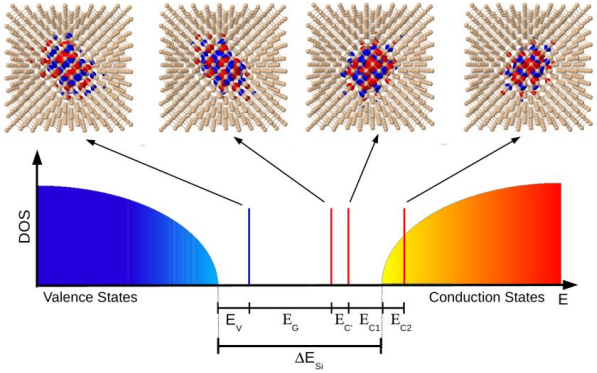


C2SEPEM

BerkeleyGW

Porting Large-Scale Materials Science Codes to GPUs for Next Generation Exascale Architectures

Mauro Del Ben (LBNL)

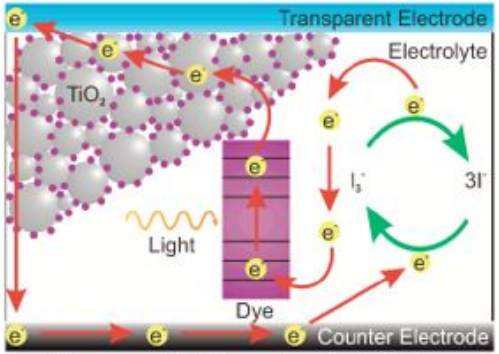
OpenACC Summit 2021

Outlook

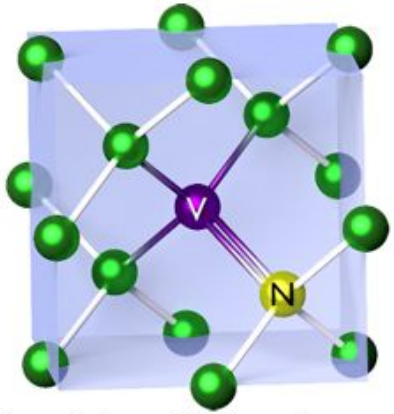
- Introduction / Motivation
- BerkeleyGW and the Epsilon module
- Baseline performance
- GPU support via OpenACC for Epsilon
- Summary

Material Science/Chemistry on the Path to Exascale

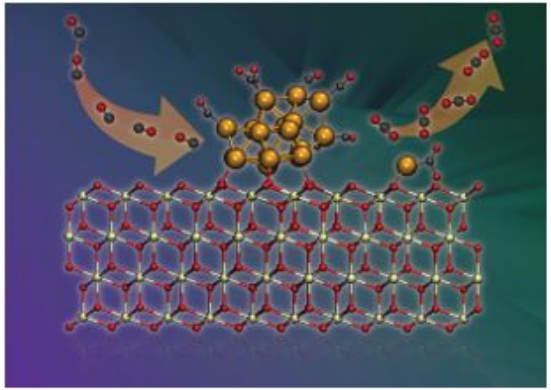
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



Defects in crystals: **qubits**/**quantum computers**
https://www.nist.gov/programs-projects/diamond-nv-center-magnetometry

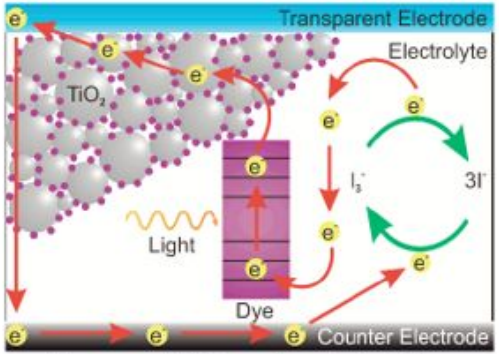


Chemical reaction at interfaces: **Catalysis**

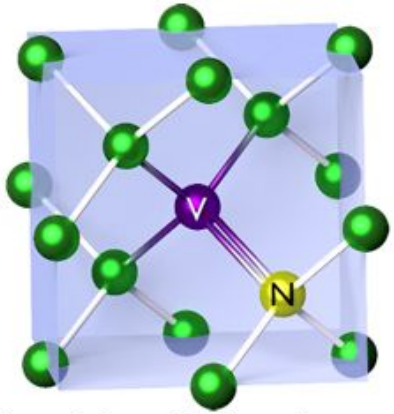
Mat. Sci & Chem apps, such as VASP, Quantum ESPRESSO, QMCPACK, NWchem, BerkeleyGW, CP2K, etc... **heavily use HPC facilities**

Material Science/Chemistry on the Path to Exascale

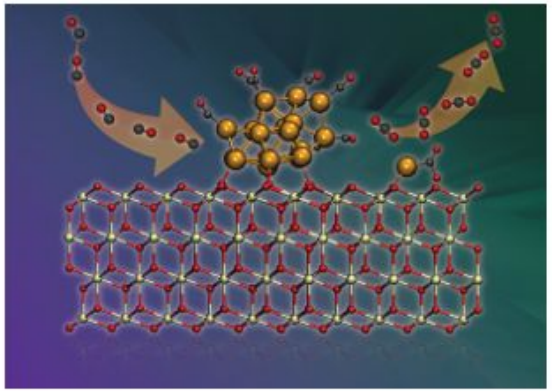
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



Defects in crystals: **qubits/quantum computers**
https://www.nist.gov/programs-projects/diamond-nv-center-magnetometry



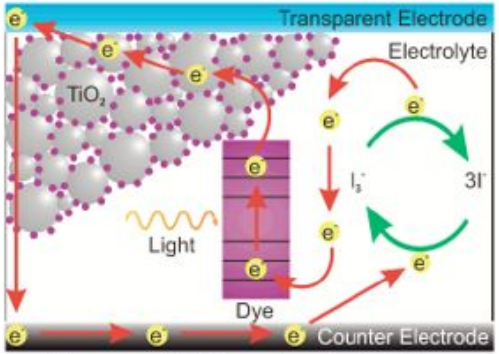
Chemical reaction at interfaces: **Catalysis**

Used to **study and understand the fundamental electronic properties of materials**: *necessary to design the components of novel devices*

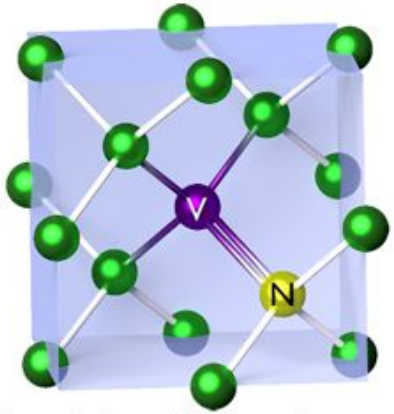
- Applications: Quantum Computers, Batteries, Photovoltaics, Catalysis, etc...

Material Science/Chemistry on the Path to Exascale

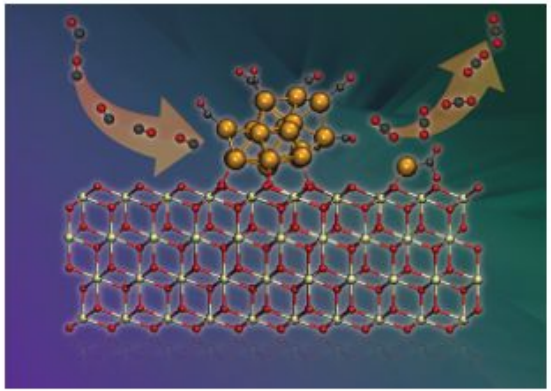
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



Defects in crystals: **qubits/quantum computers**
https://www.nist.gov/programs-projects/diamond-nv-center-magnetometry



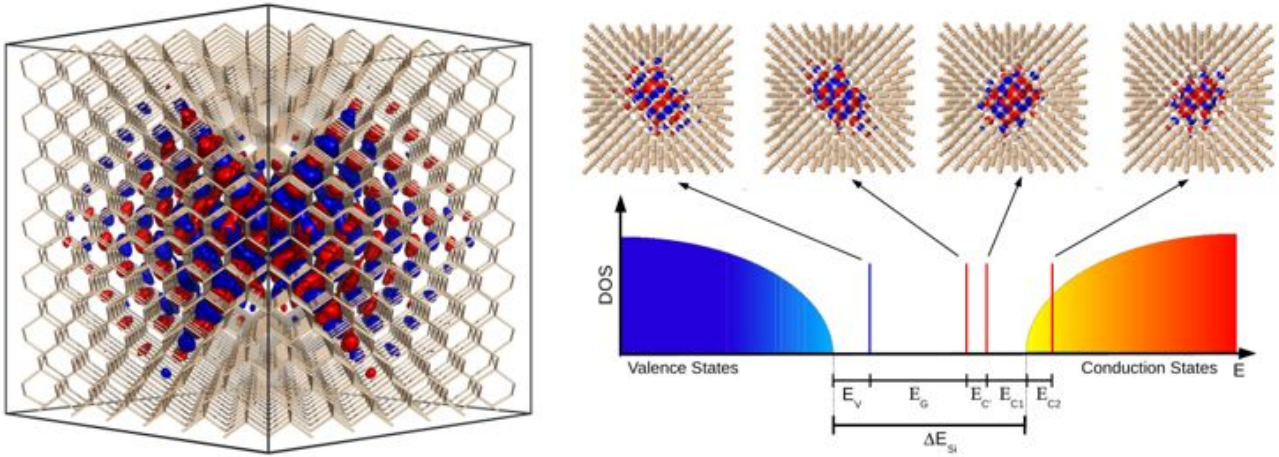
Chemical reaction at interfaces: **Catalysis**

Density Functional Theory (DFT) the workhorse for over three decades

- Excellent compromise between accuracy and computational efficiency
- Ground state theory: often problematic for excited state phenomena

Excited State Properties of Complex Materials

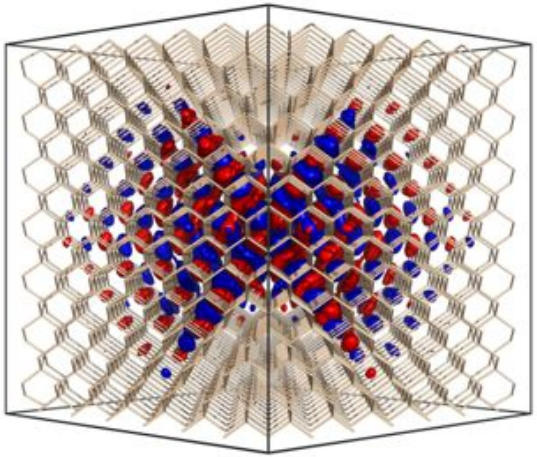
Focus shift from ground to excited state properties

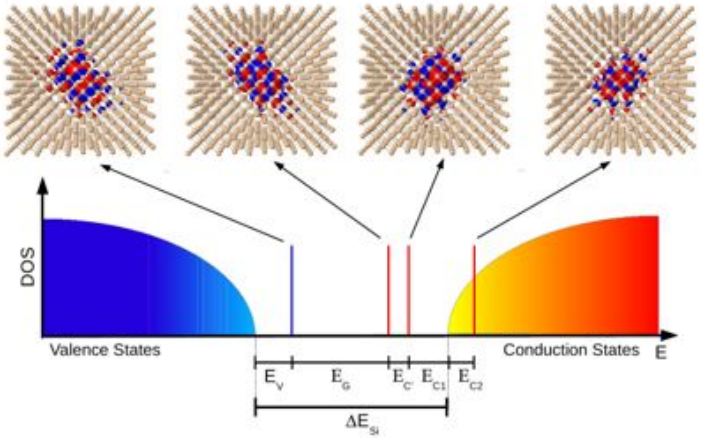


Example: Divacancy point defect in crystalline silicon, prototype of a solid-state Qubit

Excited State Properties of Complex Materials

Focus shift from ground to excited state properties



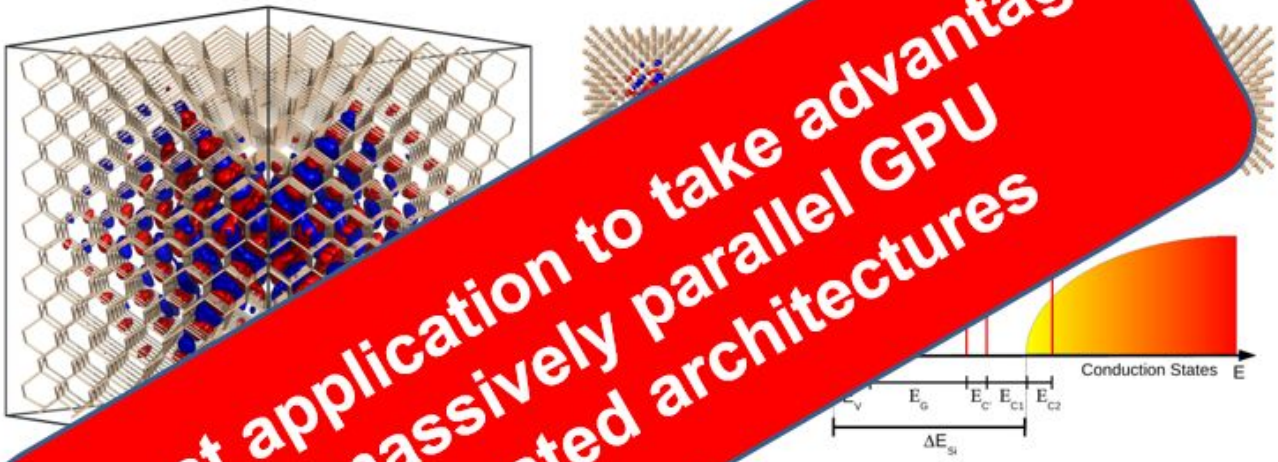


Example: Divacancy point defect in crystalline silicon, prototype of a solid-state Qubit

- Accuracy beyond DFT: **GW and GW+BSE**
- Unprecedented simulation sizes: **1000's of atoms**

Excited State Properties of Complex Materials

Focus shift from ground to excited state



Ex silicon, prototype of a solid-state Qubit

Great application to take advantage of massively parallel GPU accelerated architectures

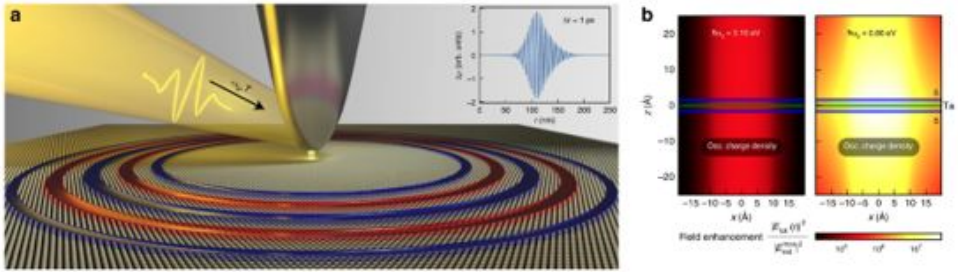
- Acc t: **GW and GW+BSE**
- Unpre simulation sizes: **1000's of atoms**

The BerkeleyGW Software Package

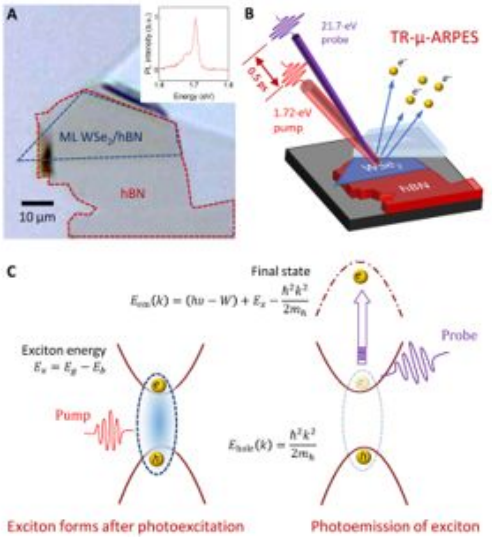
BerkeleyGW

BerkeleyGW is a general purpose software package for studying electron excited-state properties of materials employing the GW and GW plus Bethe-Salpeter equation (BSE)

The BerkeleyGW Software Package



a Plasmon modes are excited in monolayer TaS_2 with an ultrafast laser pulse (energy $\hbar\omega_0 = 1.02$ eV, modulated with a Gaussian profile of width of T = 80



Recent BerkeleyGW highlights:

- Universal slow plasmons and giant field enhancement in atomically thin quasi-two-dimensional metals *Nature Communications* **volume 11**, 1013 (2020)
- Experimental measurement of the intrinsic excitonic wave function *Science Advances* **Vol. 7**, 17, (2021)

Simulation at the highest level of accuracy for: quantum computing, energy storage/conversion, photovoltaics, nanoelectronics, etc...

The Path to Exascale

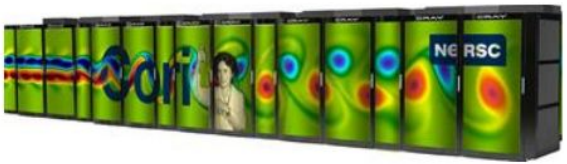
Exascale for the DOE Office of Science
Means: **GPU Accelerated Systems!**

AMD

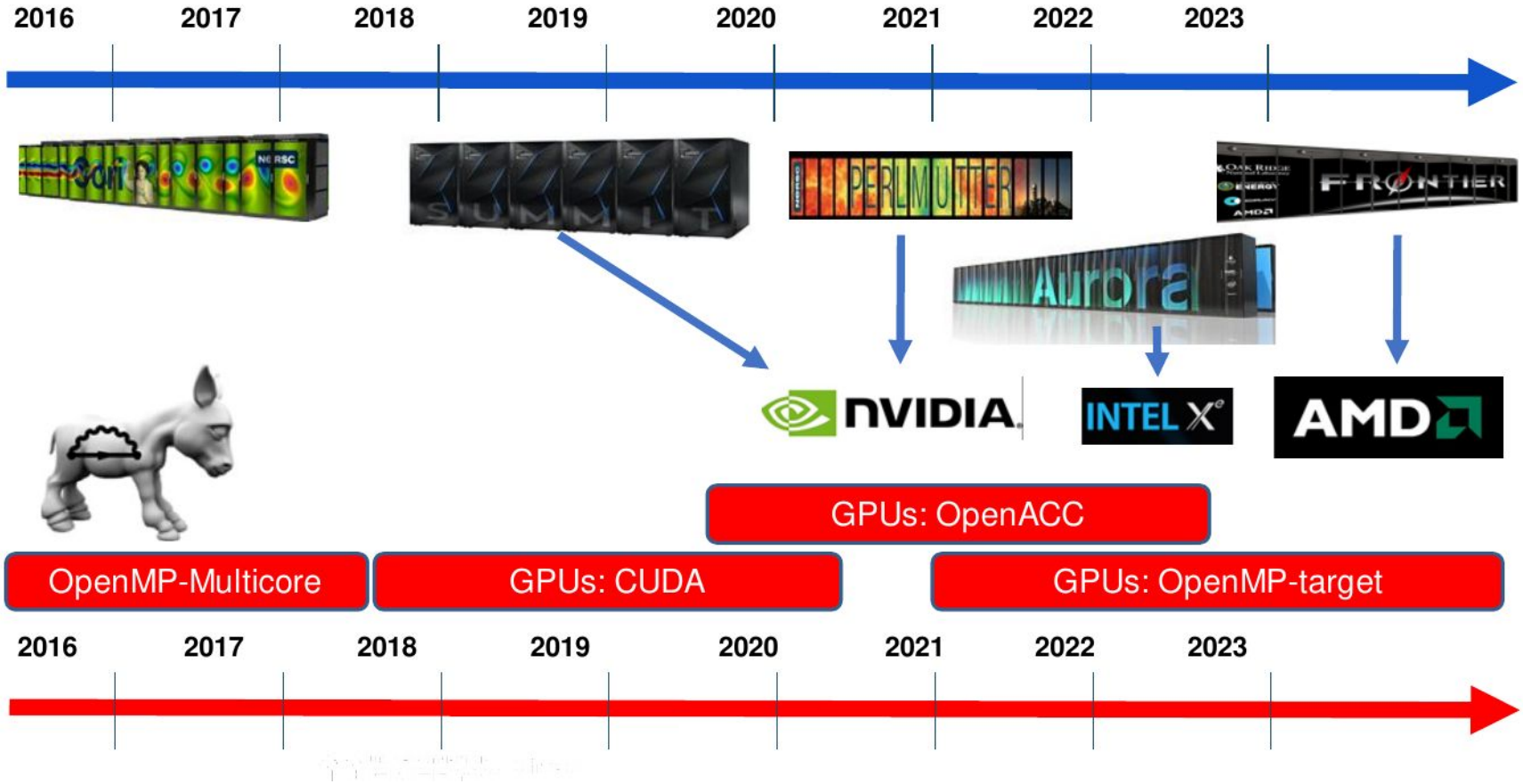
INTEL

NVIDIA

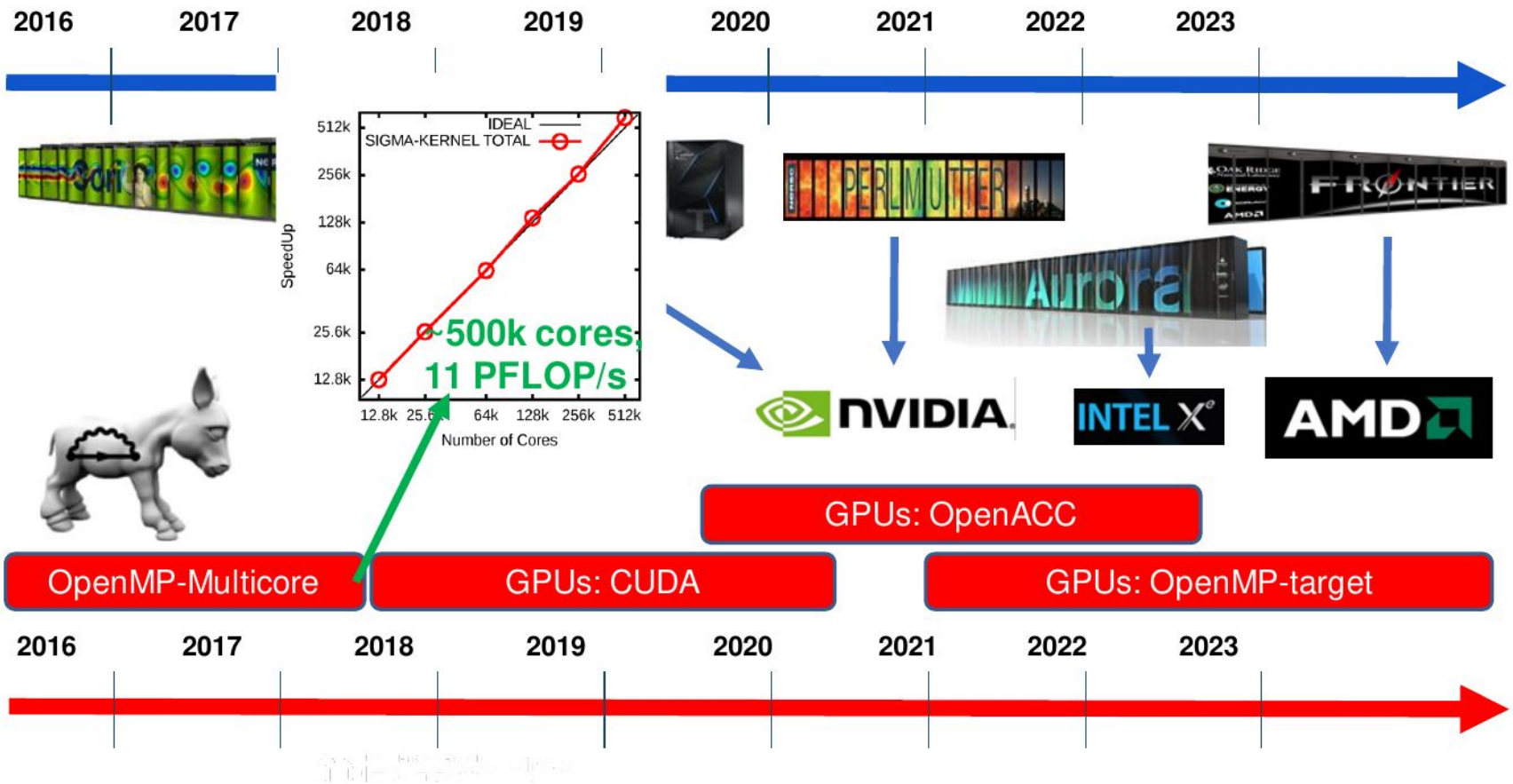
2016 2017 2018 2019 2020 2021 2022 2023



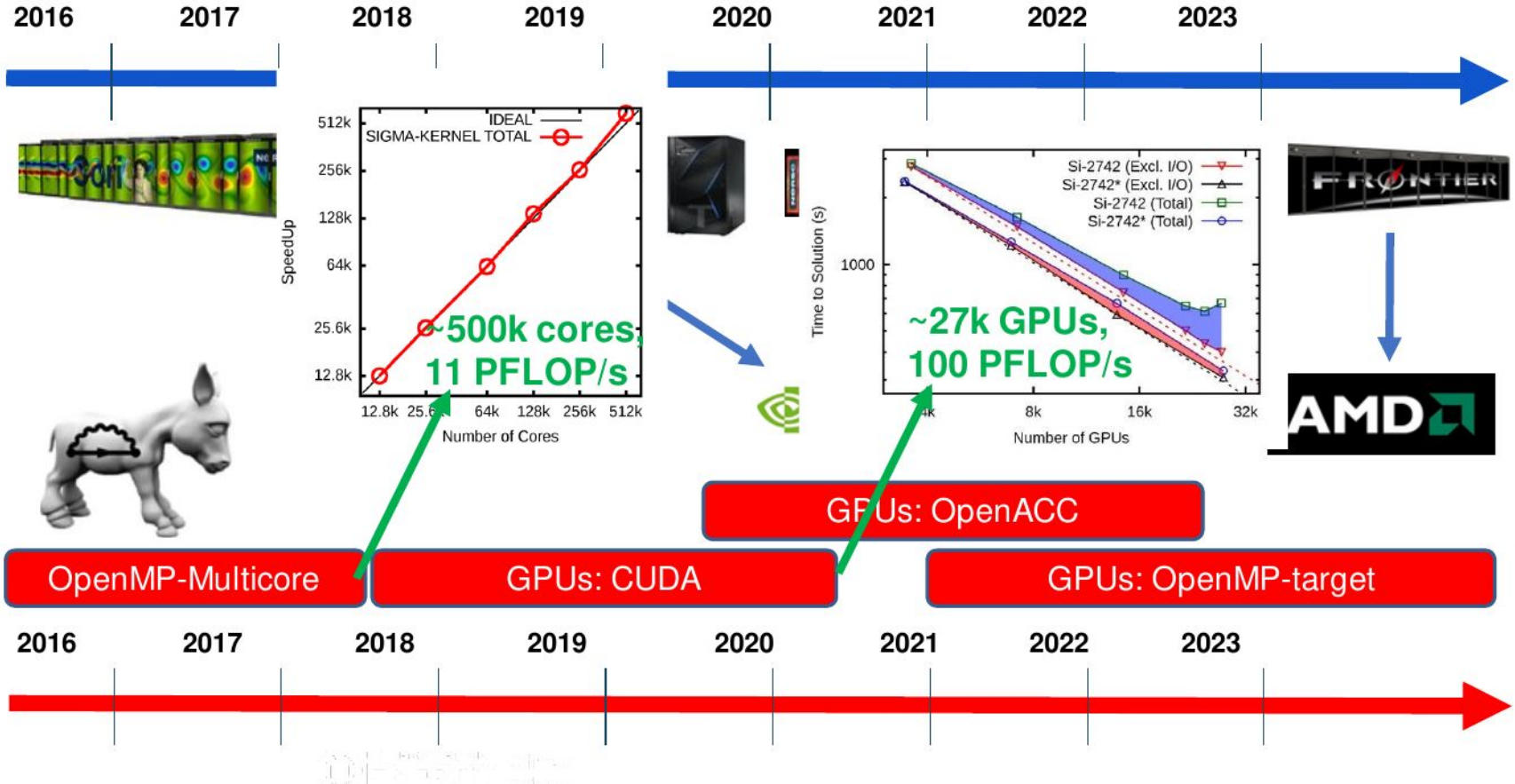
BerkeleyGW on the Path to Exascale



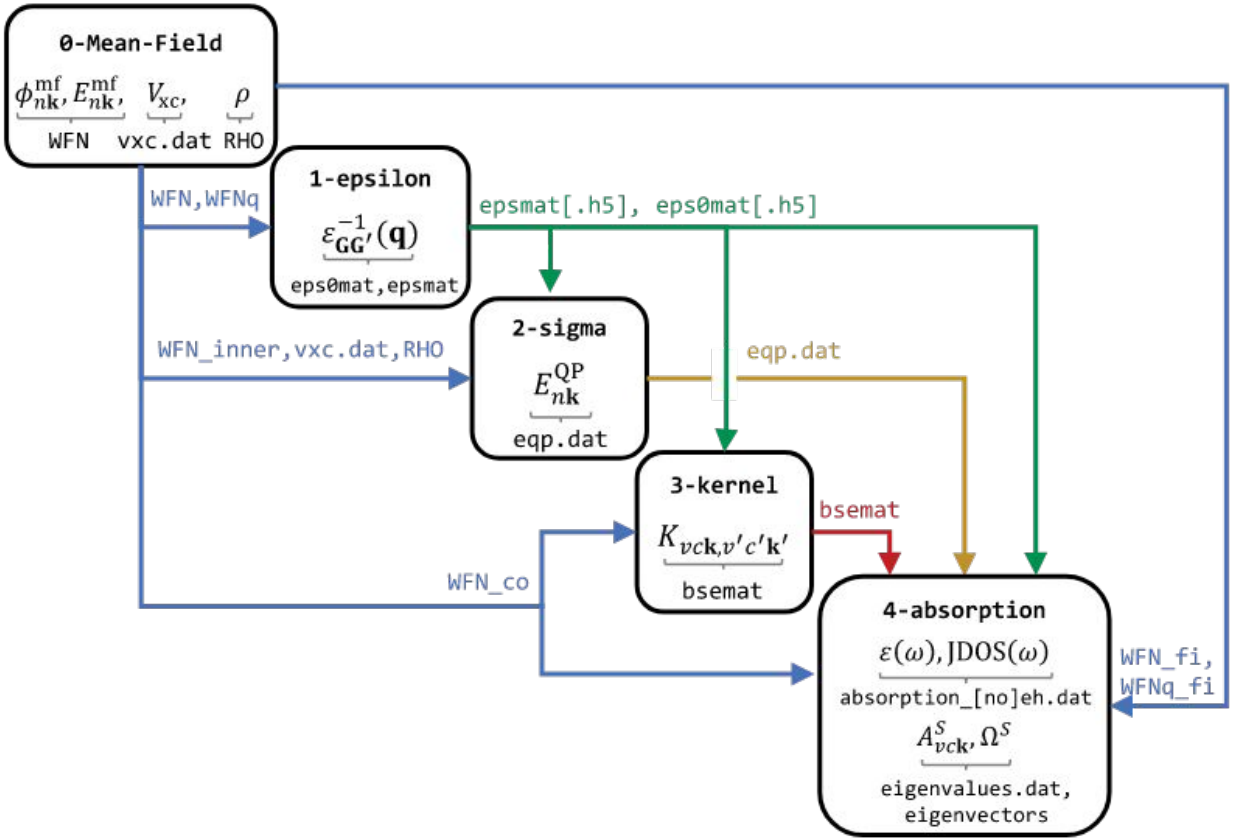
BerkeleyGW on the Path to Exascale



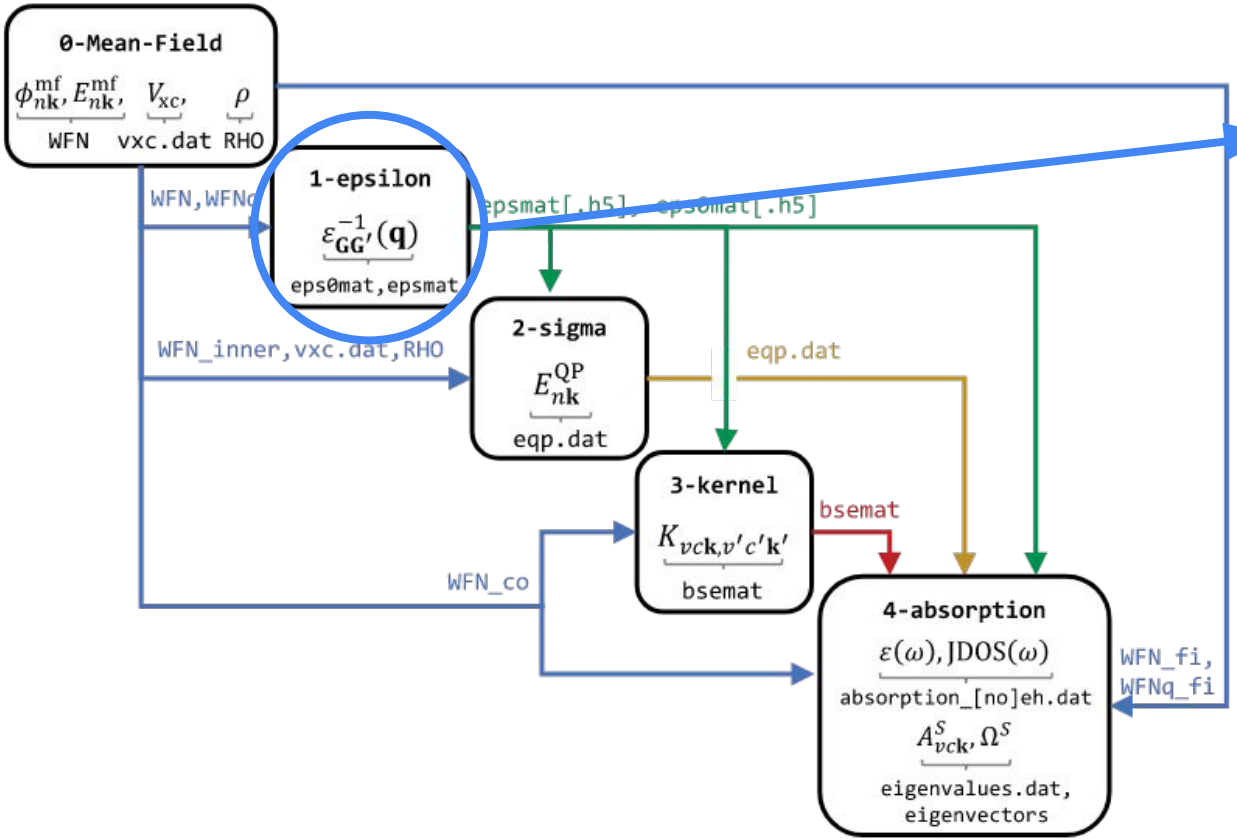
BerkeleyGW on the Path to Exascale



The BerkeleyGW *a software package for studying electron excited-state properties of materials employing the GW, Bethe-Salpeter equation (BSE) and beyond*



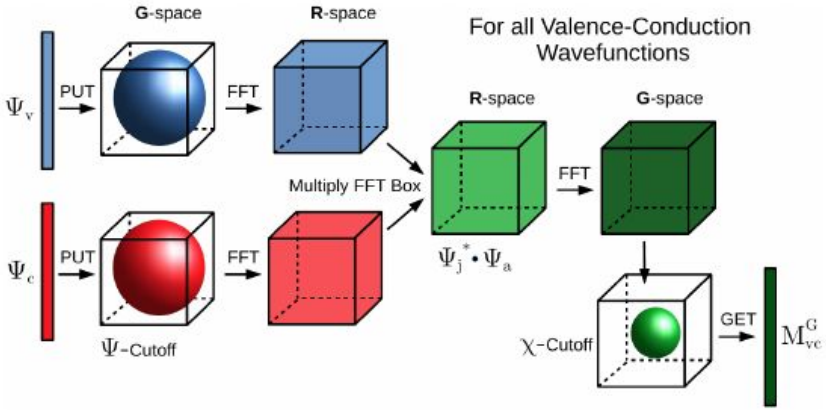
The BerkeleyGW *a software package for studying electron excited-state properties of materials employing the GW, Bethe-Salpeter equation (BSE) and beyond*



OpenACC / OMP-Target Optimization

Epsilon: Computational Kernels

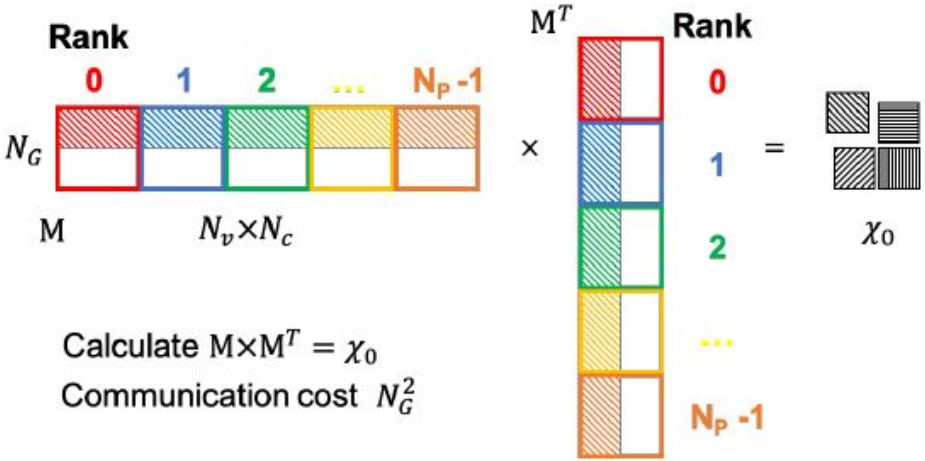
MTXEL



FFT based kernel

Initial implementation:
Single stream no blocking of innermost loop

CHI-0

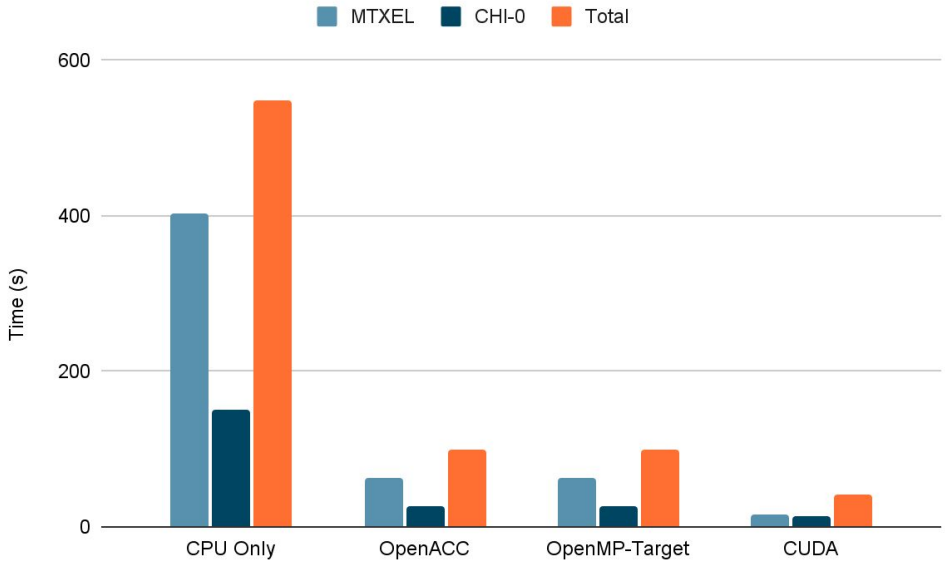


ZGEMM based kernel

Initial implementation:
Simple offload of the submatrices before ZGEMM

Baseline Performance

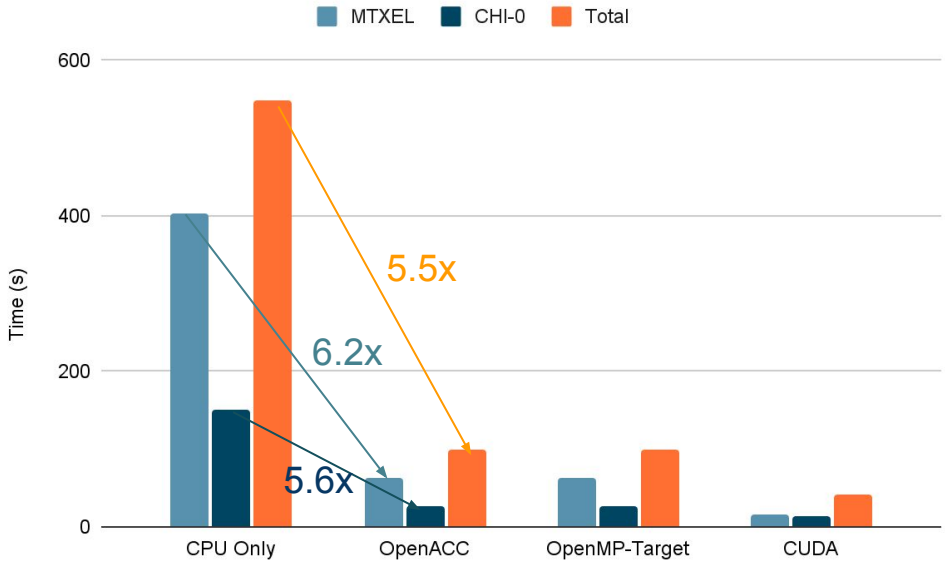
Epsilon: Baseline Performance (Cori-GPU)



	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP-Target	63	27	99
CUDA	15.2	14.7	41

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

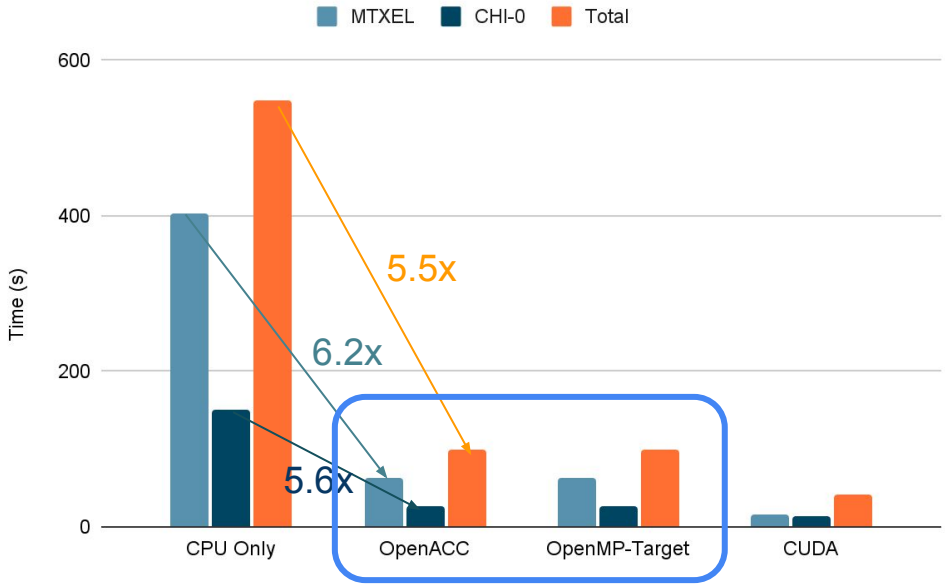
Epsilon: Baseline Performance (Cori-GPU)



	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP-Target	63	27	99
CUDA	15.2	14.7	41

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

Epsilon: Baseline Performance (Cori-GPU)

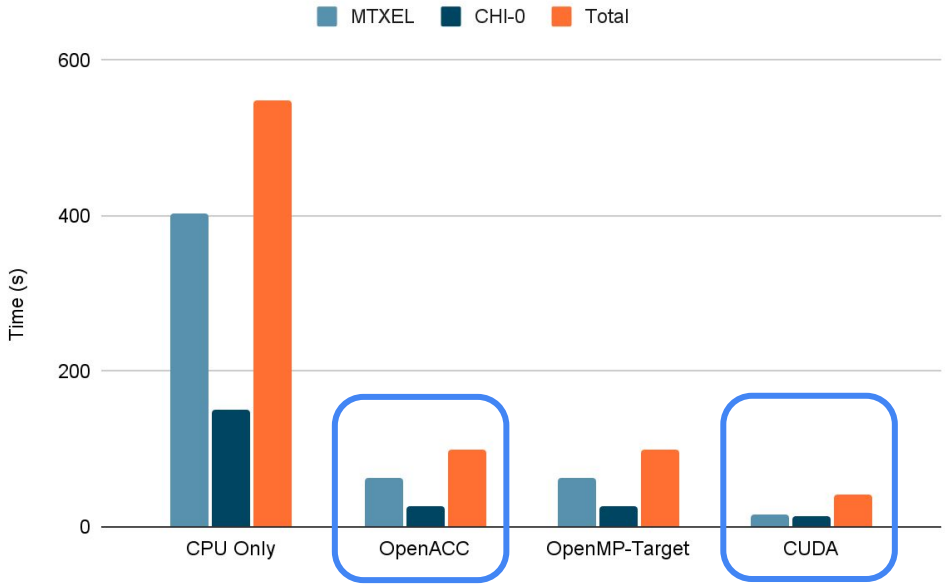


	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP-Target	63	27	99
CUDA	15.2	14.7	41

OpenACC and OMP-target
Identical performance

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

Epsilon: Baseline Performance (Cori-GPU)

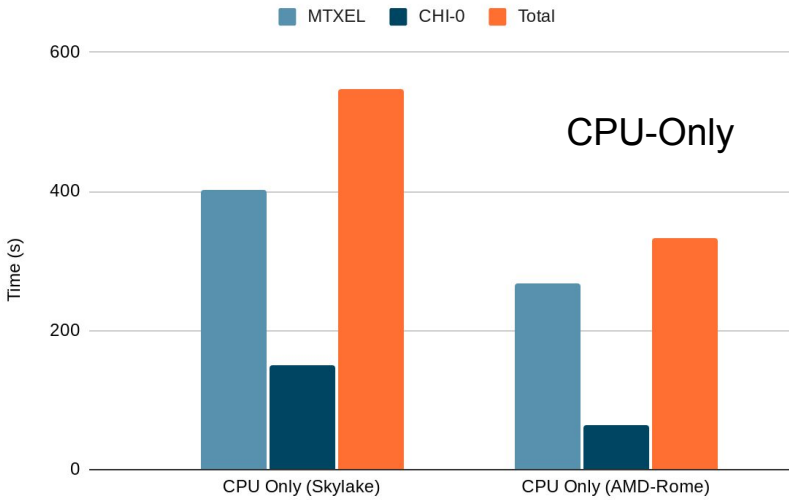


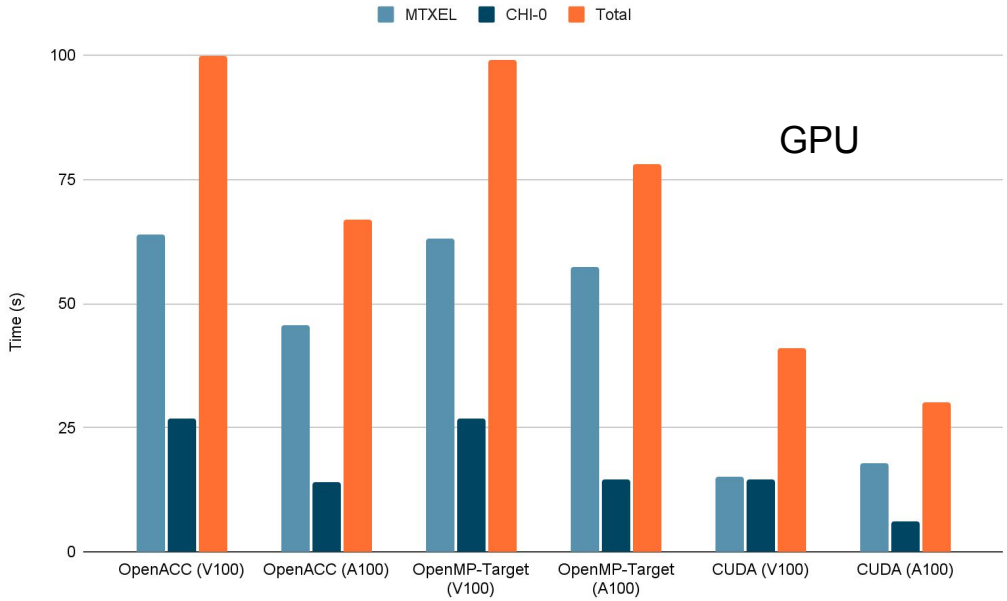
	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP Target	63	27	99
CUDA	15.2	14.7	41

4.2x 1.8x 2.4x

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

Epsilon: Baseline Performance (Cori-GPU vs DGX)

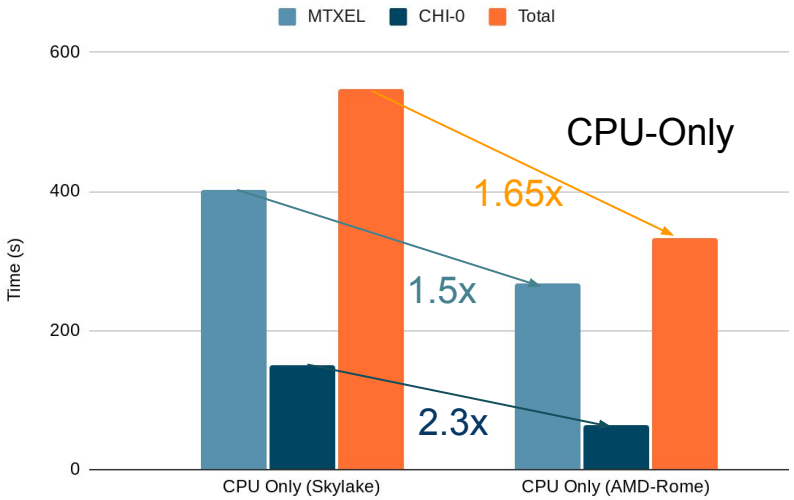


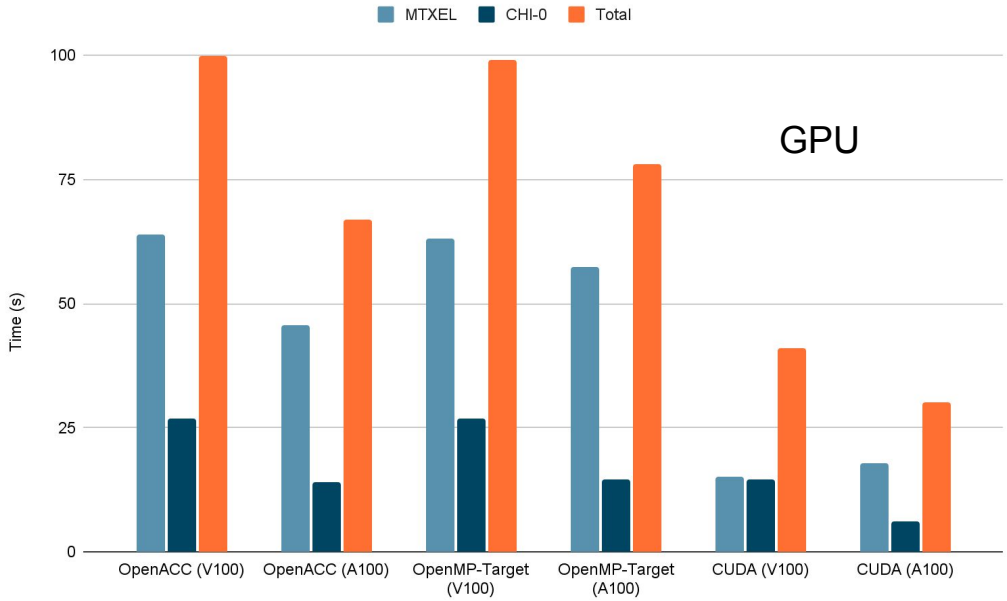


- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Cori-DGX, 1 node (8 A100 GPUs + 2 sockets 64 cores AMD-Rome)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU runs:
  - Cori-GPU (Skylake): 20 MPI tasks each 2 OpenMP threads
  - Cori-DGX: 32 MPI tasks each 4 OpenMP threads
- GPU runs:
  - Cori-GPU: 8 MPI tasks each 1 V100 GPU / 5 OpenMP threads
  - Cori-DGX: 8 MPI tasks each 1 A100 GPU / 16 OpenMP threads

	MTXEL	CHI-0	Total
OpenACC (V100)	64	27	100
OpenACC (A100)	45.8	14	67
OpenMP-Target (V100)	63	27	99
OpenMP-Target (A100)	57.5	14.6	78
CUDA (V100)	15.2	14.7	41
CUDA (A100)	17.9	6.1	30.1

Epsilon: Baseline Performance (Cori-GPU vs DGX)

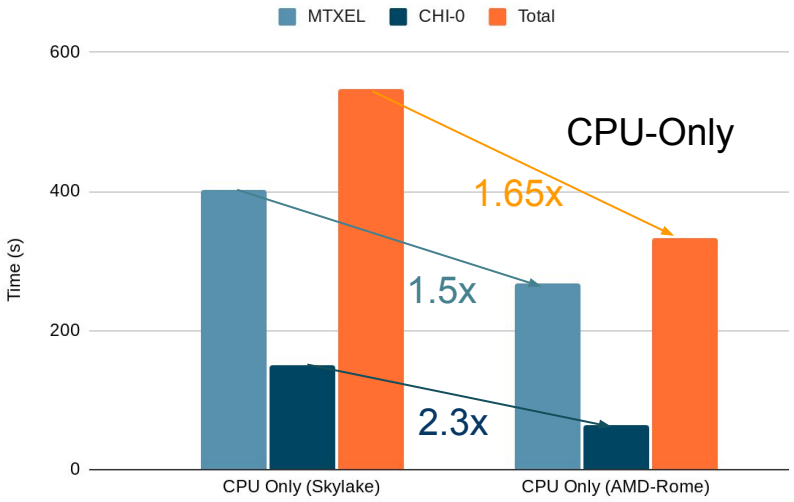


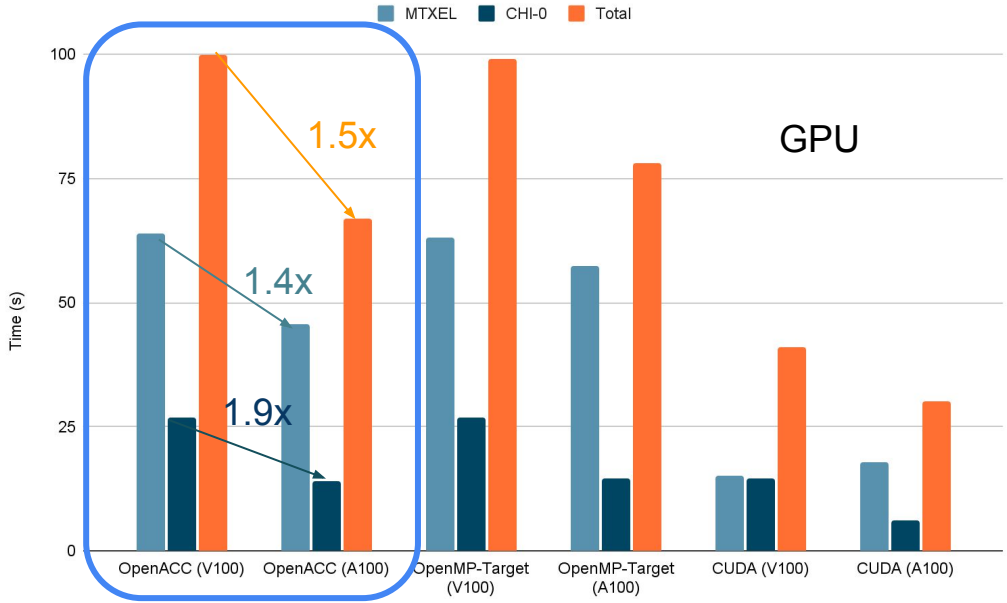


- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Cori-DGX, 1 node (8 A100 GPUs + 2 sockets 64 cores AMD-Rome)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU runs:
  - Cori-GPU (Skylake): 20 MPI tasks each 2 OpenMP threads
  - Cori-DGX: 32 MPI tasks each 4 OpenMP threads
- GPU runs:
  - Cori-GPU: 8 MPI tasks each 1 V100 GPU / 5 OpenMP threads
  - Cori-DGX: 8 MPI tasks each 1 A100 GPU / 16 OpenMP threads

	MTXEL	CHI-0	Total
OpenACC (V100)	64	27	100
OpenACC (A100)	45.8	14	67
OpenMP-Target (V100)	63	27	99
OpenMP-Target (A100)	57.5	14.6	78
CUDA (V100)	15.2	14.7	41
CUDA (A100)	17.9	6.1	30.1

Epsilon: Baseline Performance (Cori-GPU vs DGX)

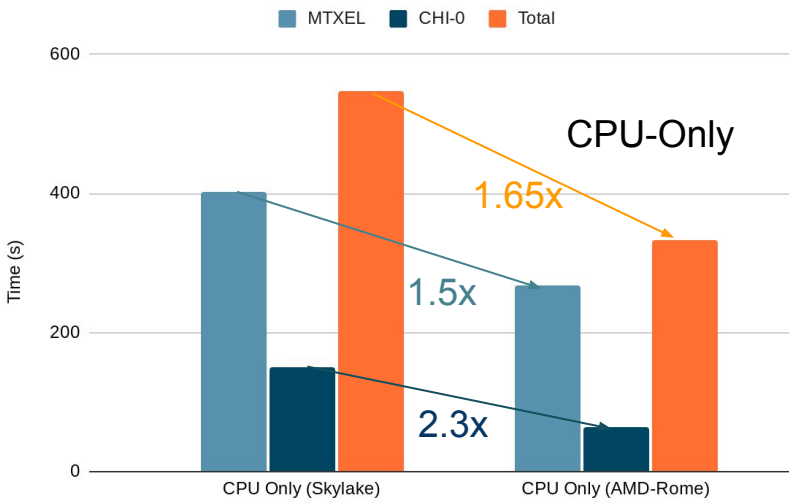


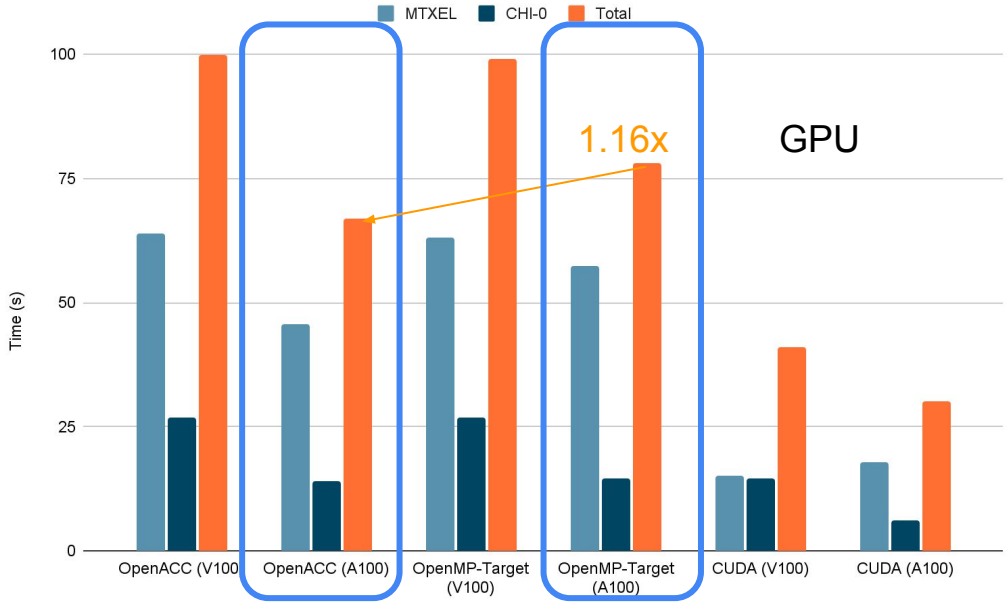


- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Cori-DGX, 1 node (8 A100 GPUs + 2 sockets 64 cores AMD-Rome)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU runs:
  - Cori-GPU (Skylake): 20 MPI tasks each 2 OpenMP threads
  - Cori-DGX: 32 MPI tasks each 4 OpenMP threads
- GPU runs:
  - Cori-GPU: 8 MPI tasks each 1 V100 GPU / 5 OpenMP threads
  - Cori-DGX: 8 MPI tasks each 1 A100 GPU / 16 OpenMP threads

	MTXEL	CHI-0	Total
OpenACC (V100)	64	27	100
OpenACC (A100)	45.8	14	67
OpenMP-Target (V100)	63	27	99
OpenMP-Target (A100)	57.5	14.6	78
CUDA (V100)	15.2	14.7	41
CUDA (A100)	17.9	6.1	30.1

Epsilon: Baseline Performance (Cori-GPU vs DGX)

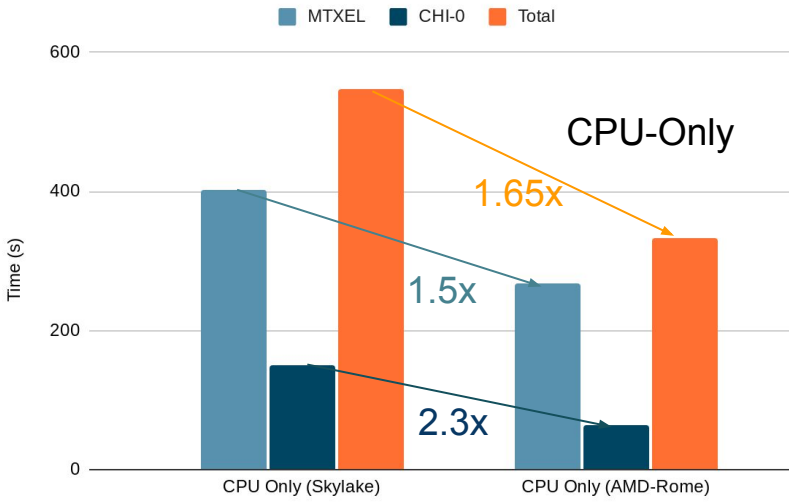


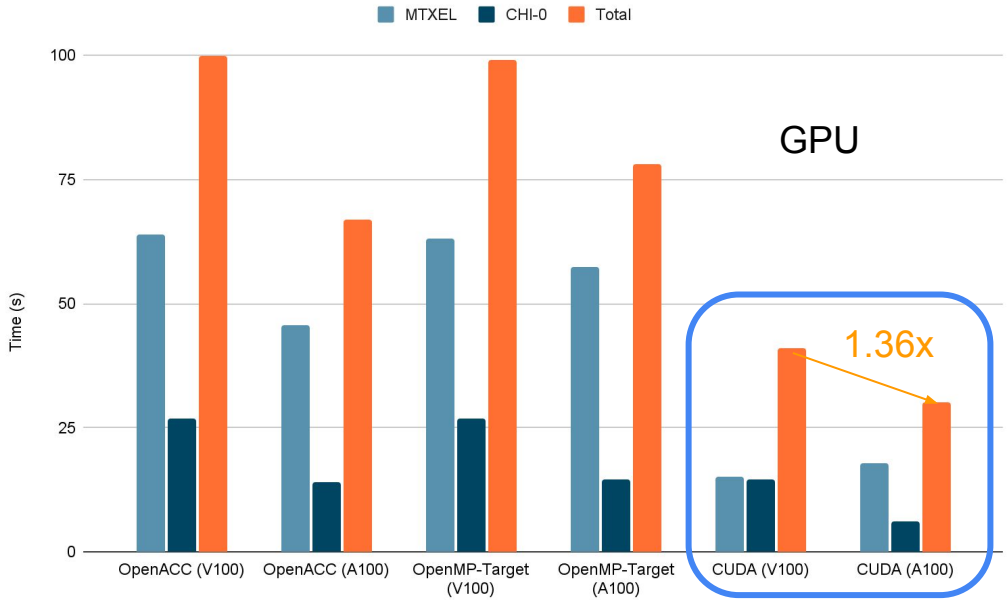


- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Cori-DGX, 1 node (8 A100 GPUs + 2 sockets 64 cores AMD-Rome)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU runs:
  - Cori-GPU (Skylake): 20 MPI tasks each 2 OpenMP threads
  - Cori-DGX: 32 MPI tasks each 4 OpenMP threads
- GPU runs:
  - Cori-GPU: 8 MPI tasks each 1 V100 GPU / 5 OpenMP threads
  - Cori-DGX: 8 MPI tasks each 1 A100 GPU / 16 OpenMP threads

	MTXEL	CHI-0	Total
OpenACC (V100)	64	27	100
OpenACC (A100)	45.8	14	67
OpenMP-Target (V100)	63	27	99
OpenMP-Target (A100)	57.5	14.6	78
CUDA (V100)	15.2	14.7	41
CUDA (A100)	17.9	6.1	30.1

Epsilon: Baseline Performance (Cori-GPU vs DGX)



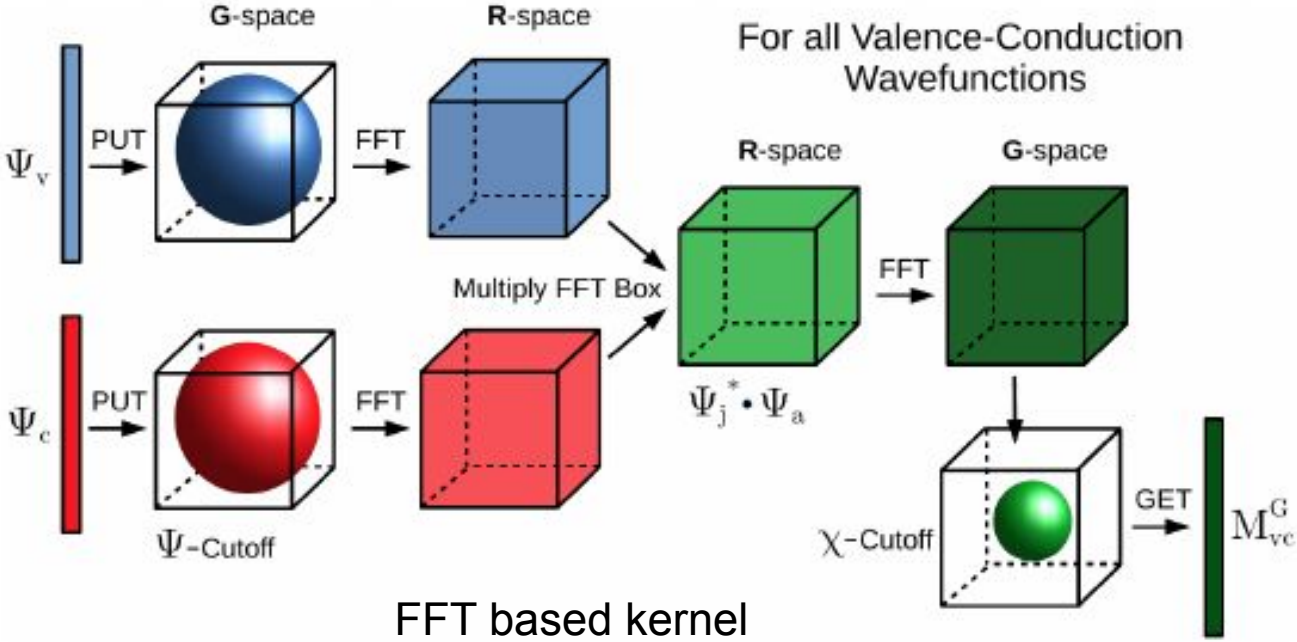


- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Cori-DGX, 1 node (8 A100 GPUs + 2 sockets 64 cores AMD-Rome)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU runs:
  - Cori-GPU (Skylake): 20 MPI tasks each 2 OpenMP threads
  - Cori-DGX: 32 MPI tasks each 4 OpenMP threads
- GPU runs:
  - Cori-GPU: 8 MPI tasks each 1 V100 GPU / 5 OpenMP threads
  - Cori-DGX: 8 MPI tasks each 1 A100 GPU / 16 OpenMP threads

	MTXEL	CHI-0	Total
OpenACC (V100)	64	27	100
OpenACC (A100)	45.8	14	67
OpenMP-Target (V100)	63	27	99
OpenMP-Target (A100)	57.5	14.6	78
CUDA (V100)	15.2	14.7	41
CUDA (A100)	17.9	6.1	30.1

MTXEL Optimization

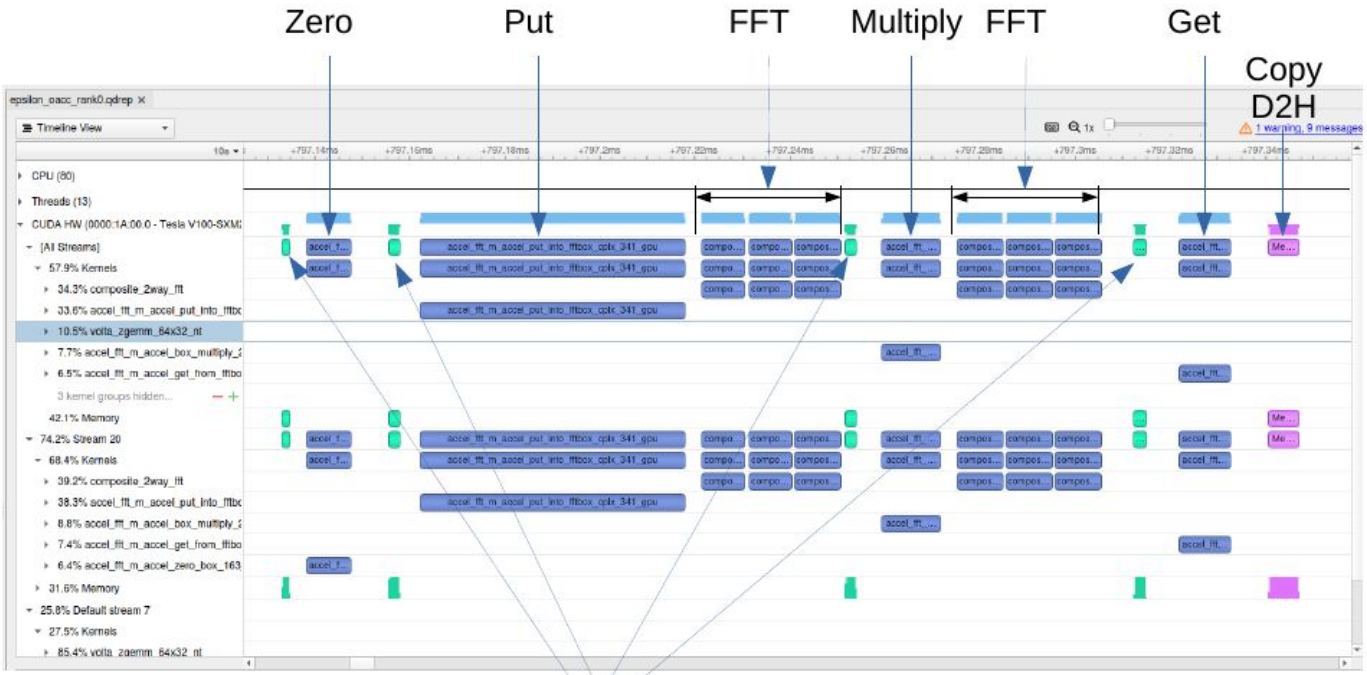
Epsilon: MTXEL Kernel



FFT based kernel

Initial port: Single stream no loop blocking of innermost loop

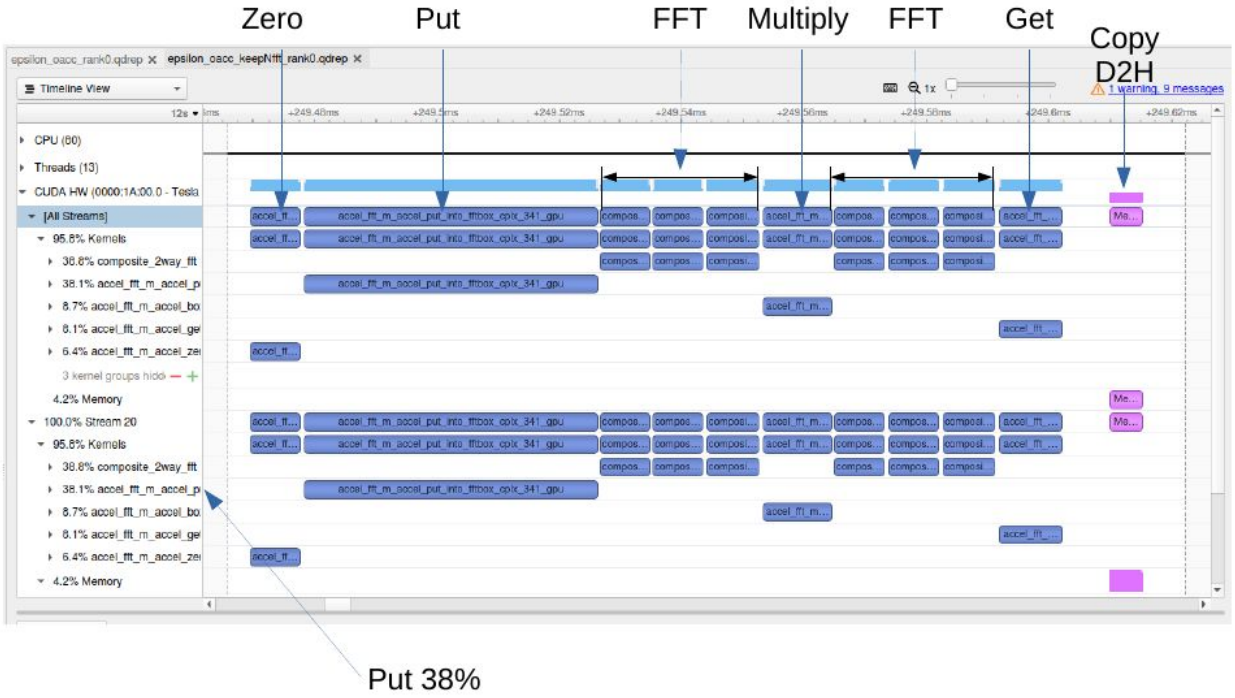
MTXEL: Baseline (64s)



12 Bytes, I think this is coping in the Nfft array (integer array with 3 elements) before running the kernels

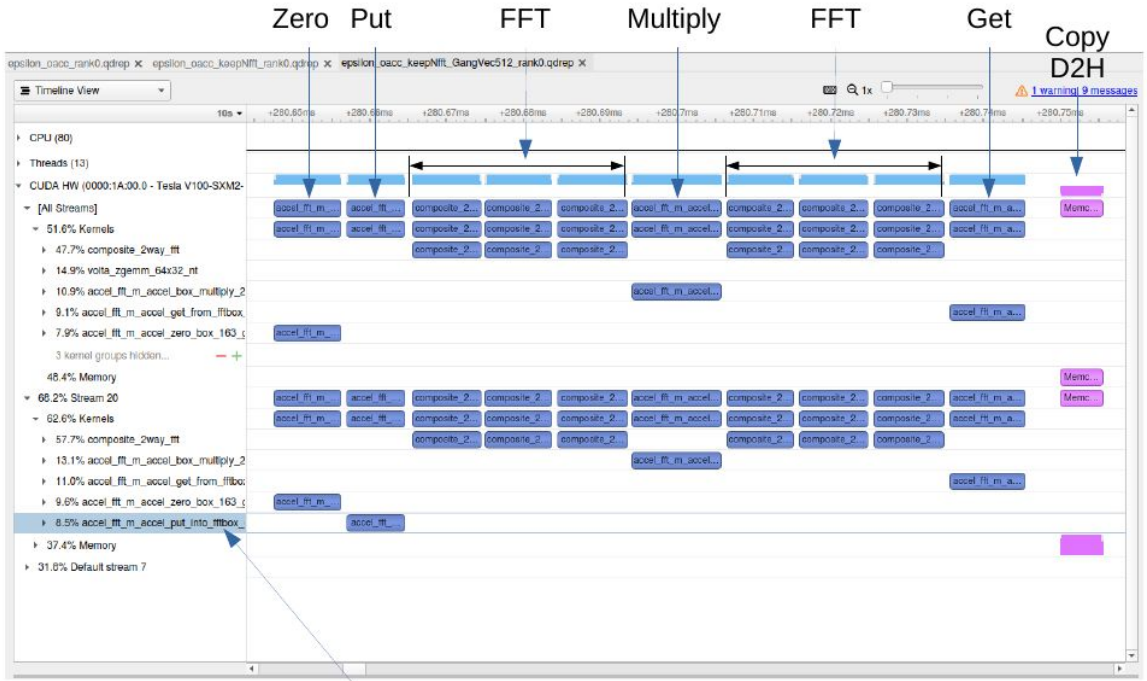
MTXEL: Move small arrays into GPU memory (58s)

Copy in Nfft array before kernel execution



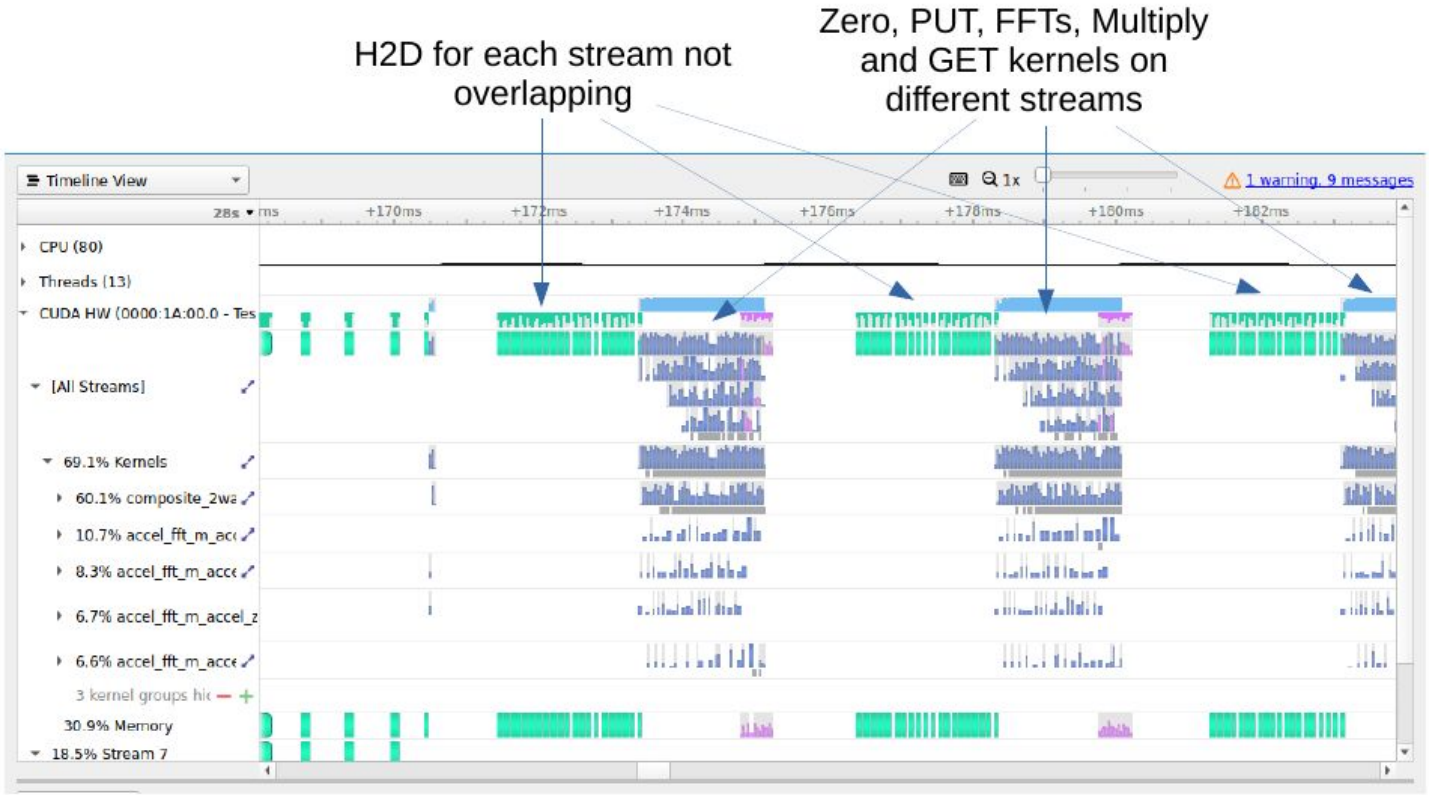
MTXEL: Improve Kernel Performance (53s)

For Put Use: !$acc parallel vector_length(512) loop gang vector
Instead of: !$acc parallel loop

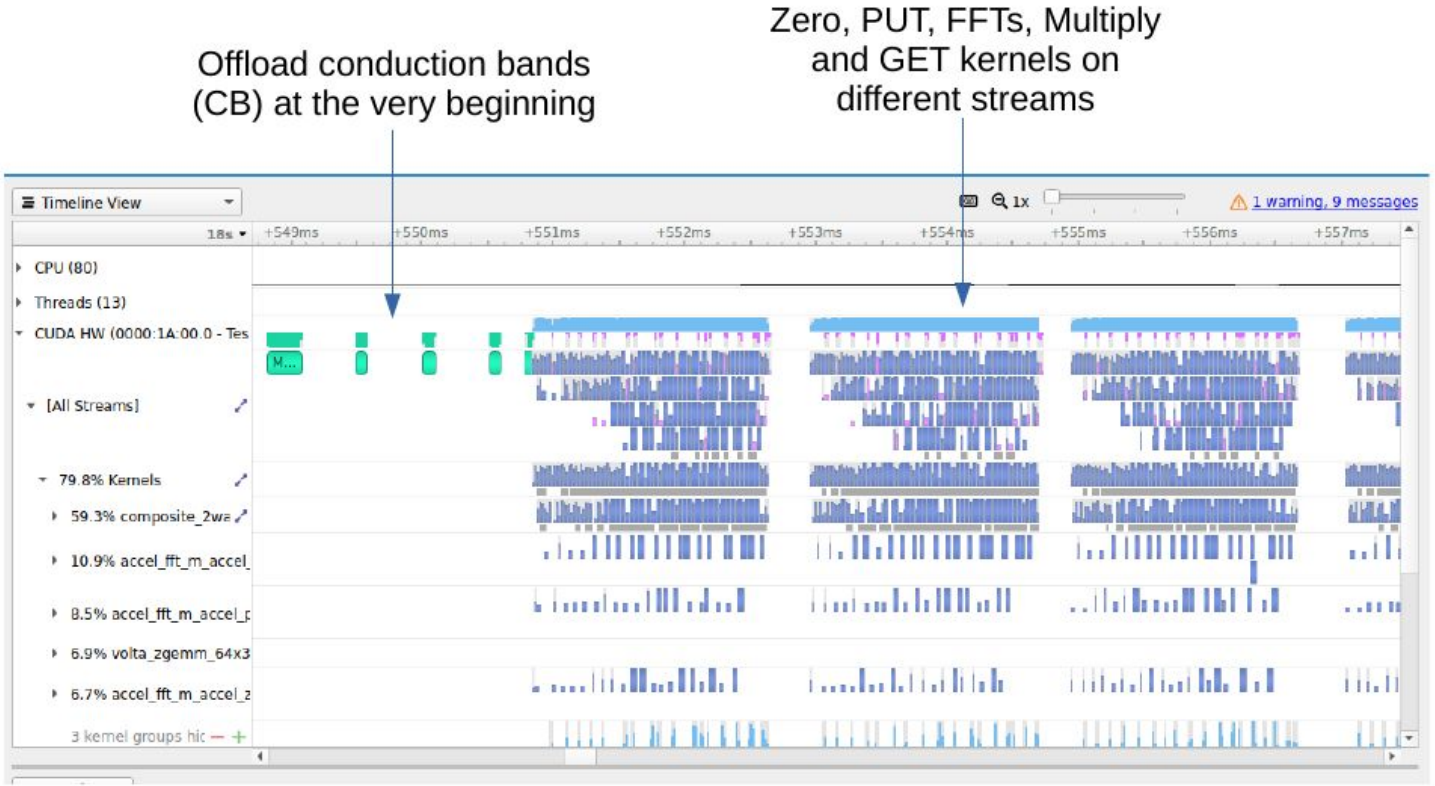


Using vector_length(512) loop gang vector construct for all kernel slightly improve performance 52.1 vs 52.6s

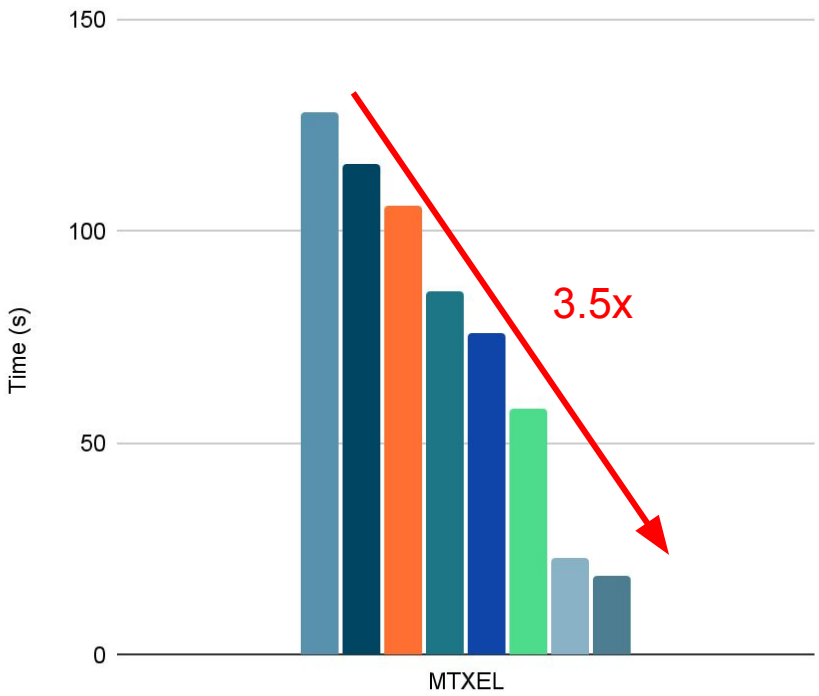
MTXEL: Introduce Streams (43s)



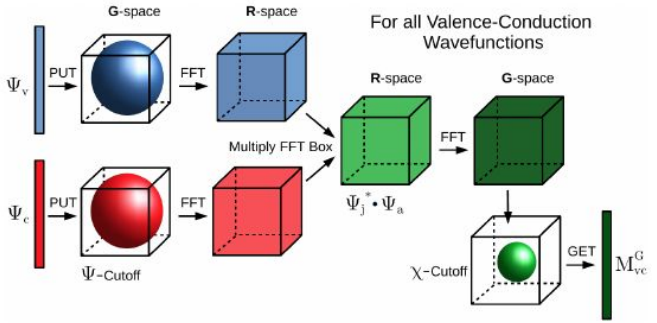
MTXEL: Streams + Offload CB (18.5s)



MTXEL: Summary of Optimization (OpenACC)



- Baseline
- Move Nfft Array to device
- Use "loop gang vector vector_length(512)" for PUT, Mult and Get kernels
- Introduce Streamed FFT
- Introduce Batched FFT
- Batched FFT with offload of CB
- Batched with single plan creation
- Streamed with offload of CB



Time in seconds

	MTXEL
Baseline	64
Move Nfft Array to device	58
Use "loop gang vector vector_length(512)" for PUT, Mult and Get kernels	53
Introduce Streamed FFT	43
Introduce Batched FFT	38
Batched FFT with offload of CB	29
Batched with single plan creation	23
Streamed with offload of CB	18.5

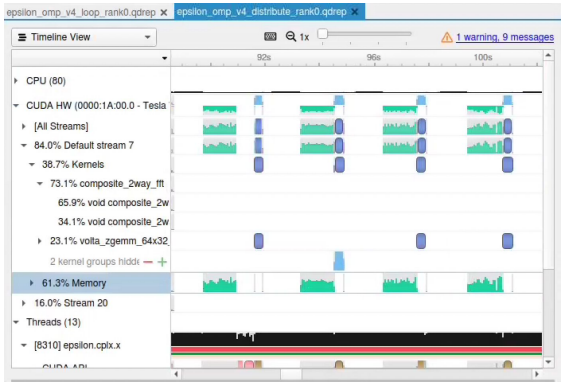
MTXEL: OpenMP-Target

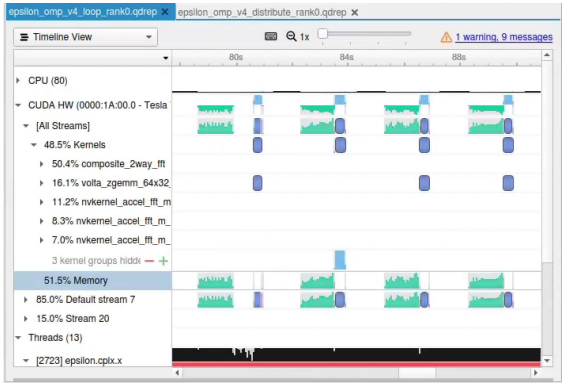
Before: 63 sec

```
!$omp target teams distribute parallel do collapse(3)
      do iz = 1, Nfft(3)
         do iy = 1, Nfft(2)
            do ix = 1, Nfft(1)
     box_out(ix,iy,iz) = box_in(ix,iy,iz) * box_out(ix,iy,iz)
            enddo
         enddo
      Enddo
!$omp end target teams distribute parallel do
```

After: 56 sec, >10% speedup

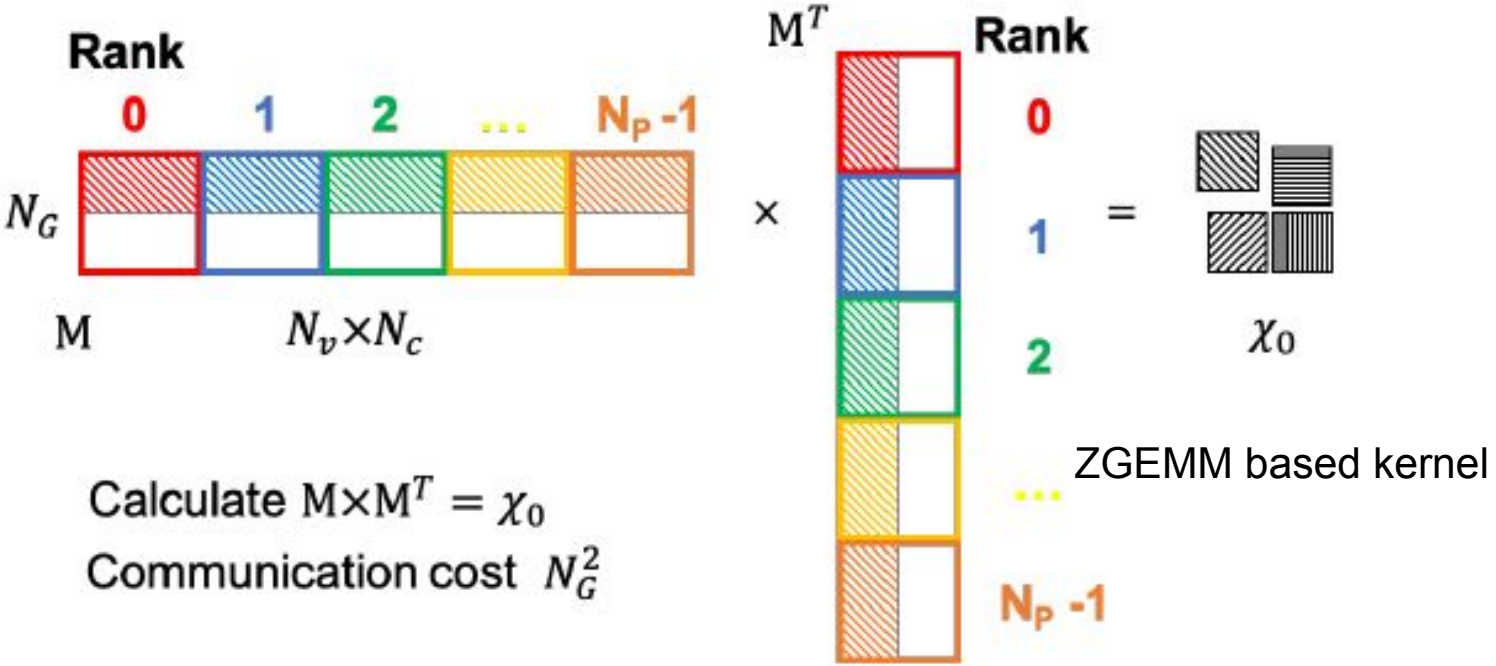
```
!$omp target teams loop thread_limit(128) map(to:Nfft) collapse(3)
      do iz = 1, Nfft(3)
         do iy = 1, Nfft(2)
            do ix = 1, Nfft(1)
            box_out(ix,iy,iz) = box_in(ix,iy,iz) * box_out(ix,iy,iz)
            enddo
         enddo
      enddo
!$omp end target teams loop
```





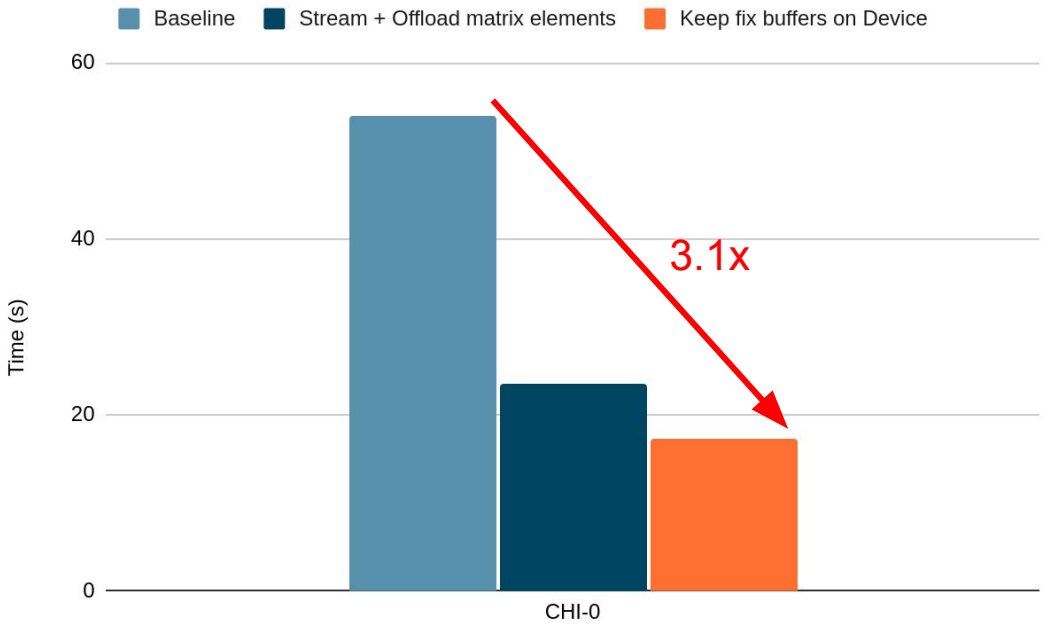
CHI-0 Optimization

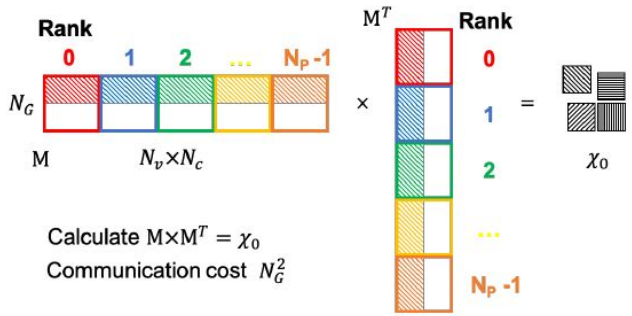
Epsilon: CHI-0 Kernel



Initial port: Simple offload of the submatrices before ZGEMM

CHI-0: Summary of Optimization (OpenACC)





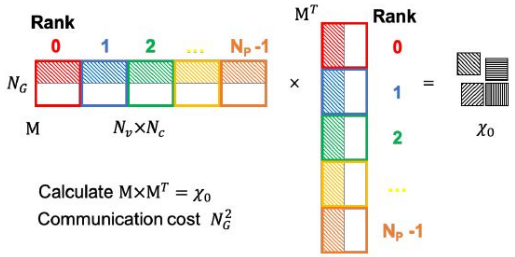
Time in seconds

	CHI-0
Baseline	27
Stream + Offload matrix elements	11.8
Keep fix buffers on Device	8.6

CHI-0: Summary of Optimization (OpenACC)

Copy all data at start

```
313 +       nrow_max = MAXVAL(scal%nprd)
314 +       ncol_max = MAXVAL(scal%npcd)
315 +       SAFE_ALLOCATE(this%chilocal, (nrow_max, ncol_max))
316 +       SAFE_ALLOCATE(this%gmetempr, (nrow_max, ntot))
317 +       SAFE_ALLOCATE(this%gmetempc, (ncol_max, ntot))
318 +
319 +       if ( chi_summation_algo /= CPU_ALGO ) then
320 +          !$acc enter data copyin(pol) async(1)
321 +          !$acc enter data copyin(pol%gme) async(1)
322 +          !$acc enter data copyin(scal) async(1)
323 +          !$acc enter data copyin(scal%nprd, scal%npcd, scal%imyrowd, scal%imycold) async(1)
324 +          !$acc enter data copyin(indt, pht) async(1)
325 +          !$acc enter data copyin(peinf) async(1)
326 +          !$acc enter data create(this) async(1)
327 +          !$acc enter data create(this%chilocal, this%gmetempr, this%gmetempc) async(1)
328 +       end if
329 +
```



Re-use cublas handle

```
152  228  #ifdef OPENACC
153       -      cuErr = cublasCreate(cublas_handle)
154       -      !$acc host_data use_device(a, b, c)
155       -      call cublasZgemm(transa, transb, &
156       -                       m, n, k, &
157       -                       alpha, a, lda, b, ldb, &
                                 beta, c, ldc)
             !$acc end host_data
             cuErr = cublasDestroy(cublas_handle)
             if ( common_blas_handle_exist ) then
               !$acc host_data use_device(a, b, c)
               cuErr = cublasZgemm_v2(cublas_handle_common, &
                                      cublasConvertCharToV2(transa), cublasConvertCharToV2(transb), &
                                      m, n, k, &
                                      alpha, a, lda, b, ldb, &
                                      beta, c, ldc)
               !$acc end host_data
             else
               cuErr = cublasCreate(cublas_handle)
               !$acc host_data use_device(a, b, c)
240  +        call cublasZgemm(transa, transb, &
```

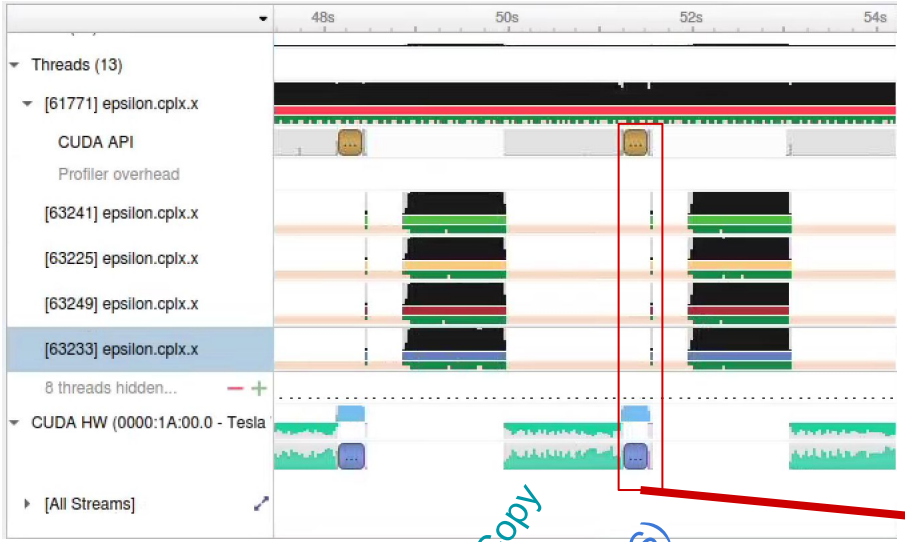
```
!$acc parallel private(icurr) present(scal%npcd, indt, pht, scal%imycold, scal%imycold, tmpcolindex, tmpcolph) &
!$acc& async(1) vector_length(512)
!$acc loop gang vector
do icurr=1,scal%npcd(ipe)
  tmpcolindex(icurr) = indt(scal%imycold(icurr,ipe),it,irk)
  tmpcolph(icurr) = pht(scal%imycold(icurr,ipe),it,irk)
enddo
!$acc end parallel

!$acc parallel private(iv, j, mytot, icurr) present(peinf, scal%nprd, scal%npcd, gmetempr, gmetempc, polgme, &
!$acc& tmprowindex, tmpcolindex, tmprowph, tmpcolph) &
!$acc& async(1) vector_length(512)
!$acc loop gang vector collapse(2)
do iv = 1,peinf%nvownactual
  do j = 1, peinf%ncownactual

    mytot = itot + (iv-1)*peinf%ncownactual + j
```

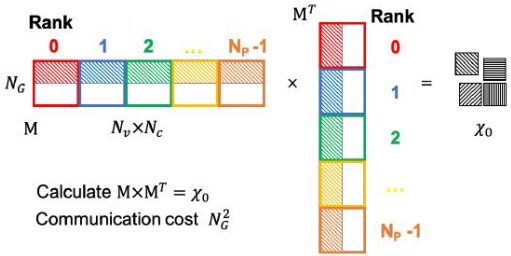
New OpenACC kernel for data prep on GPU

CHI-0: Summary of Optimization (OpenACC)

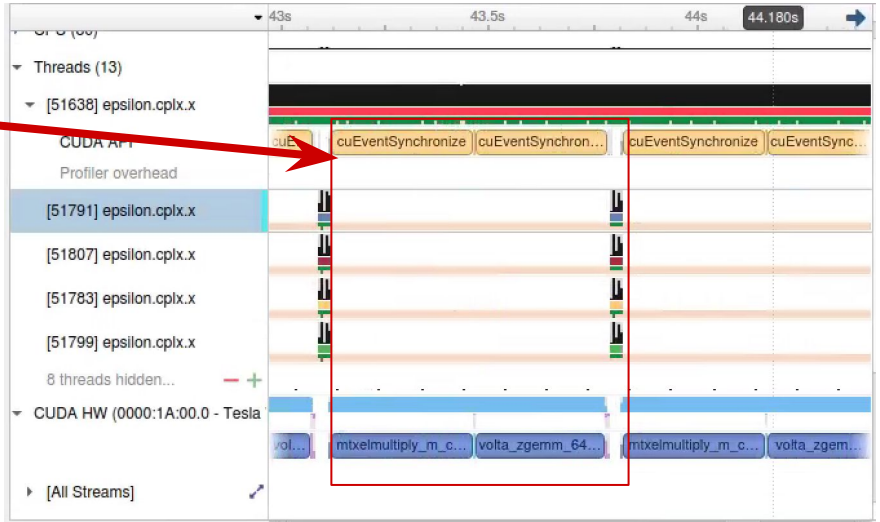


Host 2 Device Copy

ZGEMM (cuBLAS)

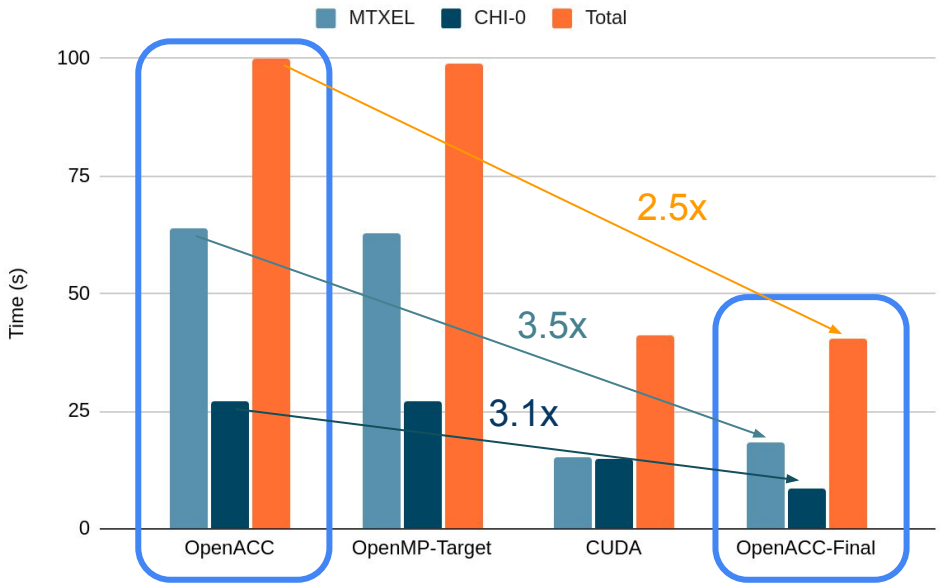


After



Summary

Epsilon: Final Results (Cori-GPU)



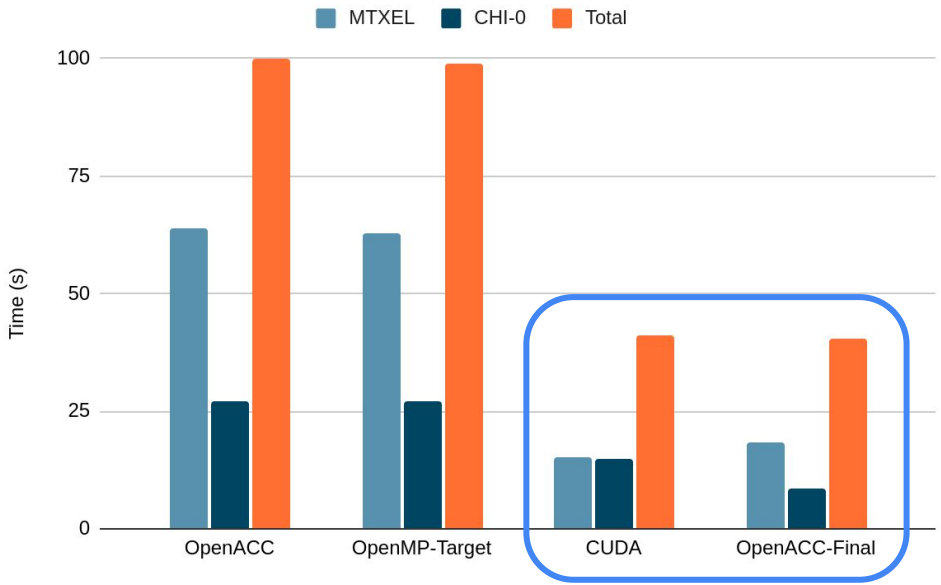
Time in seconds

	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP-Target	63	27	99
CUDA	15.2	14.7	41
OpenACC-Final	18.5	8.6	40.4

Most of the hackathon focused on OpenACC implementation: overall 2.5x speedup compared to pre-hackathon version

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

Epsilon: Final Results (Cori-GPU)



Time in seconds

	MTXEL	CHI-0	Total
CPU Only	402	150	548
OpenACC	64	27	100
OpenMP-Target	63	27	99
CUDA	15.2	14.7	41
OpenACC-Final	18.5	8.6	40.4

OpenACC implementation comparable performance to the CUDA implemenattion

- Cori-GPU, 1 node (8 V100 GPUs + 2 sockets 20 cores Skylake)
- Si-214 system (scaled: 4Ry CT ; 3000 bands)
- CPU run, 20 MPI tasks each 2 OpenMP threads
- GPU runs, 8 MPI tasks each 1 GPU / 5 OpenMP threads

Special Thanks:

- Helen He (LBNL)
- David Rogers (ORNL)
- Robert Searles (NVIDIA)
- Brent Leback (NVIDIA)

Summary

- OpenACC optimization of the two major kernel of epsilon
  - MTXEL: Streams and Batch-FFT both implemented with similar performance
  - CHI-0: Offload on matrix elements + streams + overlap MPI communication and ZGEMM on GPU
- Overall 2.5x speed-up for entire execution
  - MTXEL: 3.5x speed-up
  - CHI-0: 3.1x speed-up
- The OpenACC implementation achieve similar performance as the optimized CUDA implementation

NEXT:

- Implement blocking of matrix elements for CHI-0 to avoid hitting memory limit on GPU
- Achieve similar performance for the OpenMP-target implementation

Backup

MTXEL OpenACC, Original

```
! copy in arrays
accel_mtxel_eps%gvec_comp = gvec%components
call accel_update_to(accel_mtxel_eps%gvec_comp, mtxel_algo)
accel_mtxel_eps%outer_sort = vwfn%isort
call accel_update_to(accel_mtxel_eps%outer_sort, mtxel_algo)
accel_mtxel_eps%inner_sort = cwfn%isort
call accel_update_to(accel_mtxel_eps%inner_sort, mtxel_algo)
accel_mtxel_eps%result_sort = pol%isrtx
call accel_update_to(accel_mtxel_eps%result_sort, mtxel_algo)

do jsp = ispin, ispin * kp%nspinor
  accel_mtxel_eps%outer_bands = ZERO
  accel_mtxel_eps%outer_bands(1:vwfn%ngv) = vwfn%zv(1:vwfn%ngv, jsp)
  !$acc update device(accel_mtxel_eps%outer_bands) async(1)
  accel_mtxel_eps%inner_bands = ZERO
  accel_mtxel_eps%inner_bands(1:peinf%ncownactual * cwfn%ngc) = cwfn%zc(1:peinf%ncownactual * cwfn%ngc, jsp)
  !$acc update device(accel_mtxel_eps%inner_bands) async(1)

  call accel_zero_box(accel_mtxel_eps%outer_fftbox, Nfft, mtxel_algo, queue=1)
  call accel_put_into_fftbox(vwfn%ngv, accel_mtxel_eps%outer_bands, accel_mtxel_eps%gvec_comp, &
                             accel_mtxel_eps%outer_sort, accel_mtxel_eps%outer_fftbox, &
                             Nfft, 1.0D+00, mtxel_algo, queue=1)
  call accel_run_fft(accel_mtxel_eps%outer_fftbox, accel_mtxel_eps%outer_plan, 1, mtxel_algo)

  do ic_loc = 1, peinf%ncownactual
    ic_FE = peinf%invindexc(ic_loc)
    ic = vwfn%nband + ic_FE
    offset_g = (ic_loc-1)*cwfn%ngc

    call accel_zero_box(accel_mtxel_eps%inner_fftbox, Nfft, mtxel_algo, queue=1)
    call accel_put_into_fftbox(cwfn%ngc, accel_mtxel_eps%inner_bands(offset_g+1:offset_g+cwfn%ngc), accel_mtxel_eps%gvec_comp, &
                               accel_mtxel_eps%inner_sort, accel_mtxel_eps%inner_fftbox, Nfft, &
                               1.0D+00, mtxel_algo, queue=1)
    call accel_run_fft(accel_mtxel_eps%inner_fftbox, accel_mtxel_eps%outer_plan, 1, mtxel_algo)
    call accel_box_multiply(accel_mtxel_eps%inner_fftbox, &
                            accel_mtxel_eps%outer_fftbox, Nfft, 1, mtxel_algo, queue=1)

    call accel_run_fft(accel_mtxel_eps%inner_fftbox, accel_mtxel_eps%outer_plan, 1, mtxel_algo)
    call accel_get_from_fftbox(pol%nmtx, accel_mtxel_eps%result_array, &
                               accel_mtxel_eps%gvec_comp, accel_mtxel_eps%result_sort, &
                               accel_mtxel_eps%inner_fftbox, Nfft, scale, mtxel_algo, queue=1)
    !$acc update self(accel_mtxel_eps%result_array(1:pol%nmtx)) async(1)

    ! Get energy denominator (static), or transition energy (FF)
    call get_eden(gvec, vwfn, cwfn, kp, kpq, iv, ic, offset_g, iv_loc, &
                  ic_loc, ispin, irk, freq_idx, pol, eden)

    call update_gme(accel_mtxel_eps%result_array, kp, eden, kfact, iv, ic, jsp,&
                    ic_loc, iv_loc, ispin, irk, freq_idx, pol)

    call timing%stop(timing%fft_mltply)
  end do ! ic_loc
end do ! jsp
```

Update pre-existing buffers on GPU
Perform outer FFT
Loop over inner FFTs
Run each inner FFT multiple times
Update CPU with inner FFT results, postprocess

MTXEL OpenACC, Batched

```
!$acc enter data copyin(vwfn%zv, cwfn%zc, vwfn%isort, cwfn%isort, Nfft, gvec%components, pol%isrtx) create(fftbox1, fftbox2, tmparray)
do jsp=ispin,ispin*kp%nspinor
  call accel_zero_box(fftbox1, Nfft, mtxel_algo)
  call accel_put_into_fftbox(vwfn%ngv, vwfn%zv(:,jsp), gvec%components, vwfn%isort, fftbox1, Nfft, 1.0D+00, mtxel_algo)
  call accel_run_fft(fftbox1, outer_plan, 1, mtxel_algo)

  ! Do a round-robin distribution of the conduction band into batches
  do i_batch = 1, (peinf%ncownactual - 1) / pol%n_ffts_per_batch + 1
    ! Create indexing arrays for batch and create batches of initial
    ! wavefunctions
    n_ffts_in_batch = 0
    call accel_zero_box_batched(fftbox2, Nfft, pol%n_ffts_per_batch, mtxel_algo)
    do i_in_batch = 1, pol%n_ffts_per_batch
      ic_loc(i_in_batch) = (i_batch - 1) * pol%n_ffts_per_batch + i_in_batch
      if (ic_loc(i_in_batch) .gt. peinf%ncownactual) cycle

      n_ffts_in_batch = n_ffts_in_batch + 1
      ic_FE = peinf%invindexc(ic_loc(i_in_batch))
      ic(i_in_batch) = vwfn%nband + ic_FE
      offset_g(i_in_batch) = (ic_loc(i_in_batch)- 1) * cwfn%ngc

      call accel_put_into_fftbox(cwfn%ngc, cwfn%zc(offset_g(i_in_batch)+1:,jsp), gvec%components, &
                                 cwfn%isort, fftbox2(:,:,:,i_in_batch), Nfft, 1.0D+00, mtxel_algo)
    end do

    ! Do the hard work of FFTs across batch
    call accel_run_fft_batched(fftbox2, inner_plan, 1, mtxel_algo)
    call accel_box_multiply_batched(fftbox2, fftbox1, Nfft, 1, n_ffts_in_batch, mtxel_algo)
    call accel_run_fft_batched(fftbox2, inner_plan, 1, mtxel_algo)

    ! Extract finished FFTs and update results
    do i_in_batch = 1, n_ffts_in_batch
      call accel_get_from_fftbox(pol%nmtx, tmparray(:, i_in_batch), gvec%components, pol%isrtx, &
                                 fftbox2(:,:,:,i_in_batch), Nfft, scale, mtxel_algo)
      !$acc update self(tmparray(:,i_in_batch))

      ! Get energy denominator (static), or transition energy (FF)
      call get_eden(gvec, vwfn, cwfn, kp, kpq, iv, ic(i_in_batch), offset_g(i_in_batch), iv_loc, &
                    ic_loc(i_in_batch), ispin, irk, freq_idx, pol, eden)
      call update_gme(tmparray(:, i_in_batch), kp, eden, kfact, iv, ic(i_in_batch), jsp, ic_loc(i_in_batch), &
                      iv_loc, ispin, irk, freq_idx, pol)
      call timing%stop(timing%fft_mltply)
    end do ! i_in_batch
  end do ! i_batch
end do ! jsp
!$acc exit data delete(vwfn%zv, cwfn%zc, vwfn%isort, cwfn%isort, Nfft, gvec%components, pol%isrtx, fftbox1, fftbox2, tmparray)
```

Allocate and copy all data to GPU
Perform outer FFT
Loop over batches of FFTs
Create batch of FFTs to run
Run batch of FFTs at once
Update CPU with batch results, postprocess
Delete GPU memory