Low-scaling *GW* in ABINIT (GWR)

M. Giantomassi

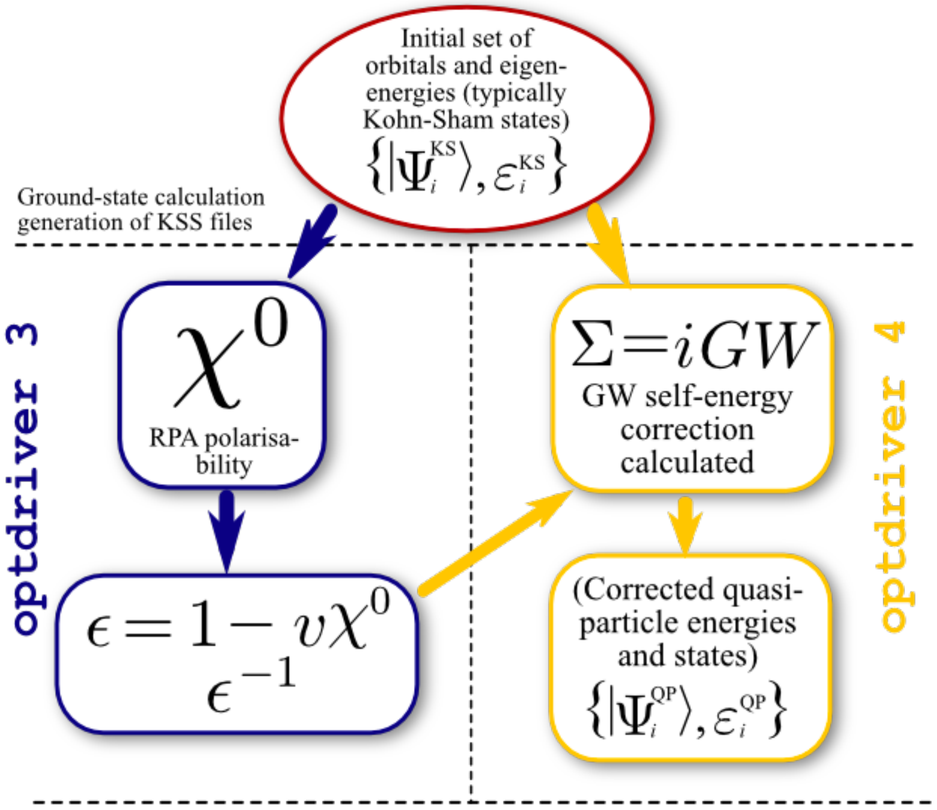
Université Catholique de Louvain
Louvain-la-Neuve, Belgium

ABIDEV 2024
11th International ABINIT Developer Workshop
Saint-Paulin, Québec, Canada, June 30th – July 4th, 2024

The *GW* implementation of ABINIT

(quartic scaling version)

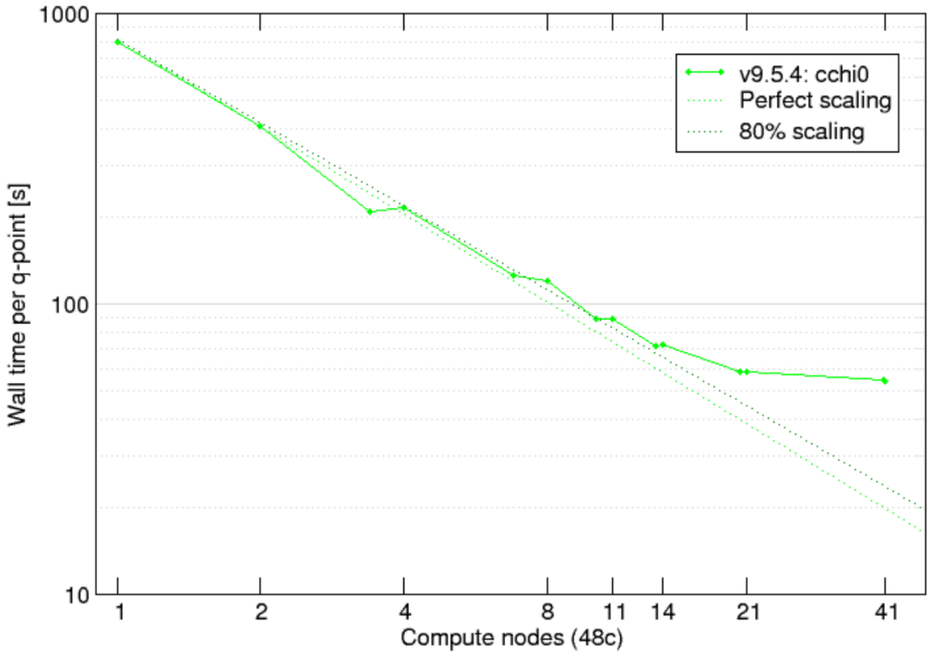
- Formalism in **G**- and ω-space (real-axis)
- Norm-conserving pseudos *(recommended)* and PAW
- Different approximations for the self-energy: *HF, COHSEX, GW*
- Different integrations techniques for Σ:
  - four different plasmon-pole models: FAST but APPROXIMATE
  - countour-deformation (**CD**): ACCURATE but SLOW
  - Analytic continuation + Pade'
- Different levels of self-consistency: G^0W^0, energy-only, qp-*GW*, full *GW*
- MPI-algorithm with *distributed* wavefunctions
- OpenMP threads for low-level loops , BLAS and FFTs
- Steps are connected via files and input variables: **getwfk** or **getwfk_filepath, getscr_filepath …**

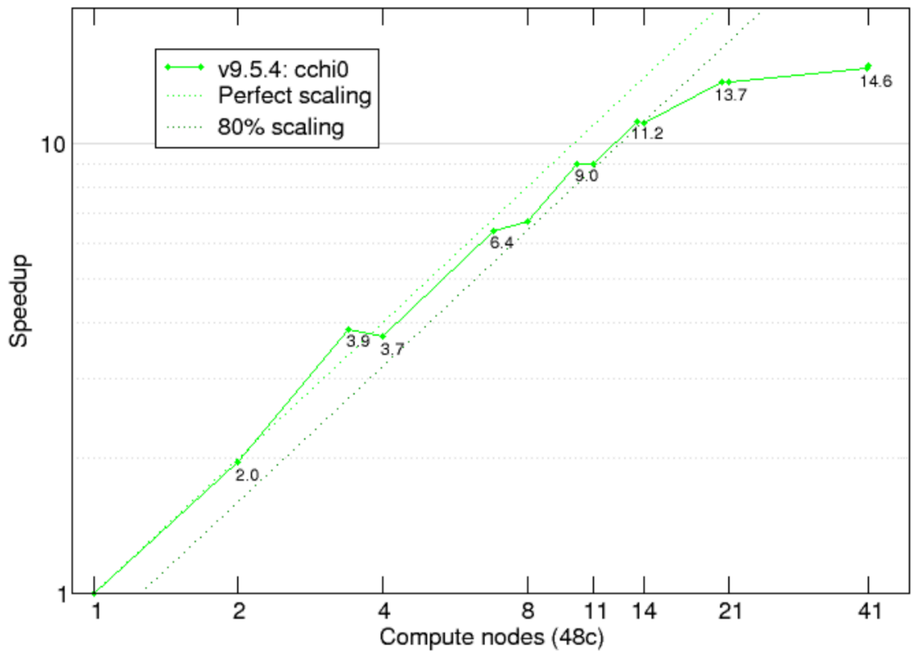


Limitations of the standard *GW* code

- **Quartic scaling** in *natom*
- **Quadratic** in the number of **k**-points in the BZ
- Memory for $W(\mathbf{g}, \mathbf{g}', \mathbf{q}, \omega)$ does not scale with MPI procs (big limitation when computing Σ)
- Only two MPI levels (*nband* and *nsppol*). Decent scalability but **not exascale-ready**:

https://gitlab.pop-coe.eu/documents/reports/-/raw/master/POP2-AR-157-Abinit.pdf



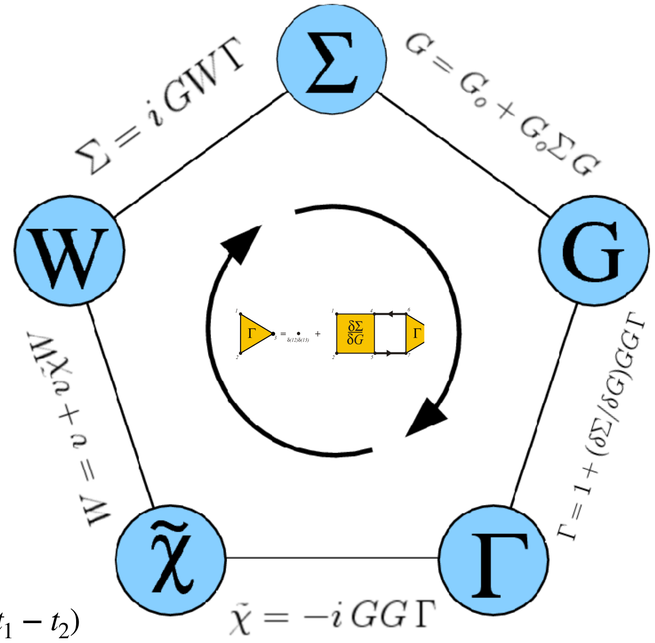


- Screening (step 3) of Zr2Y2O7/C1 testcase on MN4 “highmem” compute nodes
- 80% scaling efficiency sustained to 14 nodes (652 MPI processes)
- Fully-loaded nodes (with 48 MPI processes per node) slower than partially-loaded nodes leaving cores unused

GW in imaginary time with supercells

ABINIT GWR code (released in v10)

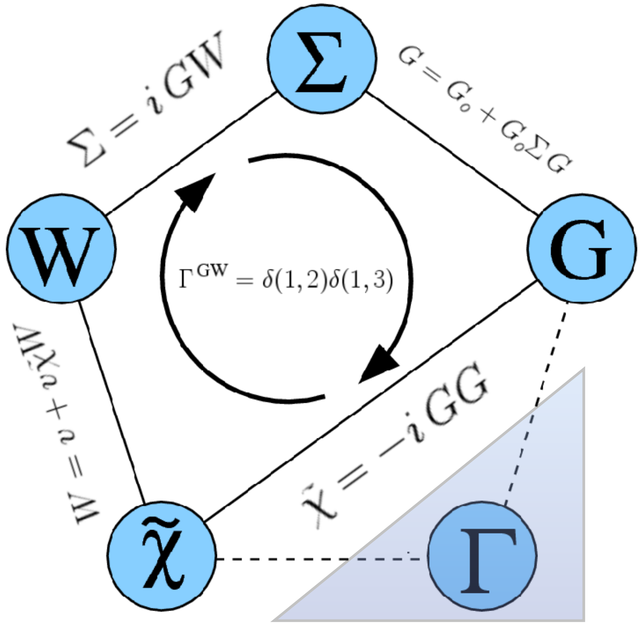
Hedin's equations



$(1) \equiv (\mathbf{r_1}, t_1)$

$\delta(12) = \delta(\mathbf{r_1} - \mathbf{r_2})\,\delta(t_1 - t_2)$

The GW approximation



GW with supercells and imaginary-axis

PHYSICAL REVIEW B 90, 054115 (2014)

In brief:

- Work with the analytic continuation of Hedin's equations on the image. axis: $(t \to i\tau, \omega \to i\omega)$
- Avoid convolutions by working in the most natural space *e.g.:*

  - $\chi(\mathbf{r}, \mathbf{R}', i\tau) = G(\mathbf{r}, \mathbf{R}', i\tau)\, G^*(\mathbf{R}', \mathbf{r}, -i\tau)$

  - $W_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) = v_{\mathbf{k}}(\mathbf{g}, \mathbf{g}')\epsilon_{\mathbf{k}}^{-1}(\mathbf{g}, \mathbf{g}', i\omega)$

Notations: $\mathbf{r}$ spans the unit cell, $\mathbf{R} = \mathbf{r} + \mathbf{L}$ spans the supercell

- Use FFTs to go from the $\mathbf{R} = \mathbf{r} + \mathbf{L}$ supercell to $\mathbf{G} = \mathbf{k} + \mathbf{g}$ and viceversa
- Sample the imaginary axis with minimax meshes $\{\omega_k\}, \{\tau_j\}$ that minimize the maximum error in the MP2 energy for given number of points N and $R = \dfrac{\Delta_{\text{max}}}{\Delta_{\text{gap}}}$
- Use inhomogeneous sine/cosine transforms for $i\omega_k \leftrightarrow i\tau_j$

Precomputed weights

$$\chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega_k) = \sum_{j=1}^{N} \gamma_{kj} \cos(\omega_k \tau_j) \chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\tau_j)$$

GWR code in a nutshell

- *optdriver* 6 to activate the GWR driver
- *gwr_task* specifies the task to perform:
  - "HDIAGO" for direct diagonalization with scalapack followed by WFK output
  - "G0W0" for one-shot method
  - "EGEW", "EGW0", "G0EW" for eigenvalue-only self-consistency
  - "RPA_ENERGY" for E_c energy with automatic extrapolation for *npweps* $\rightarrow \infty$
  - "CC4S" to produce matrix elements required by CC4S (Coupled cluster for solids)
- External files required:

  1) **DEN** file with GS density (required for all tasks)

  2) **WFK** file with empty states (only for *GW/RPA tasks*)
- **Scalapack required** in all *gwr_tasks*
- G and W are computed in the same run and stored in memory (no *getscr** variables)
- Self-consistent iterations are performed in the same dataset (no *getqps* variable)
- Automatic parallelization:
  - HDIAGO uses MPI pools to distribute **k**/spins and $H_{\mathbf{gg}'}$ matrix with scalapack
  - The other *gwr_tasks* employ a 4D MPI Cartesian grid (**g/r**, **k**-points, minimax mesh, spin)

GWR input variables

New input variables specific to GWR:

- *gwr_task* instead of *gwcalctyp*
- *gwr_ntau:* number of points in the minimax mesh (GreenX library)
- *gwr_boxcutmin:* defines the FFT mesh for G from *ecut* (crucial for performance and memory)
- *gwr_max_niter, gwr_tolqp_eig:* stopping criteria for GW self-consistency
- *gwr_np_kgts:* to specify the MPI grid (optional)
- *gwr_sigma_algo:* 1 for supercell version, 2 for convolutions in BZ with symmetries
- *gwr_max_hwtene:* Max. transition energy included in the computation of the head/wings of $\chi_{\mathbf{gg'}}(q \to 0)$
- ....

Variables in common with the legacy *GW* code:

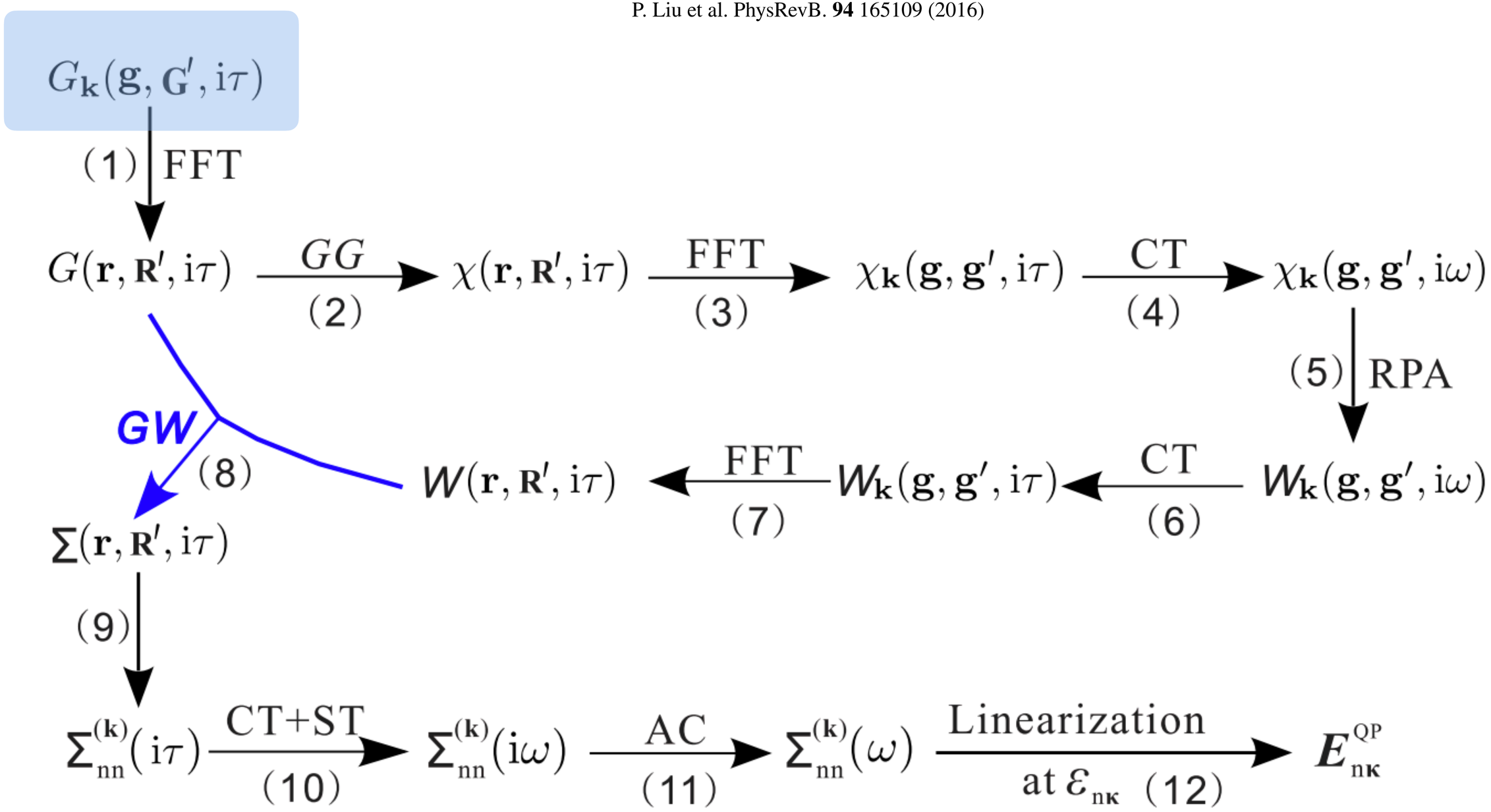
- *ecuteps:* cutoff energy for χ, W
- *ecutsigx:* cutoff energy for Σ_x
- *gw_qprange* or *(nkptgw, kptgw bdgw)* to define states in Σ_{nk}
- *inclvkb:* for the treatment of the $q \to 0$ limit in χ
- *gw_icutcoul, vcutgeo:* treatment of $q \to 0$ divergence and Coulomb cutoff for isolated systems

Input file for G_0W_0 with the GWR code

```
optdriver  6                          # Activate GWR code
gwr_task  "G0W0"                      # One-shot calculation

getden_filepath "GS_DEN"              # Read GS density
getwfk_filepath "GREEN_WFK"           # Read WFK file with empty states

nband      1000                       # Bands in Green's function
gwr_ntau   8                          # Number of minimax points
gwr_boxcutmin  1.1                    # Ratio between FFT box and G-sphere. Default: 1.1

ecuteps    8.0                        # Cut-off energy for dielectric matrix.
ecutsigx   12.0                       # Dimension of the G sum in Sigma_x.

nkptgw     2                          # number of k-point where GW corrections are computed
                                      # set it to 0 to automatically select the fundamental and the direct gap
kptgw                                 # k-points in reduced coordinates
  0.0   0.0   0.0
  0.5   0.0   0.0

bdgw                                  # calculate GW corrections for bands from 4 to 5
  4  5
  4  5
```

GWR algorithm

P. Liu et al. PhysRevB. **94** 165109 (2016)



WFK generation with direct diagonalization

```
optdriver  6                      # enter GWR code
gwr_task "HDIAGO"                 # direct diago.
getden_filepath "GS_DEN"          # read GS density to build H
nband 1200                        # occ + empty states
```

$$G(\mathbf{r},\mathbf{r}',i\tau) = \Theta(\tau)\overline{G}(\mathbf{r},\mathbf{r}',i\tau) + \Theta(-\tau)\underline{G}(\mathbf{r},\mathbf{r}',i\tau)$$

$$\overline{G}(\mathbf{r},\mathbf{r}',i\tau) = -\sum_{n}^{\text{unocc}} \psi_n(\mathbf{r})\psi_n^*(\mathbf{r}')e^{-\varepsilon_n\tau} \qquad (\tau > 0)$$

Bounded exp.

$$\underline{G}(\mathbf{r},\mathbf{r}',i\tau) = \sum_{n}^{\text{occ}} \psi_n(\mathbf{r})\psi_n^*(\mathbf{r}')e^{-\varepsilon_n\tau} \qquad (\tau < 0)$$

Scalapack diago vs iterative eigensolvers:

- Iterative solvers are efficient provided *nband* << *npw*
- High-energy states are difficult to converge with iterative methods
- Direct diago. easily outperforms iterative solvers (*e.g.* lobpcg) if many bands are needed
- In ZnO, for instance, ~3000 bands are needed to converge…

ZnO with 4 nodes on Lumi

ecut	40.0
mpw	3909
ngkpt	8 8 5
nbdbuf	10% nband
tolwfr	1.0d-18

wall-time (s)

nband	slk_diago	lobpcg
1000	11	105
2000	21	306
3000	35	FAIL

(512 cores, 2 Gb per core)

MPI distribution of G, χ, W in GWR

- 4D MPI grid to distribute memory and operations over:
  - collinear spins inside *spin_comm* (trivial algo.)
  - IBZ **k**-points inside *kpt_comm*
  - **g'** components inside g_*comm*
  - $i\tau/i\omega$ points inside tau_*comm* (almost trivial algo.)

$$G^{\sigma}_{\mathbf{k}}(\underbrace{\mathbf{g}, \mathbf{g}'}_{PBLAS}, \pm i\tau)$$

- *spin_comm* and tau_*comm* levels are very efficient (few MPI communications)
- *kpt_comm* and g_*comm* are network intensive but **crucial** to keep memory at bay
- To go to the supercell, indeed, we need to pre-compute and store in memory:

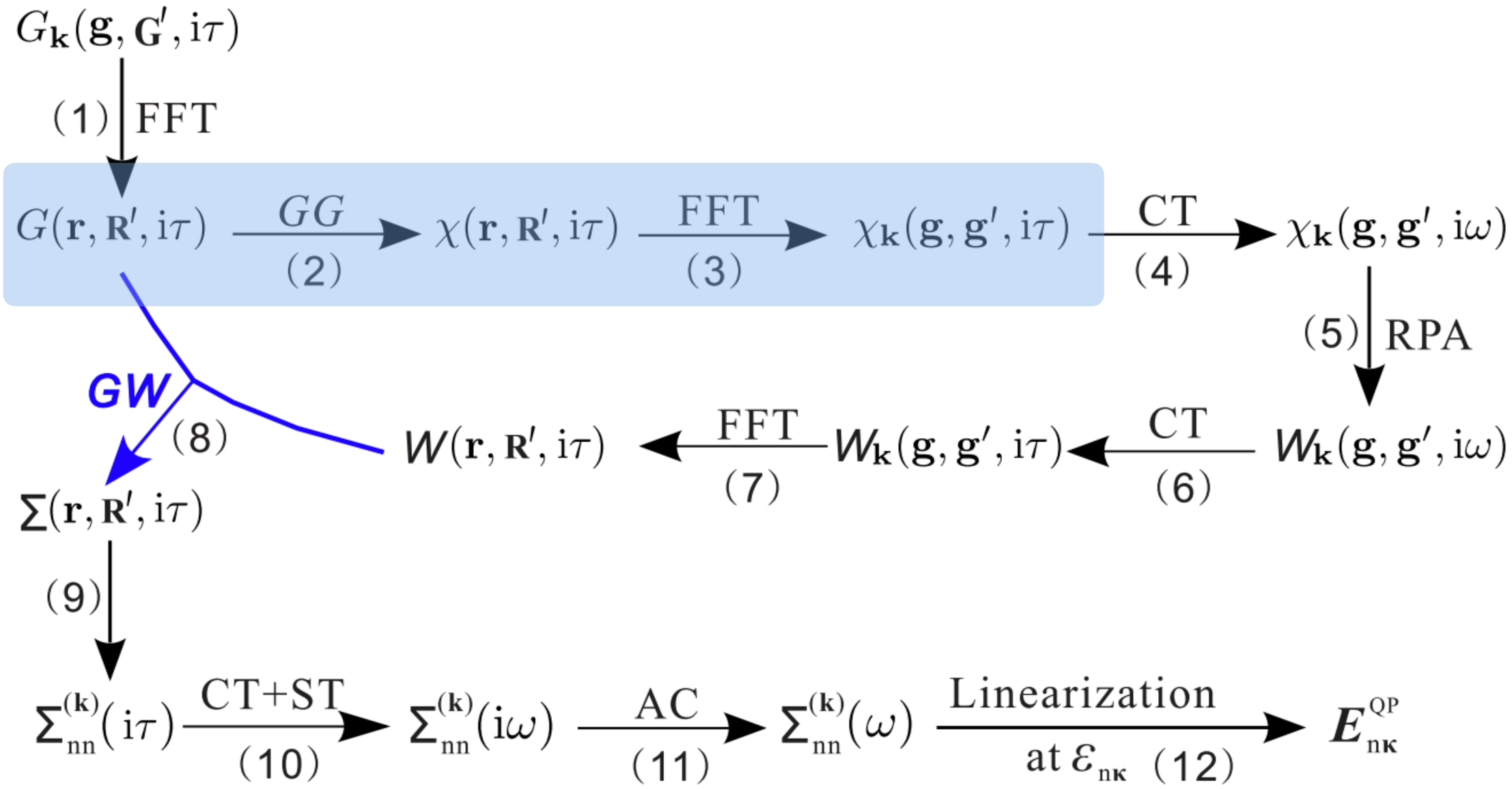
$$G_{\mathbf{k}}(\mathbf{r}, \mathbf{g}') = \sum_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{g})\mathbf{r}} G_{\mathbf{k}}(\mathbf{g}, \mathbf{g}') \quad \text{for each } \mathbf{k} \in \text{BZ}$$

$$\text{memory} \propto \frac{N_{\text{BZ}}}{\text{np}_k} \times \frac{\text{nfft}}{\text{np}_g} \times \text{npw}$$

- For optimal performance, MPI procs should be a multiple of *gwr_ntau* x *nsppol* but mind the memory for *G*!
- Matrices are stored in single precision by default (—enable-gw-dpc="yes" to use double precision)

GWR algorithm

P. Liu et al. PhysRevB. **94** 165109 (2016)



From G to χ in $i\tau$ space (step 1)

NB: the loop over τ-points is external. At each iteration, we have to consider $\pm\tau_i$ (not always shown in the equations)

For each **k** in the BZ, do:

➡ Use symmetries to build $G_{\mathbf{k}}$ in the BZ

➡ FFT along **g** index

➡ MPI-transpose to have $\mathbf{g}'$ local on each proc

$$G_{\mathbf{k}_{IBZ}}(\mathbf{g},\mathbf{g}') \overset{symm}{\rightarrow} G_{\mathbf{k}}(\mathbf{g},\mathbf{g}')$$

k in the BZ

$$G_{\mathbf{k}}(\mathbf{r},\mathbf{g}') = \sum_{\mathbf{g}} e^{i(\mathbf{k}+\mathbf{g})\mathbf{r}} G_{\mathbf{k}}(\mathbf{g},\mathbf{g}')$$

Local to each MPI proc

MPI distributed inside **g_comm**

$$G_{\mathbf{k}}(\mathbf{r},\mathbf{g}') \overset{PTRANS}{\rightarrow} \tilde{G}_{\mathbf{k}}(\mathbf{g}',\mathbf{r})$$

Local

MPI distributed

Cons:

- Workspace memory $\propto \frac{N_{\mathrm{BZ}}}{\mathrm{np}_k} \times \frac{\mathrm{nfft}}{\mathrm{np}_g} \times \mathrm{npw}$
- Lots of calls to PTRANS: $\left(\frac{2N_{\mathrm{BZ}}}{\mathrm{np}_k}\right)$
- Memory increases with $N_{\mathbf{r}}$ (*ecut* and *gwr_boxcutratio*)

Pros:

- Linear scaling in N_{BZ}
- Scales well with np_k (less PTRANS calls)

From G to χ in $i\tau$ space (step 2)

Step 1. For each $\mathbf{r}$ in unit cell, use $\tilde{G}$ to compute:

$$G(\mathbf{r}, \mathbf{R}') = \sum_{\mathbf{k}\mathbf{g}'} G(\mathbf{r}, \mathbf{k} + \mathbf{g}') e^{-i(\mathbf{k}+\mathbf{g}')\mathbf{R}'}$$

Need all $\mathbf{k}$ in the BZ for the FFT!

$\mathbf{k}$-parallelism is really low-level!

$$\chi(\mathbf{r}, \mathbf{R}', i\tau) = G(\mathbf{r}, \mathbf{R}', i\tau) G^*(\mathbf{R}', \mathbf{r}, -i\tau)$$

Only $\chi_{\mathbf{r}}(\mathbf{R}')$ is stored at fixed $\mathbf{r}$

$$\chi(\mathbf{r}, \mathbf{G}') = \sum_{\mathbf{R}' \in S} \chi(\mathbf{r}, \mathbf{R}') e^{i\mathbf{G}'\mathbf{R}'}$$

Transform *immediately* to $\mathbf{G}'$-space ($\mathbf{k} + \mathbf{g}'$) and store results in temp. PBLAS matrix $\tilde{\chi}$

Step 2. Once all $\mathbf{r}$ have been computed, MPI-transpose χ and perform FFT along the $\mathbf{r}$-axis

$$\chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}') = \sum_{\mathbf{r} \in C} e^{-i(\mathbf{k}+\mathbf{g})\mathbf{r}} \chi(\mathbf{r}, \mathbf{k} + \mathbf{g}')$$

Only k-points in the IBZ are stored
Matrices are PBLAS-distributed

Cons:

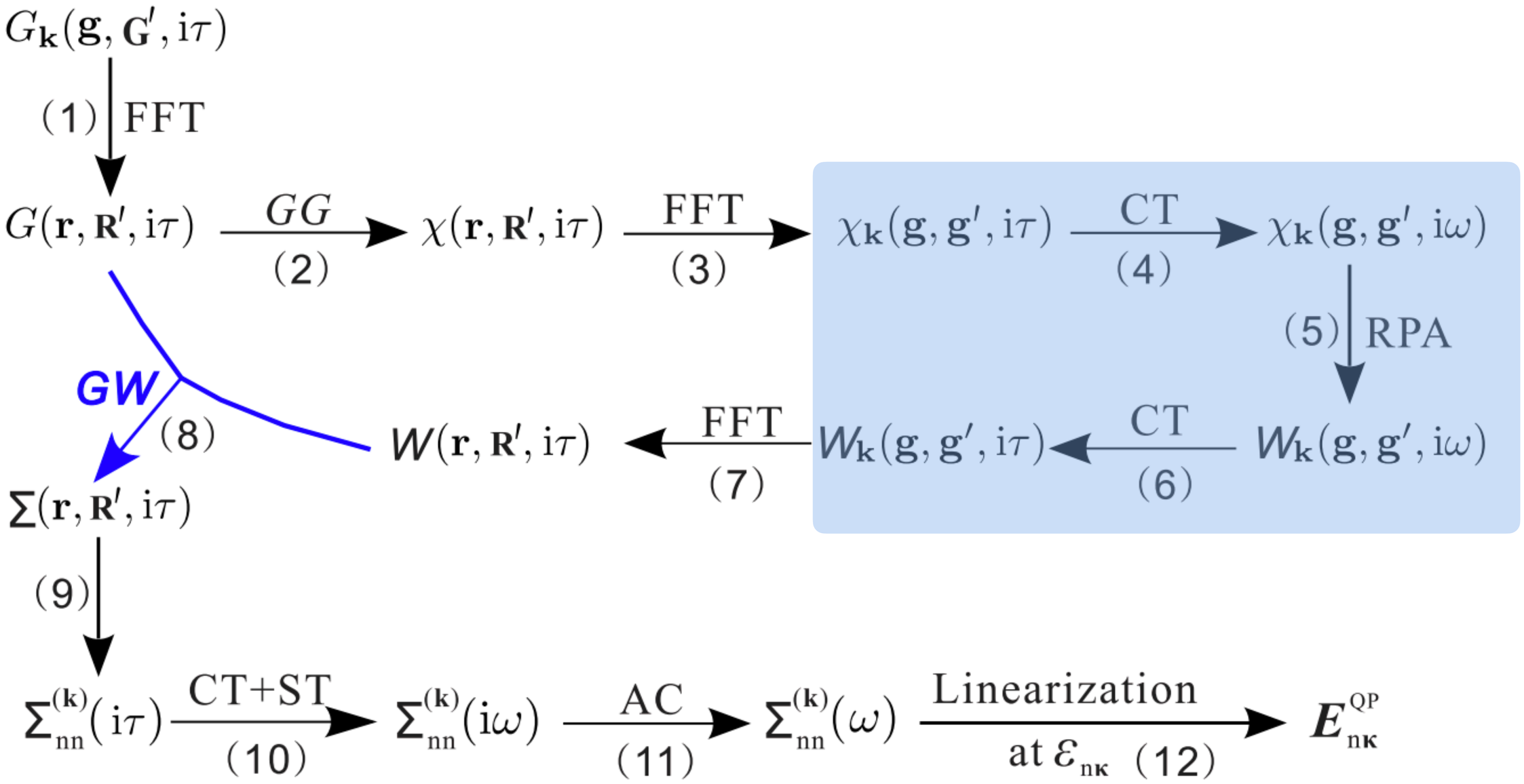
- $\mathbf{k}$-parallelism requires *nfft* communications
- We loose part of the speedup gained in step 1

Pros:

- Tons of FFTs in batch mode (blocking over $\mathbf{r}$)
- Ideal scenario for OpenMP/GPUs

GWR algorithm

P. Liu et al. PhysRevB. **94** 165109 (2016)



Computing W from χ

Step 1. Cosine transform $(i\omega \to i\tau)$:

Requires communication inside tau_*comm*

$$\chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega_k) = \sum_{j=1}^{N} \gamma_{kj} \cos(\omega_k \tau_j) \chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\tau_j)$$

Step 2. Compute *symmetrized* dielectric matrix:

$$\varepsilon_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) = \delta_{\mathbf{g}\mathbf{g}'} - v_{\mathbf{k}}(\mathbf{g}, \mathbf{g}') \chi_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) \qquad v_{\mathbf{k}}(\mathbf{g}, \mathbf{g}') = \frac{4\pi}{|\mathbf{k}+\mathbf{g}|\,|\mathbf{k}'+\mathbf{g}'|}$$

vcutgeo selects the expression for v

Step 3. Compute *correlated screened Coulomb* interaction $\tilde{W}$:

$$W_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) = v_{\mathbf{k}}(\mathbf{g}, \mathbf{g}') \epsilon_{\mathbf{k}}^{-1}(\mathbf{g}, \mathbf{g}', i\omega) \qquad \tilde{W}_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) = W_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega) - v_{\mathbf{k}}(\mathbf{g}, \mathbf{g}')$$

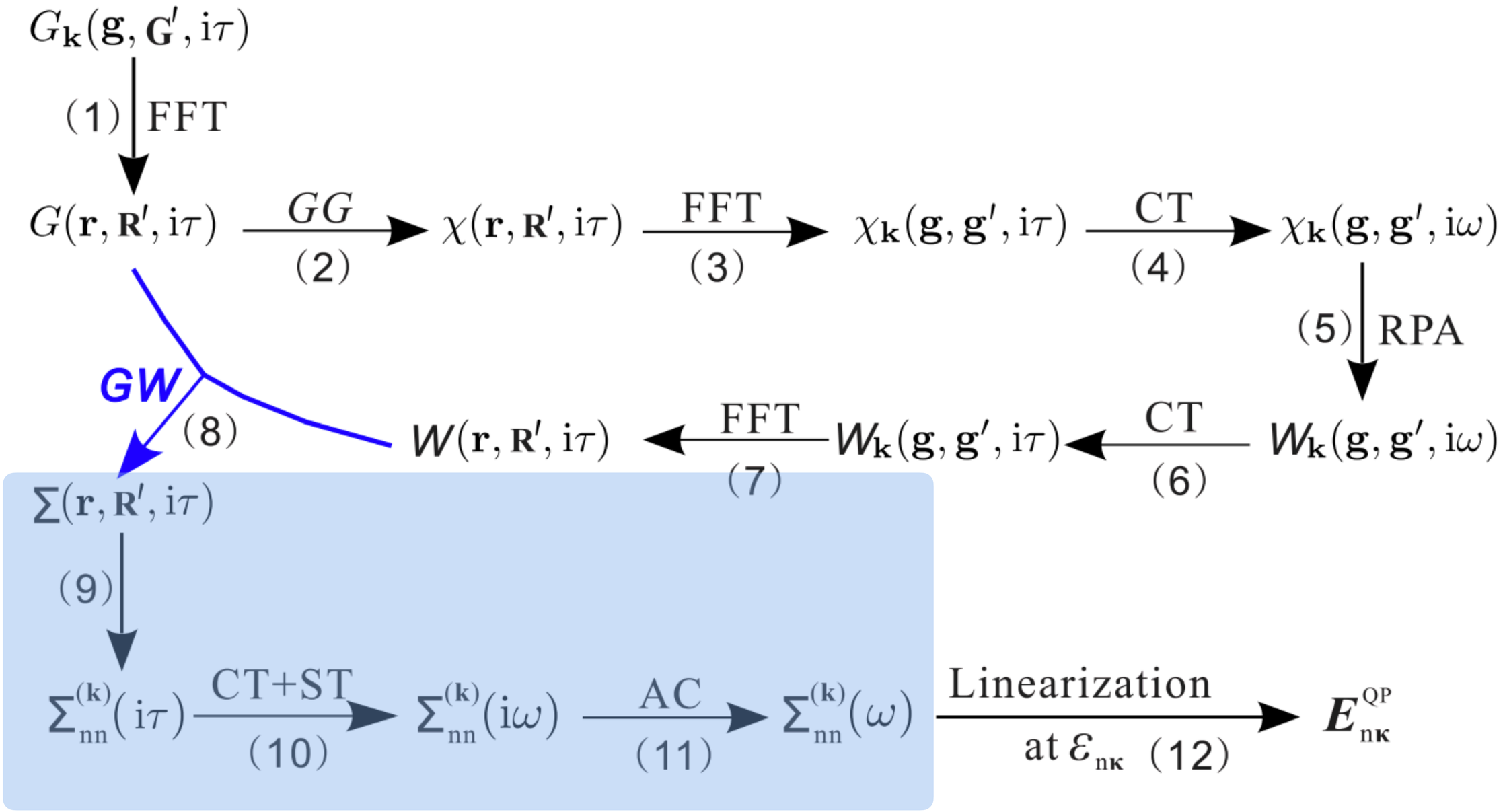
Matrix inversion with Scalapack/ELPA.

Step 4. Inverse Cosine Transform $(i\tau \to i\omega)$:

$$\tilde{W}_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\tau_k) = \sum_{j=1}^{N} \xi_{kj} \cos(\omega_k \tau_j) \tilde{W}_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\omega_j)$$

Requires communication inside tau_*comm*

GWR algorithm



Computing $\Sigma_{n\mathbf{q}}(\omega)$

Step 1. FFTs in the unit cell:

$$G_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\tau) \overset{FFT}{\Longrightarrow} G_{\mathbf{k}}(\mathbf{r}, \mathbf{g}', i\tau) \qquad \tilde{W}_{\mathbf{k}}(\mathbf{g}, \mathbf{g}', i\tau) \overset{FFT}{\Longrightarrow} \tilde{W}_{\mathbf{k}}(\mathbf{r}, \mathbf{g}', i\tau)$$

Step 2. For each $\mathbf{r}$ in C do:

$$G(\mathbf{r}, \mathbf{R}', i\tau) = \sum_{\mathbf{k}\mathbf{g}'} G(\mathbf{r}, \mathbf{k}+\mathbf{g}', i\tau) e^{-i(\mathbf{k}+\mathbf{g}')\mathbf{R}'}$$

$$W(\mathbf{r}, \mathbf{R}', i\tau) = \sum_{\mathbf{k}\mathbf{g}'} W(\mathbf{r}, \mathbf{k}+\mathbf{g}', i\tau) e^{-i(\mathbf{k}+\mathbf{g}')\mathbf{R}'}$$

$$\Sigma(\mathbf{r}, \mathbf{R}', i\tau) = -\, G(\mathbf{r}, \mathbf{R}', i\tau) W(\mathbf{r}, \mathbf{R}', i\tau)$$

Avoid storing full $\Sigma(\mathbf{r}, \mathbf{R}')$ in memory

$$\Sigma_{n\mathbf{q}}(i\tau) = \Sigma_{n\mathbf{q}}(i\tau) + \sum_{\mathbf{R}' \in S} \psi^*_{n\mathbf{q}}(\mathbf{r})\, \Sigma(\mathbf{r}, \mathbf{R}', i\tau)\, \psi_{n\mathbf{q}}(\mathbf{R}')$$

Compute partial contribution to $\Sigma_{n\mathbf{q}}$ and accumulate

Step 3. sine/cosine transforms to go to $i\omega$ space, followed by *analytic continuation* to the real-ω:

$$\Sigma_{n\mathbf{q}}(i\tau) = \Sigma^C_{n\mathbf{q}}(i\tau) + \Sigma^S_{n\mathbf{q}}(i\tau) \overset{CT+ST}{\Longrightarrow} \Sigma_{n\mathbf{q}}(i\omega) \overset{AC}{\Longrightarrow} \Sigma_{n\mathbf{q}}(\omega)$$

Step 4. Add exchange part (*sum over occ states directly*). Finally, solve the linearized QP equation

Is GWR faster than the legacy code?

Well, it depends:

- In small symmetric systems, the quartic code is still competitive but W is not MPI-distributed!
- GWR is superior if:
  - low-symmetry systems with dense **k**-meshes
  - large *ecuteps* or *nband*
  - G_0W_0 without *PPM*
  - *off-diagonal* matrix elements of Σ are needed for self-consistency

Benchmark results for ZnO:

- 8 nodes on Lumi, 2 Gb per core
- *ecut* 40.0
- *ecuteps* 12
- *ngkpt* 8 8 5
- *nomega/gwr_ntau* = 12
- $\mathrm{np}_\tau = 2$ in GWR

wall-time (s)

nband	Quartic GW	GWR
1000	3023	1947
2000	MEM_FAIL	2145
3000	MEM_FAIL	2432

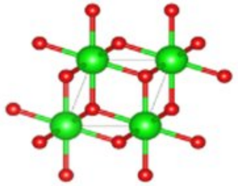
wall-time spent in the GWR routines for nband 1000

```
read_ugb_from_wfk: 16.04 [s]
build_chi0_head_and_wings: 54.36 [s]
build_sigxme:  0.64 [s]
build_green:  4:48 [minutes]
cos_transform:  4.23 [s]
build_chi:  13:11 [minutes]
cos_transform:  0.72 [s]
build_wc:  3.02 [s]
build_sigmac: 13:09 [minutes]
```

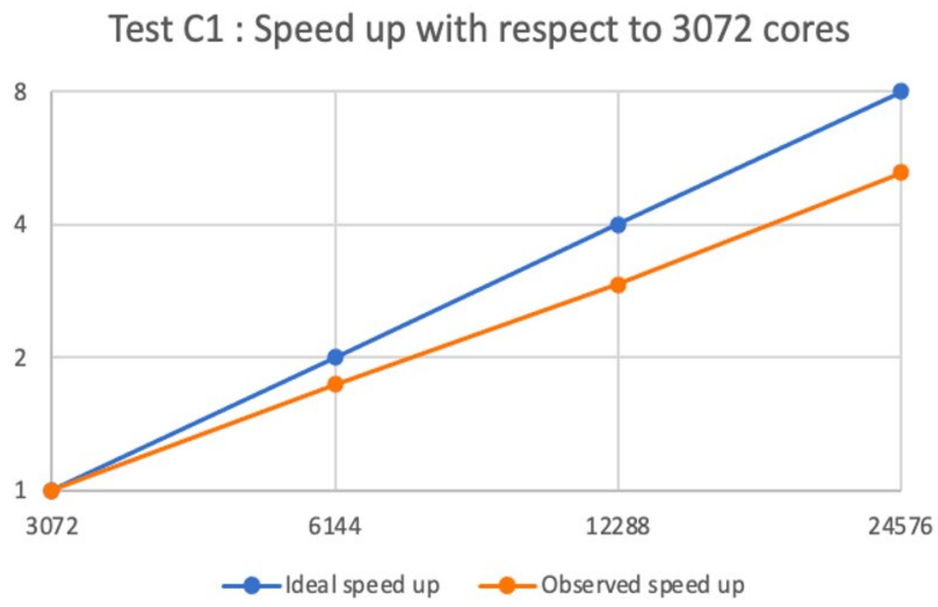
- Most of the wall-time spent to build χ and Σ in the supercell (*build_chi* and *build_sigmac*)
- Unlike the quartic-code, Σ is as expensive as χ (but we have symmetry tricks to accelerate this part, *gwr_sigma_algo* = *2*)

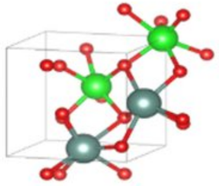
Strong scaling of GWR code

Benchmarks performed by L. Baquet



ZrO_2

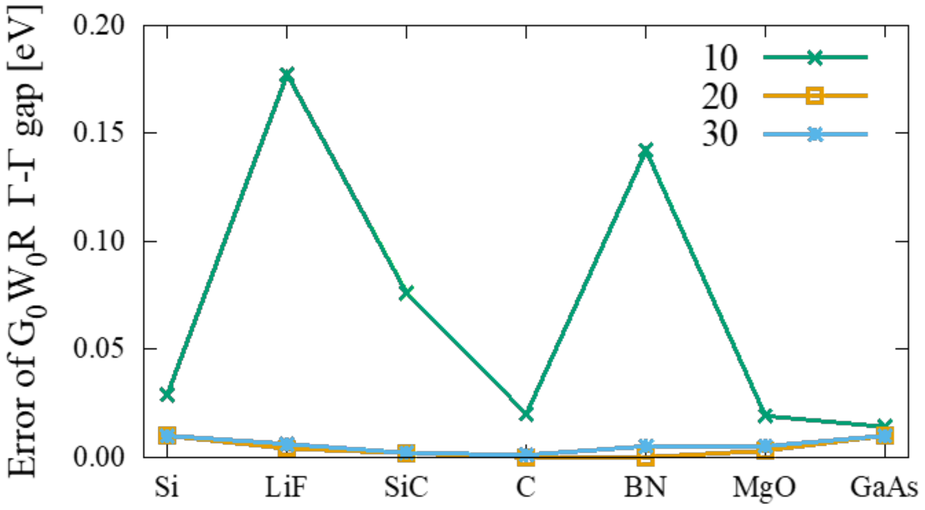




$Zr_2Y_2O_7$

Discrepancy between low-scaling G_0W_0 implementation and quadratic one, for 7 different solids.

Using 20 time-frequency points or more allows one to obtain a numerical error **on the order of 0.02eV or below**.



Phys. Rev. B **109**, 245101 – Published 3 June 2024

Pros and cons of GWR code

Pros:

- Cubic scaling in *natom*
- Linear scaling with N_k in the full BZ
- Fast convergence with minimax mesh (~20 points)
- *GW* beyond PPA: $\Sigma(\omega)$ and $A(\omega)$ at reasonable cost
- Computing off-diagonal $\Sigma^{\mathbf{k}}_{mn}$ for all **k**-points in the IBZ is not as expensive as in legacy code

Cons:

- Symmetries are more difficult to exploit, especially in the supercell
- Requires Pade' to go back to the real axis: $\Sigma(i\omega) \rightarrow \Sigma(\omega)$
- Much more memory-demanding than conventional *GW* algorithm
- Requires different MPI levels and PBLAS distribution of G, χ, W to make memory scale

Future directions:

- OpenMP threads to reduce the MPI_ALLTOALL bottleneck
- Porting FFTs to GPUs

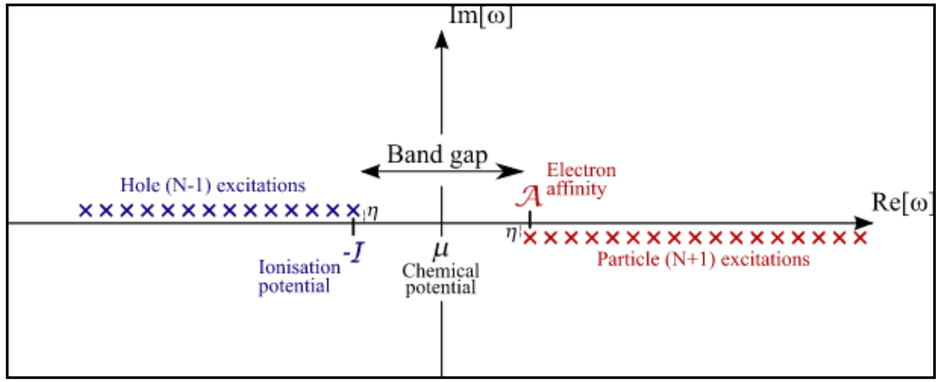
Thank you for your attention!

Supplemental material

Green's function: real ω vs $i\tau$ space

G in ω-space **(real axis)**

$$G(\mathbf{r},\mathbf{r}',\omega) = \sum_n \frac{\psi_n(\mathbf{r})\psi_n^*(\mathbf{r}')}{\omega - \varepsilon_n + i\delta^+\mathrm{sign}(\varepsilon_n)}$$



- Branch cuts and poles → ω-integration is tricky
- Analytic expression for RPA $\tilde{\chi}(\omega)$
- Direct connection with QP energies and spectral function $A(\omega)$

G in imaginary time $i\tau$

$$G(\mathbf{r},\mathbf{r}',i\tau) = \Theta(\tau)\overline{G}(\mathbf{r},\mathbf{r}',i\tau) + \Theta(-\tau)\underline{G}(\mathbf{r},\mathbf{r}',i\tau)$$

$$\overline{G}(\mathbf{r},\mathbf{r}',i\tau) = -\sum_n^{\text{unocc}} \psi_n(\mathbf{r})\psi_n^*(\mathbf{r}')e^{-\varepsilon_n\tau} \qquad (\tau > 0)$$

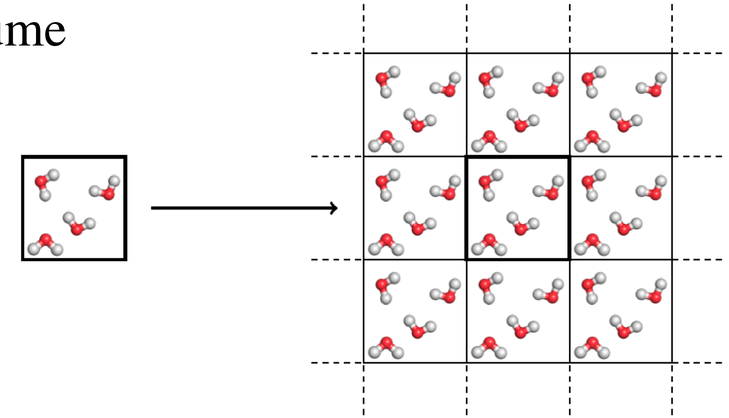
$$\underline{G}(\mathbf{r},\mathbf{r}',i\tau) = \sum_n^{\text{occ}} \psi_n(\mathbf{r})\psi_n^*(\mathbf{r}')e^{-\varepsilon_n\tau} \qquad (\tau < 0)$$

Bounded exp.

- Smooth behaviour in $i\tau/i\omega$ → integration is "easier"
- Requires $i\tau \Rightarrow i\omega$ transforms
- Requires analytic continuation to go back to the real-ω axis before computing QP energies and $A(\omega)$

Plane-wave expansion of two-point functions

- Infinite system simulated with Born-von-Karman (BvK) periodic boundary conditions *i.e.* (N_1, N_2, N_3) supercell of volume $V = N\Omega$ with $N = N_1 N_2 N_3$ and Ω the unit cell volume



- $G, \tilde{\chi}, W$ are defined in the BvK supercell
- $G, \tilde{\chi}, W$ are invariant if we translate both $\mathbf{r_1}$ and $\mathbf{r_2}$ by $\mathbf{R}$ *i.e.*:

$$G(\mathbf{r}_1, \mathbf{r}_2) = G(\mathbf{r}_1 + \mathbf{R}, \mathbf{r}_2 + \mathbf{R})$$

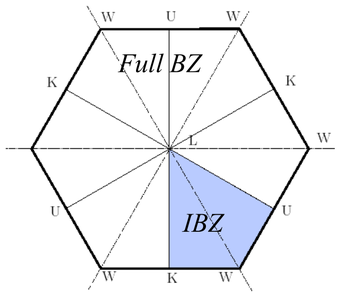
- This implies the Fourier expansion:

$$f(\mathbf{r}_1, \mathbf{r}_2) = \frac{1}{V} \sum_{\substack{\mathbf{q} \\ \mathbf{G}_1 \mathbf{G}_2}} e^{i(\mathbf{q}+\mathbf{G}_1)\cdot\mathbf{r}_1} f_{\mathbf{G}_1\mathbf{G}_2}(\mathbf{q})\, e^{-i(\mathbf{q}+\mathbf{G}_2)\cdot\mathbf{r}_2}$$

$$f_{\mathbf{G}_1\mathbf{G}_2}(\mathbf{q}) = \frac{1}{V} \iint_V e^{-i(\mathbf{q}+\mathbf{G}_1)\cdot\mathbf{r}_1} f(\mathbf{r}_1, \mathbf{r}_2)\, e^{i(\mathbf{q}+\mathbf{G}_2)\cdot\mathbf{r}_2}\, \mathrm{d}\mathbf{r}_1\, \mathrm{d}\mathbf{r}_2$$

where the $\mathbf{q}$-points belong to the BZ mesh dual to the BvK supercell: $(\frac{1}{N_1}, \frac{1}{N_2}, \frac{1}{N_3})$

Spatial symmetries in GW



- KS wavefunctions in the BZ can be reconstructed from the IBZ:

$$[H, \{\mathscr{R}, \mathbf{t}\}] = 0 \longrightarrow \begin{cases} \epsilon_{\mathscr{R}\mathbf{k}} & = & \epsilon_{\mathbf{k}} \\ u_{\mathscr{R}\mathbf{k}}(\mathbf{r}) & = & e^{-i\mathscr{R}\mathbf{k}\cdot\mathbf{t}}\, u_{\mathbf{k}}\big(\mathscr{R}^{-1}(\mathbf{r}-\mathbf{t})\big) \\ u_{\mathscr{R}\mathbf{k}}(\mathbf{G}) & = & e^{-i(\mathscr{R}\mathbf{k}+\mathbf{G})\cdot\mathbf{t}}\, u_{\mathbf{k}}(\mathscr{R}^{-1}\mathbf{G})\,. \end{cases}$$

Rotation matrix Fractional translation

- Spatial symmetry for the polarizability:

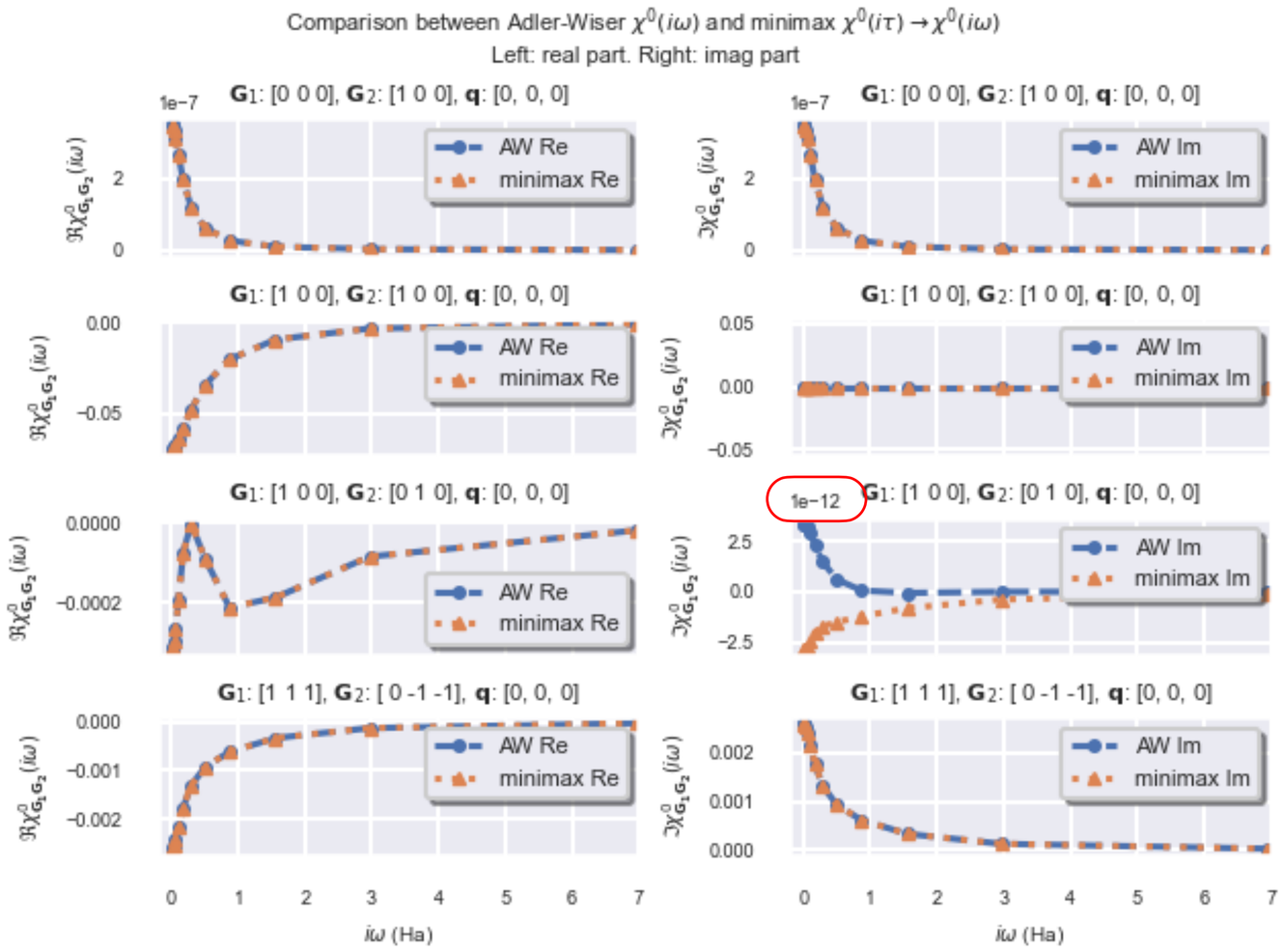
$$\chi^0(\mathbf{r}_1, \mathbf{r}_2) = \chi^0\big(\mathscr{R}^{-1}(\mathbf{r}_1 - \mathbf{t}), \mathscr{R}^{-1}(\mathbf{r}_2 - \mathbf{t})\big) \longrightarrow \chi^0{}_{\mathbf{G}_1\mathbf{G}_2}(\mathscr{R}\mathbf{q}) = e^{i\mathbf{t}\cdot(\mathbf{G}_2-\mathbf{G}_1)}\chi^0{}_{\mathscr{R}^{-1}\mathbf{G}_1\,\mathscr{R}^{-1}\mathbf{G}_2}(\mathbf{q})$$

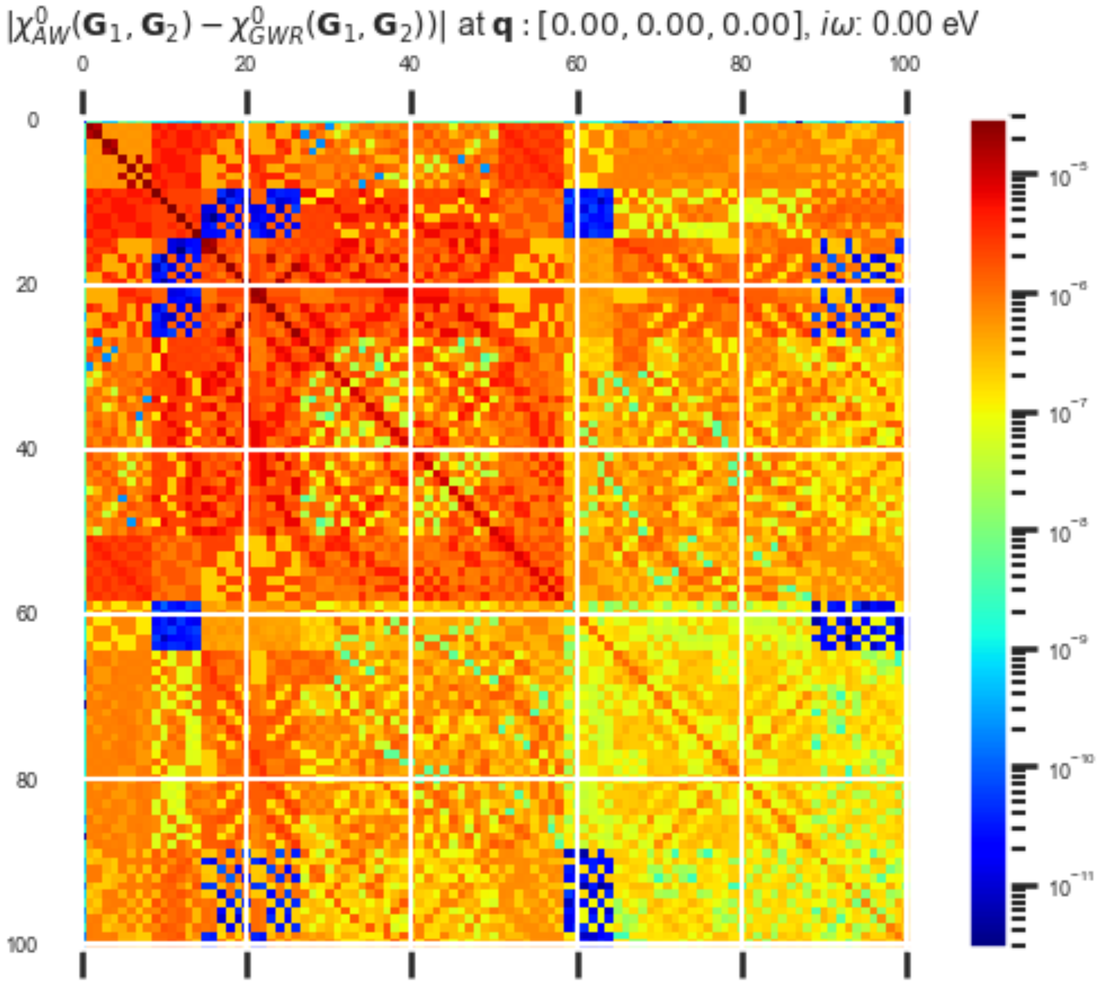
Take-home message:

- Bloch states are computed in the IBZ and then reconstructed in the BZ at *runtime*
- $G(\mathbf{k})$ and $\chi^0(\mathbf{q})$ are computed and stored only for $\mathbf{k}/\mathbf{q}$ in the IBZ
- BZ integrals depending on an external $\mathbf{q}$, can be restricted to the $\mathrm{IBZ}_{\mathbf{q}}$ defined by the *little-group* of $\mathbf{q}$
- Significant speedup and memory saving in *high-symmetry* systems. Time-reversal can be easily included

Validation: $\chi(\omega)$ with GWR and Adler-Wiser

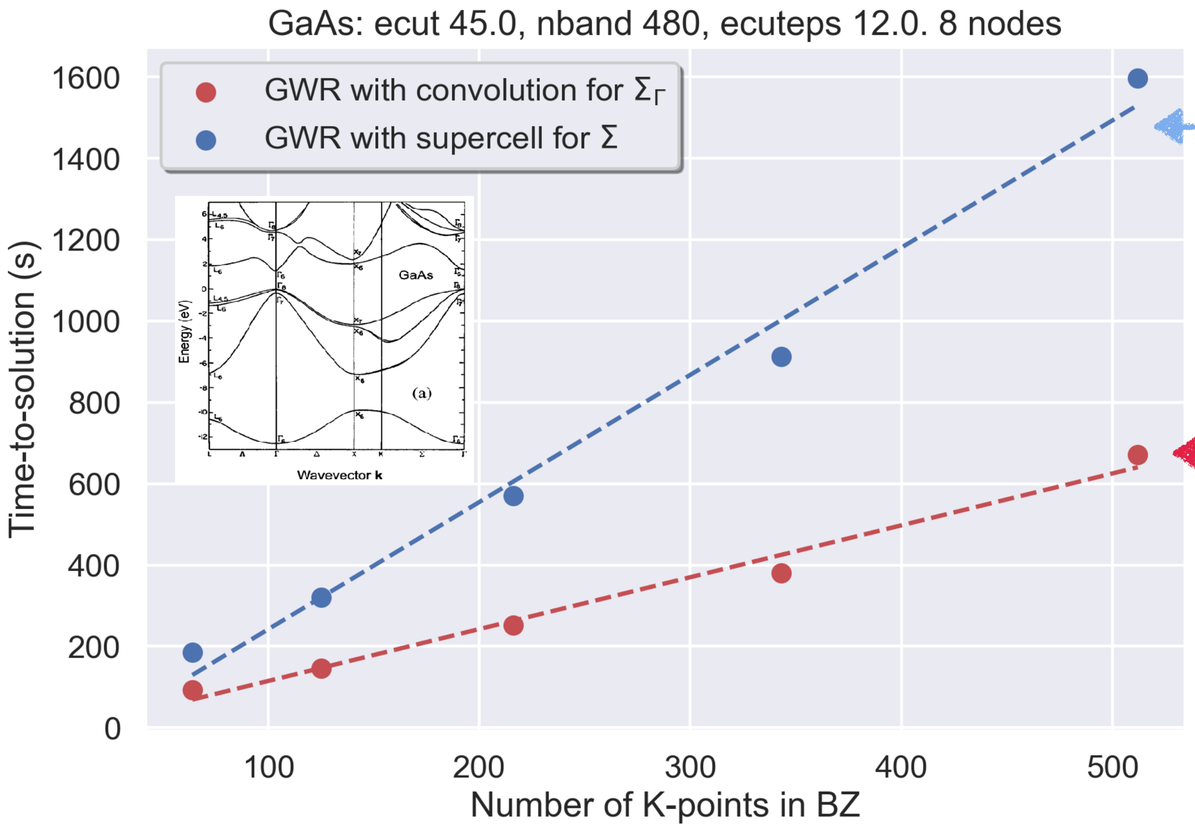
- Silicon with 4x4x4 Γ-centered **k**-mesh
- *gwr_ntau = 12*
- *nband* = 100 and *inclvkb* 2 to compute head and wings





Scaling of GWR algo. with the k-mesh size

- Linear scaling with the BZ size but computing Σ in the SC is more expensive than χ (cpu and memory)
- If one-shot QPs are needed only at the CBM/VBM, convolution + symmetries for $\Sigma_{n\mathbf{k}}$ is faster:



$$\Sigma(\mathbf{r},\mathbf{R}) \approx G(\mathbf{r},\mathbf{R})W(\mathbf{r},\mathbf{R})$$

self-energy in the supercell

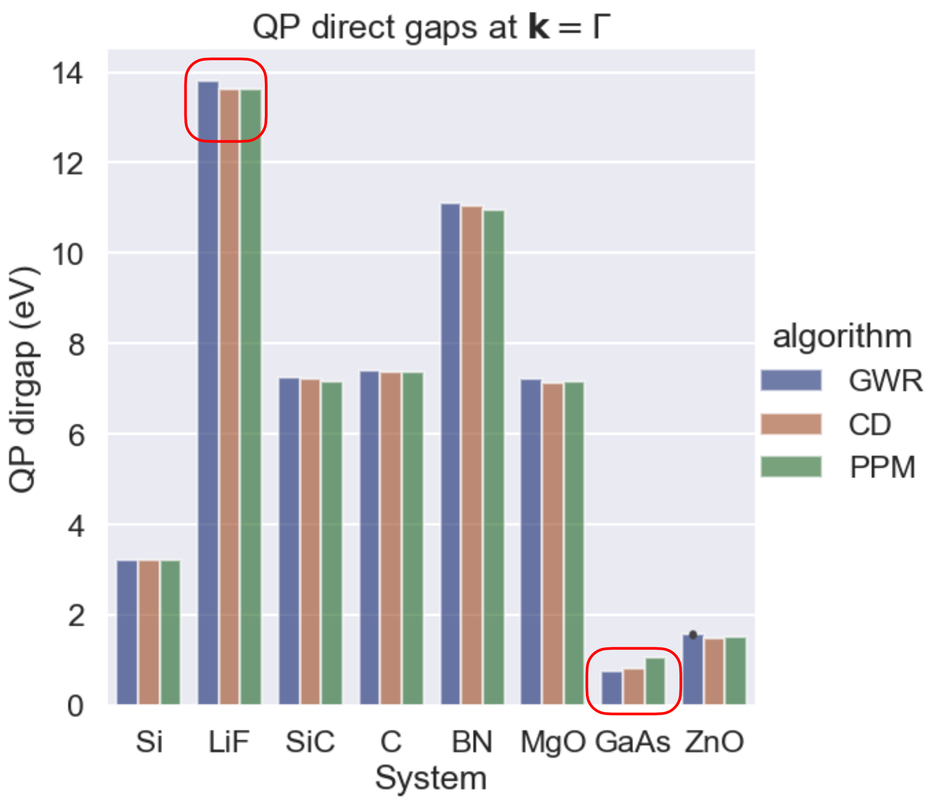
$$\Sigma_{\mathbf{k}}(\mathbf{r},\mathbf{r}') \approx \sum_{\mathbf{q}}^{\mathscr{L}_{\mathbf{k}}} G_{\mathbf{k}+\mathbf{q}}(\mathbf{r},\mathbf{r}')W_{\mathbf{q}}(\mathbf{r},\mathbf{r}')$$

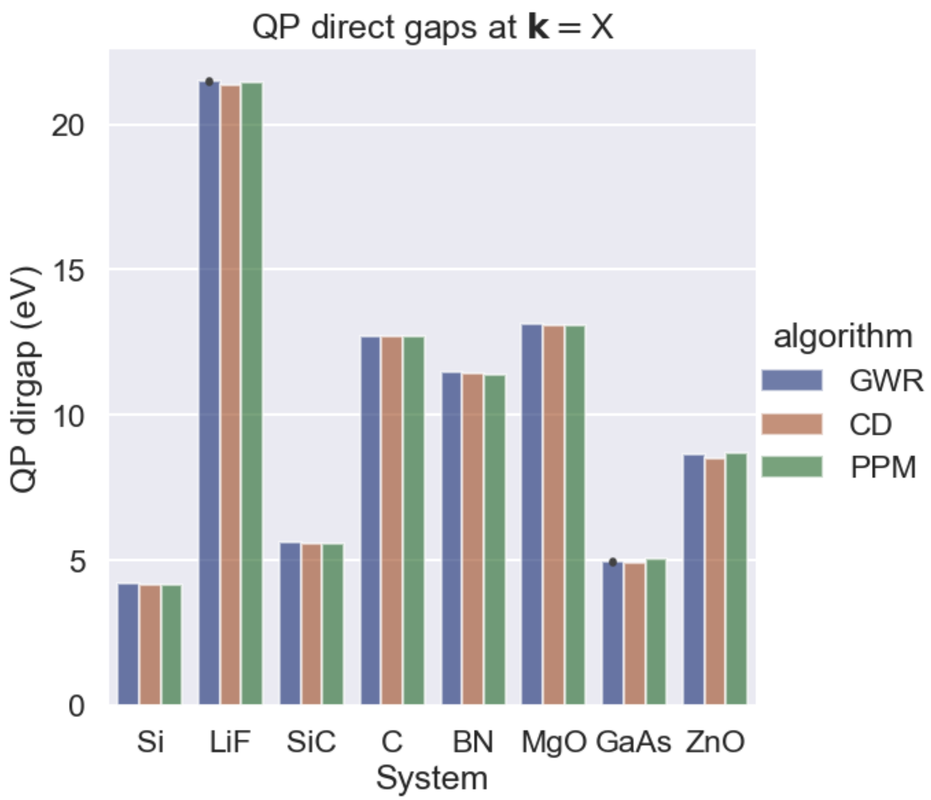
self-energy at **k** via convolutions and symmetries (*gwr_sigma_algo = 2*)

- Computing Σ in the supercell is the recommended approach if one needs $\Sigma_{n\mathbf{k}}$ for **all** **k** in the IBZ, e.g.:
  - band structure interpolation of G_0W_0 results
  - self-consistency (requires off-diagonal matrix elements for which symmetries are not easy to exploit)

QP direct gaps with GWR and quartic GW

- 4x4x4 Γ-centered **k**-mesh
- $nband = 100 \times n_{\mathrm{occ}}$, $ecuteps = 14$ Ha
- *gwr_ntau = 20 in GWR, nfreqre = 50, freqremax=1.5 Ha, nfreqim 10* for CD





- Overall, good agreement. CD is our reference
- Largest difference between GWR and quartic code for LiF at Γ (~0.2 eV)
- In GaAs, GWR and CD agree with each other, PPM overestimates CD/GWR by ~0.2 eV