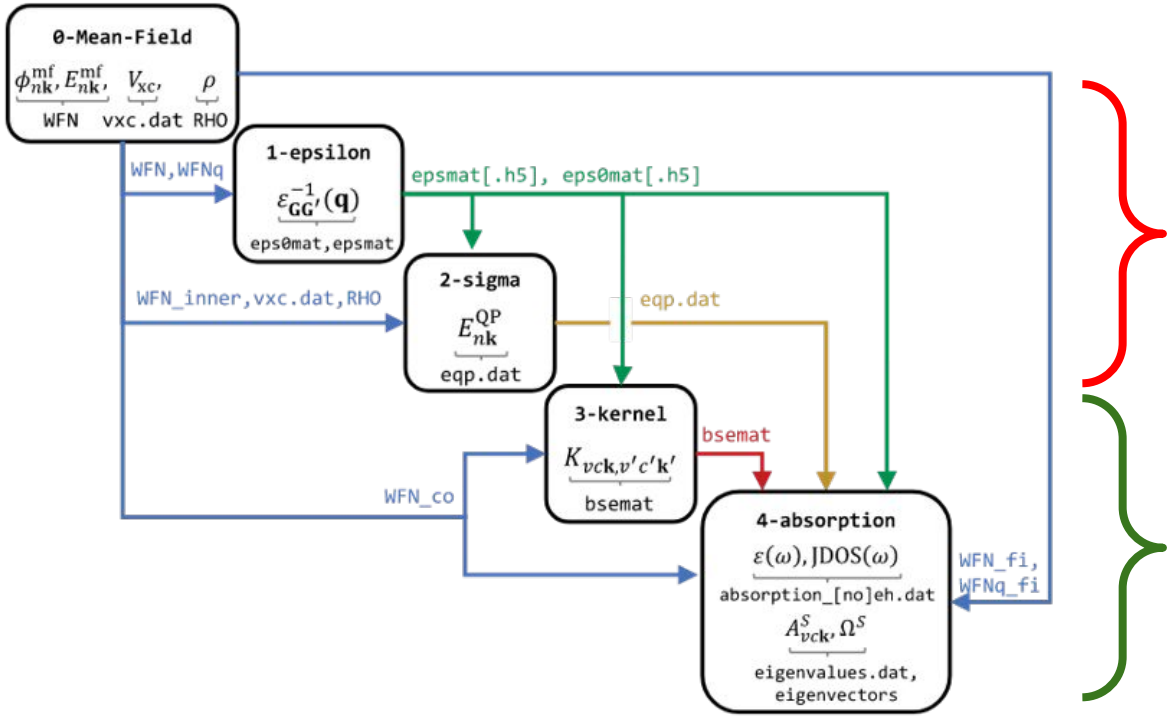
BerkeleyGW

Software Design and Vision



Developments focused on advanced MBPT methods to study the excited state properties of electrons in materials.

BerkeleyGW Workflow

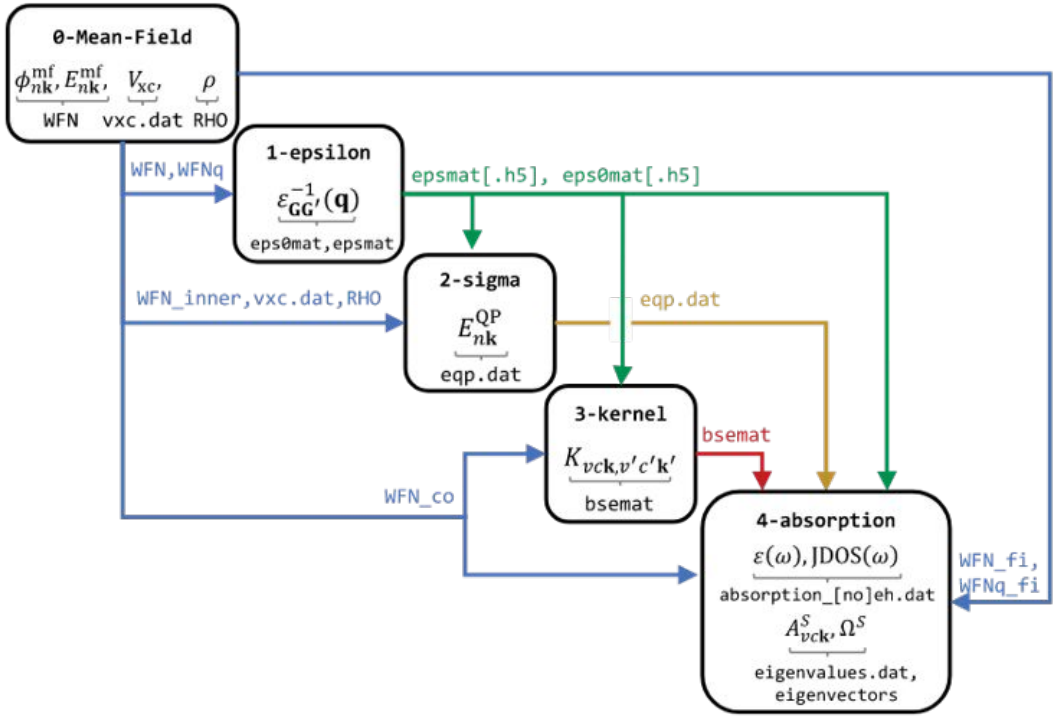


GW Workflow

BSE Workflow

http://manual.berkeleygw.org/4.0/overview-workflow/

BerkeleyGW Workflow



Epsilon: Generate the dielectric function and its frequency dependence

Sigma: Solve Dyson's equation for quasiparticle energies

Kernel: Compute BSE kernel matrix elements on a coarse k-point grid

Absorption: Interpolate BSE kernel matrix elements onto a fine k-point grid, diagonalize the BSE Hamiltonian, and compute optical absorption spectrum

http://manual.berkeleygw.org/4.0/overview-workflow/

BerkeleyGW

The Very Basic Operations

- **Epsilon**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + LU decomposition / inversion
- **Sigma**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + Tensor like reduction
- **Kernel**: FFTs + Matrix Multiplications (ZGEMM/DGEMM)
- **Absorption**: Interpolation techniques + Diagonalization

BerkeleyGW

The Very Basic Operations

- **Epsilon**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + LU decomposition / inversion
- **Sigma**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + Tensor like reduction
- **Kernel**: FFTs + Matrix Multiplications (ZGEMM/DGEMM)
- **Absorption**: Interpolation techniques + Diagonalization

```
login2.frontera(1002)$ ls
arch.mk   Common          documentation  flavor_cplx.mk  frontera_gpu.mk  LOGO          NonLinearOptics  Sigma     Visual
bin       config          Epsilon        flavor.mk       library          Makefile      PlotXct          TDGW      xctph
BSE       Copyright.txt   examples       flavor_real.mk  license.txt      MeanField     README.md        testsuite
login2.frontera(1003)$
```

BerkeleyGW

The Very Basic Operations

- **Epsilon**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + LU decomposition / inversion
- **Sigma**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + Tensor like reduction
- **Kernel**: FFTs + Matrix Multiplications (ZGEMM/DGEMM)
- **Absorption**: Interpolation techniques + Diagonalization

```
login2.frontera(1002)$ ls
arch.mk   Common          documentation  flavor_cplx.mk  frontera_gpu.mk  LOGO          NonLinearOptics  Sigma     Visual
bin       config          Epsilon        flavor.mk       library          Makefile      PlotXct          TDGW      xctph
BSE       Copyright.txt   examples       flavor_real.mk  license.txt      MeanField     README.md        testsuite
login2.frontera(1003)$
```

BerkeleyGW

The Very Basic Operations

BGW_Hackathon_EPW24/EPSILON-parallel/

- **Epsilon**: FFTs + **Matrix Multiplications** (ZGEMM/DGEMM) + LU

BGW_Hackathon_EPW24/SIGMA-serial/

- **Sigma**: FFTs + Matrix Multiplications (ZGEMM/DGEMM) + **Tensor like reduction**

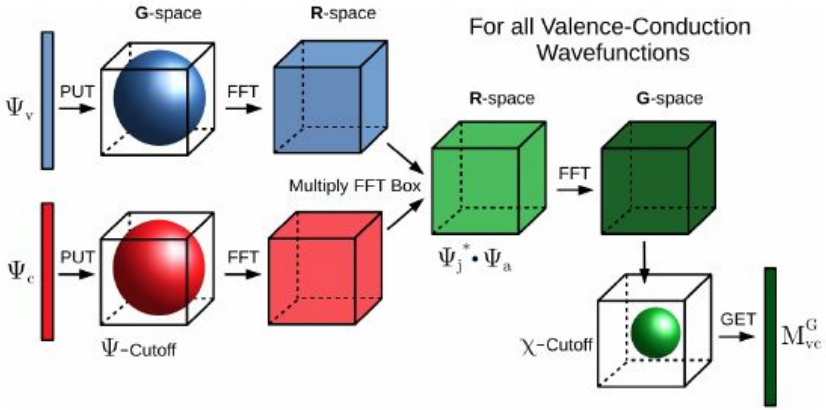
BGW_Hackathon_EPW24/EIGENSOLVERS-parallel/

- **Absorption**: Interpolation techniques + **Diagonalization**

```
login2.frontera(1002)$ ls
arch.mk   Common          documentation  flavor_cplx.mk  frontera_gpu.mk  LOGO          NonLinearOptics  Sigma     Visual
bin       config          Epsilon        flavor.mk       library          Makefile      PlotXct          TDGW      xctph
BSE       Copyright.txt   examples       flavor_real.mk  license.txt      MeanField     README.md        testsuite
login2.frontera(1003)$
```

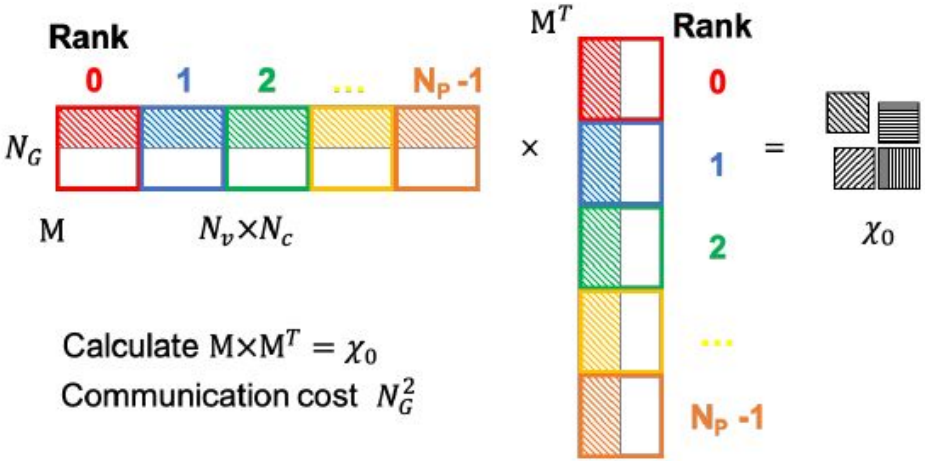
Epsilon: Computational Kernels

MTXEL



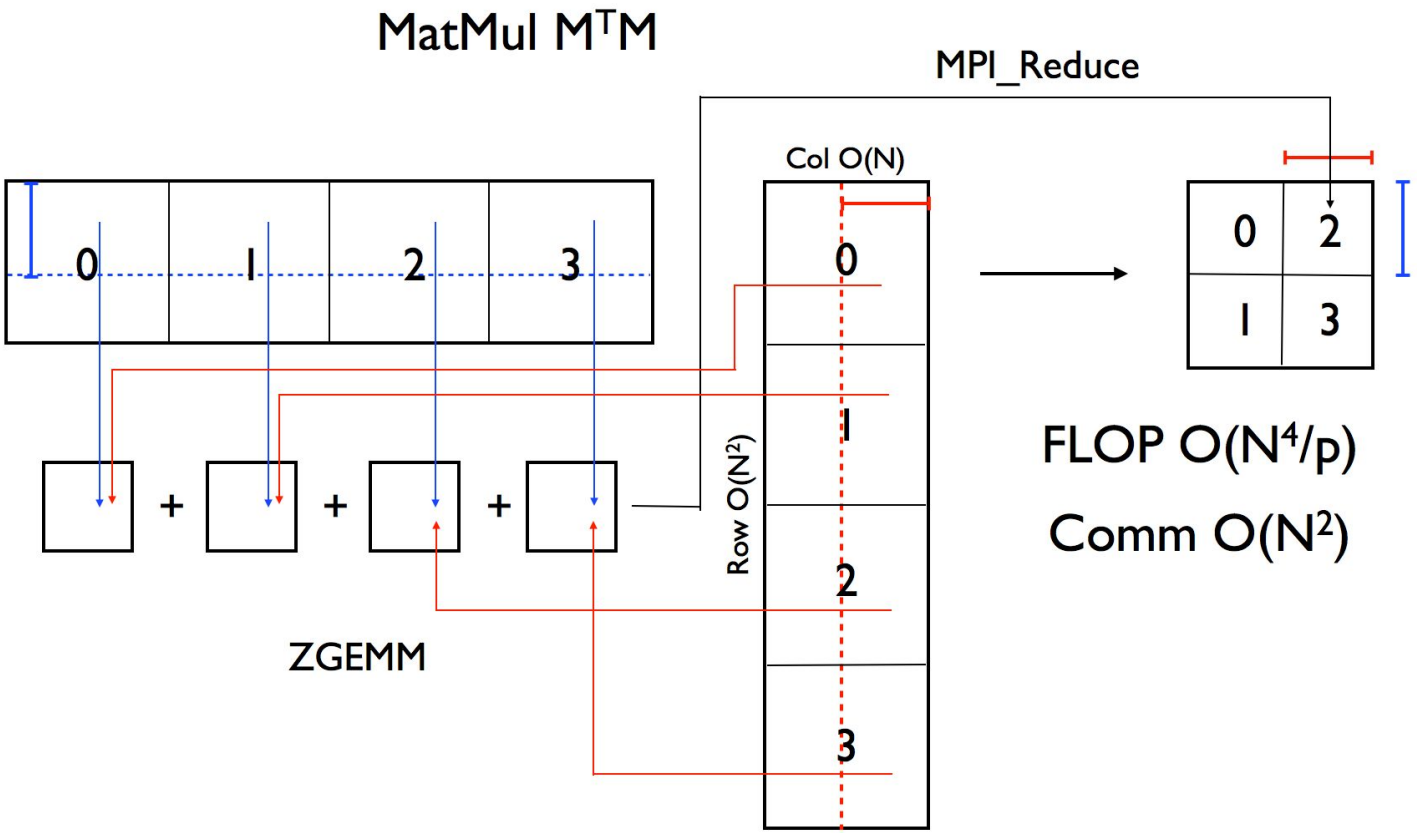
FFT based kernel

CHI-0



ZGEMM based kernel

Epsilon CHI-0: Parallel Implementation Collectives

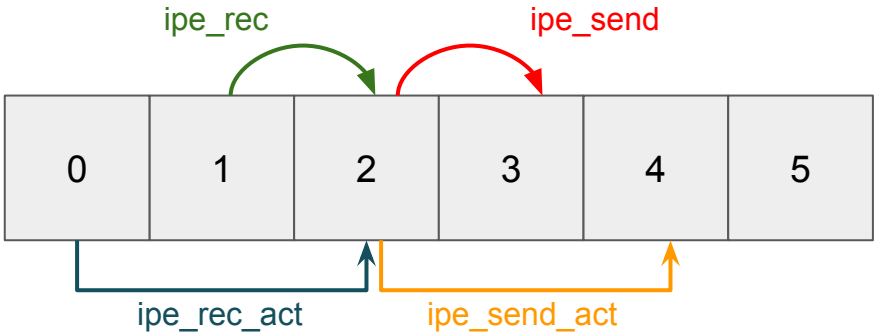


Epsilon CHI-0: Parallel Implementation Point to Point

Non-Blocking Cyclic Communication:

- Non-Blocking Isend / Irecv
- Async MemCopy (GPU)
- Overlap Communication (CPU) Computation (GPU/CPU)
- Communication only myrank+1 and myrank-1

NBCy: Task#2, second cycle



NBCy Pseudo Algorithm:

Define send proc: ipe_send = my_ipe + 1
Define receive proc: ipe_rec = my_ipe - 1
Define local buffer chi_temp_send(my_ipe)

do ipe = 1 , nprocs
  Define actual sending proc: ipe_send_act = my_ipe + ipe
  Define actual receiving proc: ipe_rec_act = my_ipe - ipe

  Define local buffer chi_temp_rec(ipe_rec_act)
  Non-blocking receive from ipe_rec: chi_temp_rec(ipe_rec_act)
  Non-blocking send to ipe_send: chi_temp_send

  Do ZGEMM for chi_local(ipe_rec_act)

  WAIT to receive chi_temp_rec(ipe_rec_act) from ipe_rec
  ACCUMULATE: chi_temp_rec = chi_temp_rec + chi_local
  WAIT till chi_temp_send has been sent to ipe_send
  SWAP chi_temp_send with chi_temp_rec
end do

Epsilon CHI-0: `EPSILON-parallel` miniapp

This miniapp simulate the CHI-0 kernel running at scale using only few computational resources, can be used to:

- Estimate required memory
- Minimum number of MPI task required
- Time to solution for the full execution
- Assess performance of the ZGEMM library and MPI communication

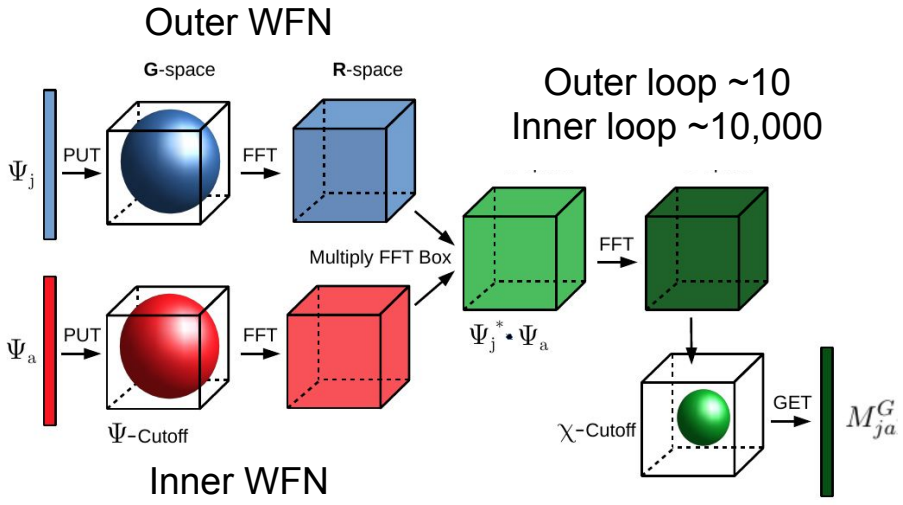
`input_Si1000_12Ry`

```
 1      ! Nspin x Nkpoint
1996    ! Nvalence bands
29346   ! Nbands total
94617   ! nmtx (distributed matrix size)
9216    ! nproc simulated
96      ! Nprow in scalapack layout (Nprow * Npcol  = Nproc)
96      ! Npcol in scalapack layout (Nprow * Npcol  = Nproc)
25      ! Number of repetition eventually will be equal to Nproc simulated
```

Sigma: Computational Kernels

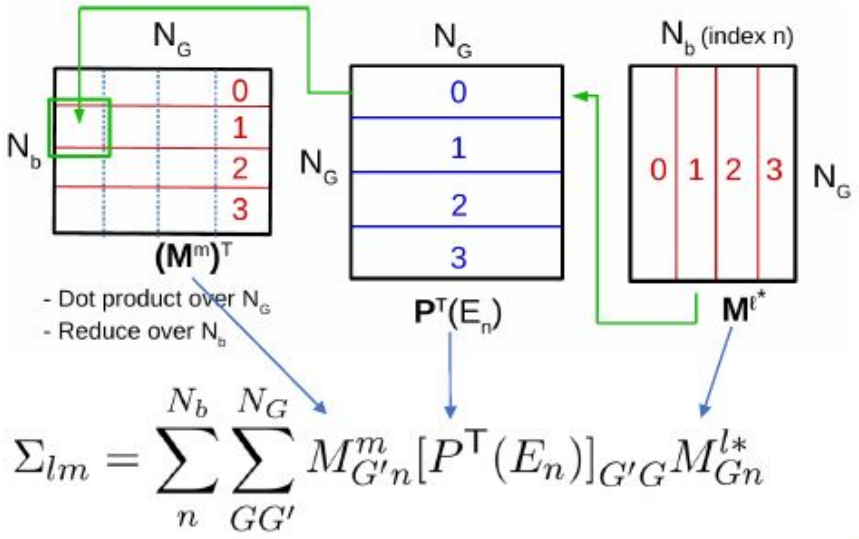
SIGMA-MTEXL (<5%)

Matrix elements via Fast-Fourier Transformation (FFT)



SIGMA-GPP (95%>)

Tensor-like reduction across different matrices with a complex matrix-vector interdependence



$$\Sigma_{lm} = \sum_{n}^{N_b} \sum_{GG'}^{N_G} M^{m}_{G'n} [P^{\mathsf{T}}(E_n)]_{G'G} M^{l*}_{Gn}$$

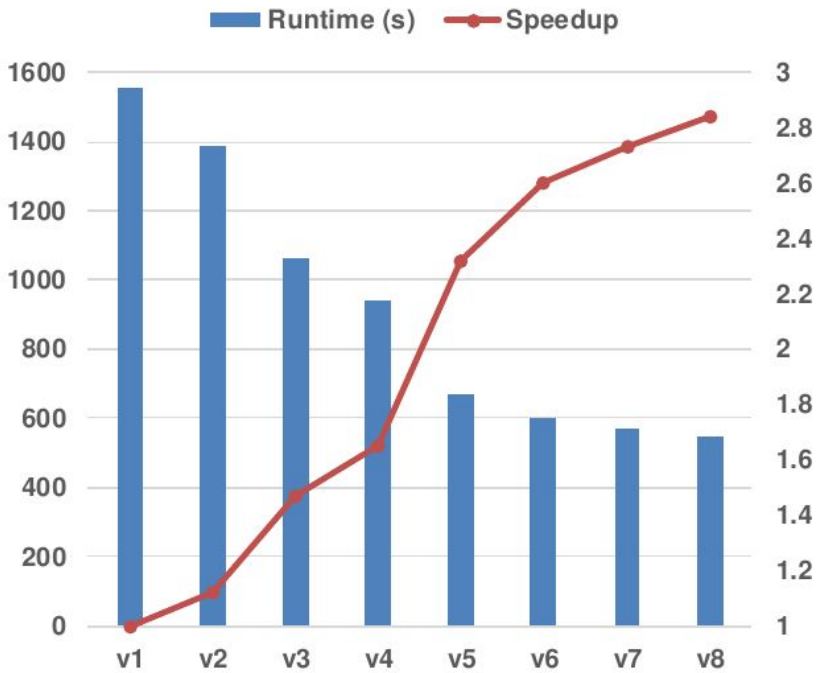
The Sigma GPP Kernel: Computational Characteristics

```
# pseudo code per invocation
for band = 1, nbands # O(1k)
 for igp = 1, ngpown # O(10k)
  for ig = 1, ncouls # O(100k)
   for iw = 1, nw    # small
    Complex number arithmetic
Reduce to arrays[iw]
```

- Tensor contraction
  - Bandwidth bound
- Reduction of 10^{12} numbers
  - Shared mem for partial sums
- Double complex numbers
  - High register usage
- Multiple multi-dim arrays
  - Memory access pattern
- Long-latency operations
  - Divisions, square roots

The Sigma GPP Kernel: Optimization Path (GPU)

1. Baseline*
2. Replace divides with reciprocals
3. Replace square roots with power of 2
4. Replace divides and square roots
5. Loop re-ordering
6. Further increase occupancy
7. Cache blocking
8. Add more arrays to shared memory



*Collapse 3 of the other loops

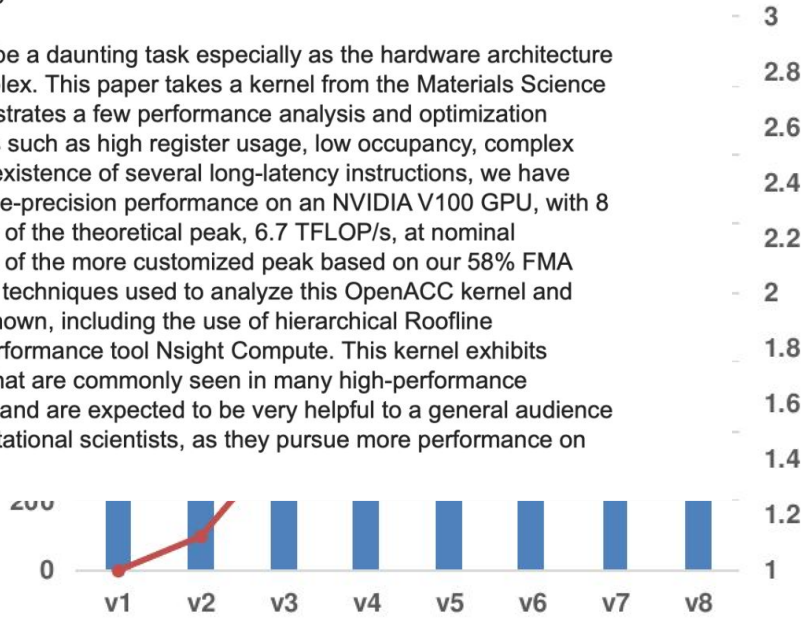
Th

Charlene J. Yang

8 steps to 3.7 tflop/s on nvidia v100 gpu: Roofline analysis and other tricks

Authors	Charlene Yang
Publication date	2020/8/26
Journal	arXiv preprint arXiv:2008.11326
Description	Performance optimization can be a daunting task especially as the hardware architecture becomes more and more complex. This paper takes a kernel from the Materials Science code BerkeleyGW, and demonstrates a few performance analysis and optimization techniques. Despite challenges such as high register usage, low occupancy, complex data access patterns, and the existence of several long-latency instructions, we have achieved 3.7 TFLOP/s of double-precision performance on an NVIDIA V100 GPU, with 8 optimization steps. This is 55% of the theoretical peak, 6.7 TFLOP/s, at nominal frequency 1312 MHz, and 70% of the more customized peak based on our 58% FMA ratio, 5.3 TFLOP/s. An array of techniques used to analyze this OpenACC kernel and optimize its performance are shown, including the use of hierarchical Roofline performance model and the performance tool Nsight Compute. This kernel exhibits computational characteristics that are commonly seen in many high-performance computing (HPC) applications, and are expected to be very helpful to a general audience of HPC developers and computational scientists, as they pursue more performance on NVIDIA GPUs.

1.
2.
3.
4.
5.
6.
7.
8. Add more arrays to shared memory



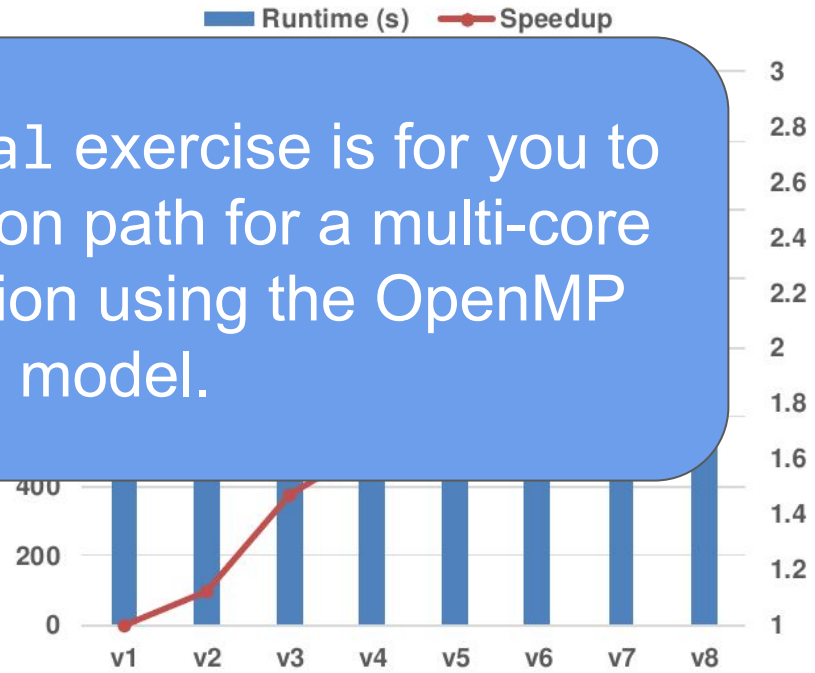
*Collapse 3 of the other loops

The Sigma GPP Kernel: Optimization Path (GPU)

1.
2.
3.
4.
5.
6.
7. Cache blocking
8. Add more arrays to shared memory

*Collapse 3 of the other loops

The Goal of the `SIGMA-serial` exercise is for you to replicate the same optimization path for a multi-core CPU and GPU Implementation using the OpenMP programming model.



The Sigma GPP Kernel: `SIGMA-serial` miniapp

This miniapp simulate the core computation in the evaluation of self-energy matrix elements using the generalized plasmon pole model (GPP). The GPP kernel runs in serial but simulate the core computation for each MPI rank in a parallel run.

```
export OMP_NUM_THREADS=1
ibrun -np 1 ./gppKerFort.ex 6000 500 25000 1024

 Where the input parameter are:
  *  6000   ! Number of bands to sum over
  *   500   ! Number of valence bands
  * 25000   ! Number of G-vectors up to the screened coulomb cutoff
  *  1024   ! Number of MPI tasks simulated
```

- Implement multi-core parallelization for the kernel using the OpenMP
- Measure parallel efficiency by increasing `OMP_NUM_THREADS`
- Use OpenMP-target to implement GPU offload

The Sigma GPP Kernel: `SIGMA-serial` miniapp

This miniapp simulate the core computation in the evaluation of self-energy matrix elements using the generalized plasmon pole model (GPP). The GPP kernel runs in serial but simulate the core computation for each MP

```
expo
ibru

 Whe
  *
  *
  * 2                                                    utoff
  *   1024    ! Number of MPI                            lated
```

The fastest implementation will win the Donkey prize!!

- Implement multi-core paralle zatio for the kernel using the OpenMP
- Measure parallel efficiency by incre sing `OMP_NUM_THREADS`
- Use OpenMP-target to implement GPU offload

Absorption: Diagonalization

Excitation energy, exiton wavefunctions and absorption spectrum are obtained as solution of the eigenvalue problem associated to the BSE Hamiltonian

Fine **k**-grid: $\left(E^{\mathrm{QP}}_{c\mathbf{k}} - E^{\mathrm{QP}}_{v\mathbf{k}}\right) A^{S}_{vc\mathbf{k}} + \sum_{v'c'\mathbf{k}'} \langle vc\mathbf{k}| K^{\mathrm{eh}} |v'c'\mathbf{k}'\rangle A^{S}_{v'c'\mathbf{k}'} = \Omega^{S} A^{S}_{vc\mathbf{k}}$

Interpolated E^{QP}

Interpolated kernel matrix elements

- Direct Solver (ScalaPACK, ELPA) $O(N^6)$
  Exact diagonalization, compute all exciton states
- Iterative Solvers (PRIMME)
  Exact diagonalization, compute selected lowest exciton states
- Haydock-Recursion Method (haydock.cplx.x) $O(N^4)$
  Computes only the absorption spectra

Absorption: Diagonalization of the BSE Hamiltonian

N	Nodes	Time
17000	4	10 s
60000	16	10 min
100000	64	15 min
150000	256	17 min
200000	512	25 min
250000	512	38 min
320000	512	68 min
416000	2560	80 min

Matrix Size	Number on Nodes	GPU Support	Time for Diagonalization (s)
19,381	1	No	215
19,381	1	Yes	83.2
65,117	16	Yes	135
155,331	128	Yes	279

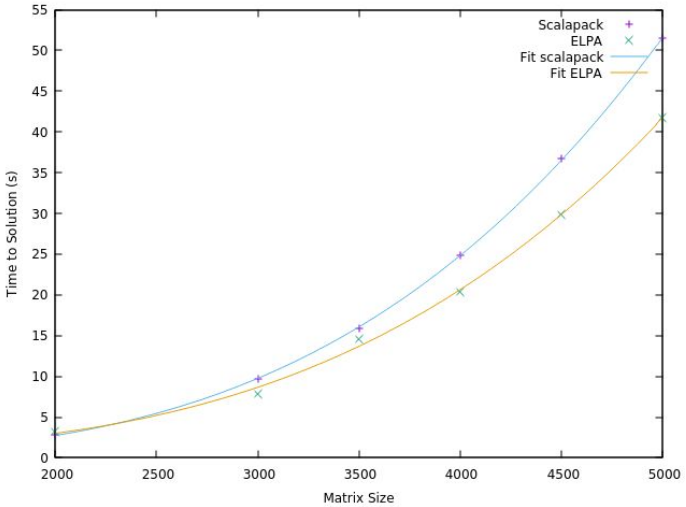
The BSE matrix size is **nk * nv * nc** for the fine k-point grid

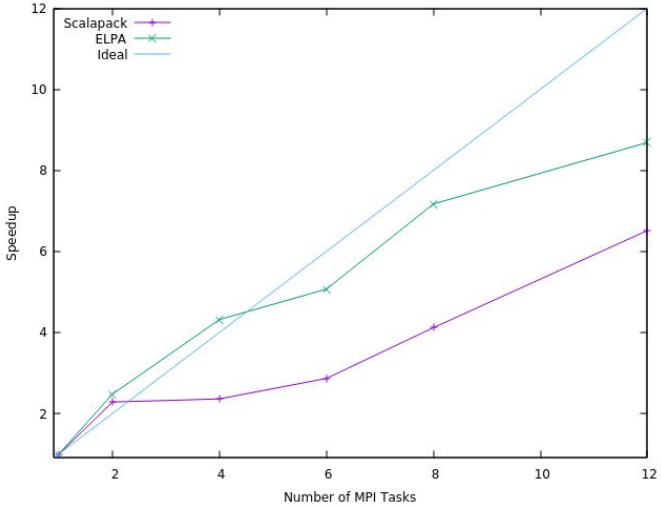
Absorption: the `EIGENSOLVERS-parallel` miniapp

This miniapp simulate the diagonalization step in absorption module using different libraries, allows to:

- Parallel scalability and weak scaling
- Time to solution for the full execution
- Assess performance of the various libraries on different architectures/HPC centers

In this exercise you'll measure the scaling of computational cost wrt matrix size (using O notation) and parallel scalability for the Scalapack (pzheevd) and ELPA libraries.





How to run:

- Login to Frontera: `ssh -X USERNAME@frontera.tacc.utexas.edu`

- Copy the tutorial material

```
cd $SCRATCH
cp /work2/05193/sabyadk/stampede3/EPWSchool2024/tutorials/Sun.1.BGW_Hack.DelBen.tar .
tar -xvf Sun.1.BGW_Hack.DelBen.tar
```

- Enter each of the folder and follow the instructions in `README.md`
  - `BGW_Hackathon_EPW24/EPSILON-parallel/` (Epsilon CHI-0 exercise)
  - `BGW_Hackathon_EPW24/EIGENSOLVERS-parallel/` (Diagonalization exercise)
  - `BGW_Hackathon_EPW24/SIGMA-serial/` (Sigma GPP exercise)
  - `BGW_Hackathon_EPW24/wannier-interpolation-GW/` (Wannier interpolation exercise)

Wannier interpolation of GW band structure

- We will use Wannier90 to interpolate the GW band structure

Wannier interpolation of band structure

Purpose: We will use Wannier functions to interpolate GW band structures, which is a more general method than inteqp distributed in BerkeleyGW. Because inteqp only does insulators and graphene, not metals. Wannier interpolation with BerkeleyGW can do metal band structures.

- cd $SCRATCH/BGW_Hackathon_EPW24/
- cd wannier-interpolation-GW
- cd 1-scf/
- ./01-calculate_scf.run

Wannier interpolation of band structure

- cd ../2.1-wfn-wannier
- Take a look at the bands.in file, the KPOINTS card should be missing. We will use Wannier90's kmesh.pl to generate a full k-grid
- $W90PATH/utility/kmesh.pl 4 4 4 >> bands.in
-
- ./01-cp-files.sh
- ./02-calculate_wfn.run
- Open Si.win file and see how the projections, atom positions, lattice vectors, mp_grid been set up properly and consistently with bands.in
- ./03-wannier90.run
- Open the Si.wout file to check the disentanglement outcome and the final spread of the Wannier functions

Wannier interpolation of band structure

- cd ../2.2-wfn-sigma/
- ./01-cp-files.sh
- ./02-calculate_wfn.run
- ./03-convert_wfn.run

- cd ../2.3-wfnq-sigma/
- ./01-cp-files.sh
- ./02-calculate_wfn.run
- ./03-convert_wfn.run

- cd ../3-epsilon/
- ./01-link-files.sh
- ./02-run_epsilon.run

Wannier interpolation of band structure

- cd ../4.1-g0w0/
- ./01-link-files.sh
- ./02-run_sigma.run **(READ BELOW!!! ;)**
- The above script runs the Sigma calculation in the background
- This is because this calculation can be long ~30 mins, since we are computing the full k-BZ! (This can be relatively easily avoided, and will be left as a Hackathon exercise/homework. See last slide)
- To track the progress: tail -f sigma.out
- You can stop tracking anytime with: Ctrl+C, and the Sigma calculation will continue running in the background
- If you feel the run is too long, you can terminate your sigma run and use what we have prepared the output:
  - cp sigma.out.ref sigma.out
  - cp sigma_hp.out.ref sigma_hp.out

Wannier interpolation of band structure

- ./03-run-sig2wan.run
- Open the generated file Si.eqp1.eig, this is the GW eqp1 energies written in the Wannier90 .eig file format

- cd ../4.2-g0w0-wannier/
- Take a look at the file 01-link-files.sh, note the GW eigenvalues will be linked
- ./01-link-files.sh
- Restart Wannier90 without redoing the Wannierization. Restart will be done with the checkpoint file .chk
- ./02-wannier90.run
- Interpolated G0W0 band structure is generated

Wannier interpolation of band structure

- Comments: It is straightforward to get rid of the full-BZ sigma calculation, but just use irreducible BZ
- In that case, one need a script to unfold the k-BZ and write a new sigma_hp.log file in the full k-BZ
- This will be left as a task for Hackathon or Homework

Wannier interpolation of band structure

- Wannier interpolation works for both insulators and metals

  (inteqp only for insulators)