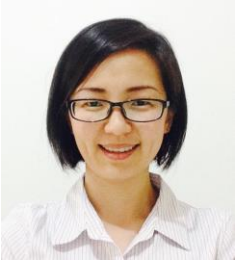
Portability Strategies for Next Generation Exascale Architectures: The BerkeleyGW Case Study

Mauro Del Ben
Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory

Towards exascale solutions in Green function methods and advanced DFT
3 - 8 October 2023, Summer School Phapos, Paphos, Cyprus

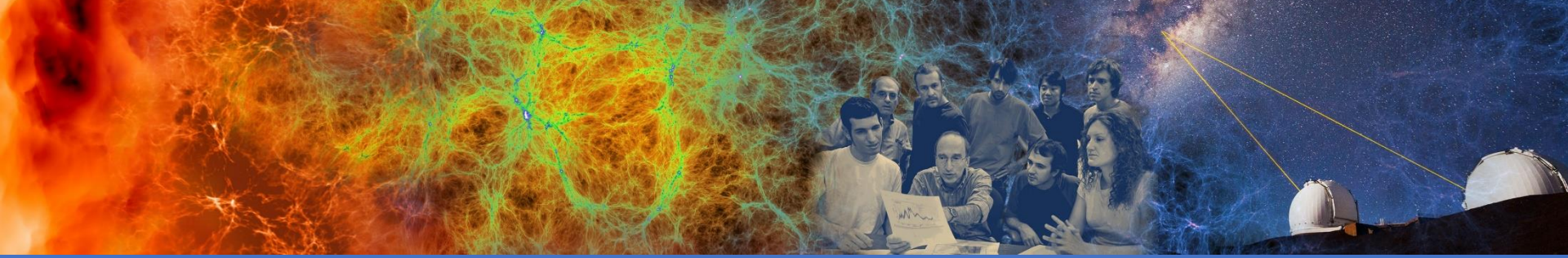
Outline



Mauro Del Ben **Charlene Yang** **Zhenglu Li** **Felipe H. da Jornada** **Steven G. Louie** **Jack Deslippe**

"Accelerating Large-Scale Excited-State GW Calculations on Leadership Class HPC Systems" *in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20 No.4 pp.1 (2020), **ACM Gordon-Bell Finalist**

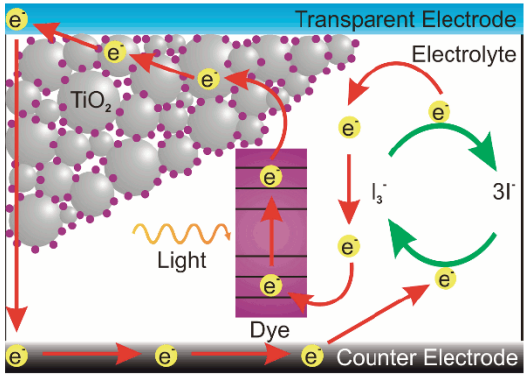
1. Motivation: Importance and Challenges
2. Performance portability strategy and implementation
3. Performance Results
4. Portability Results on Exascale Systems (Frontier & Aurora)
5. Summary and Outlook



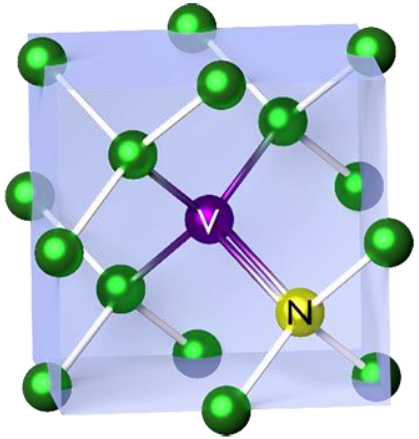
Motivation and Background

Material Science/Chemistry on the Path to Exascale

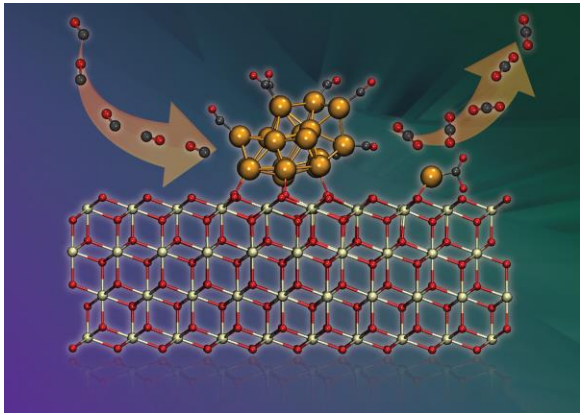
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



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

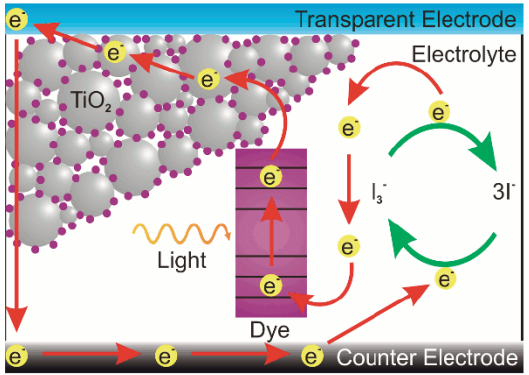


Chemical reaction at interfaces: **Catalysis**

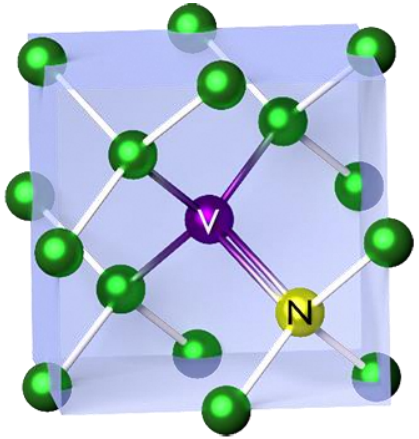
Mat. Sci & Chem apps, such as VASP, Quantum ESPRESSO, QMCPACK, NWchem, BerkeleyGW, CP2K, etc… **heavily use HPC facilities**

Material Science/Chemistry on the Path to Exascale

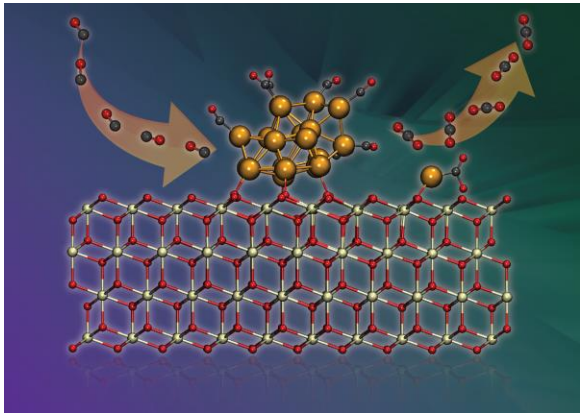
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



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



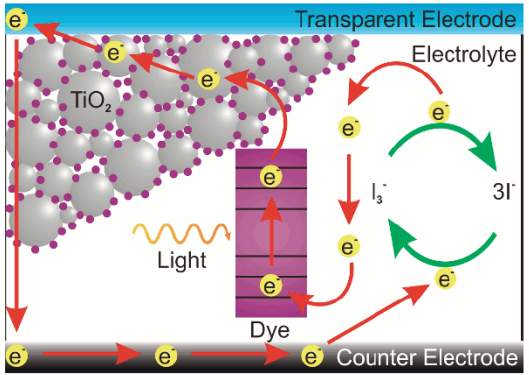
Chemical reaction at interfaces: **Catalysis**

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

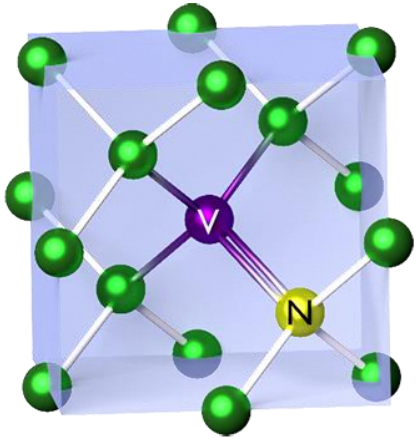
- Applications: Quantum Computers, Batteries, Photovoltaics, Catalysis, etc...

Material Science/Chemistry on the Path to Exascale

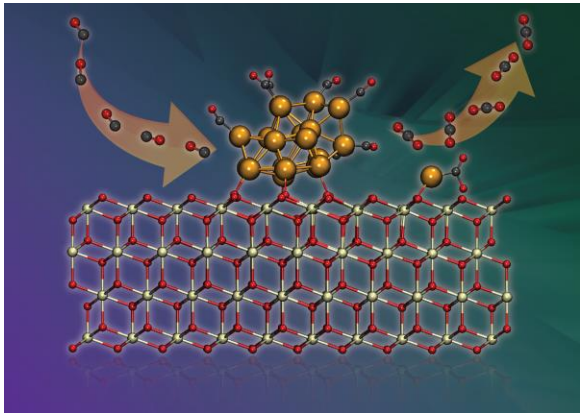
Grätzel cells: Oxide/Organic Interfaces



Cheap, reliable and sustainable **photovoltaics**



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



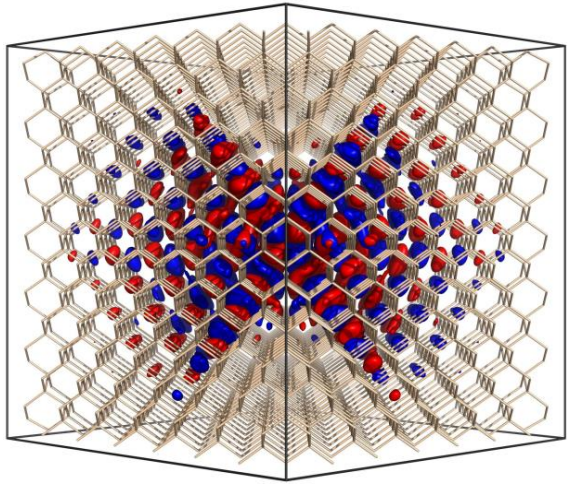
Chemical reaction at interfaces: **Catalysis**

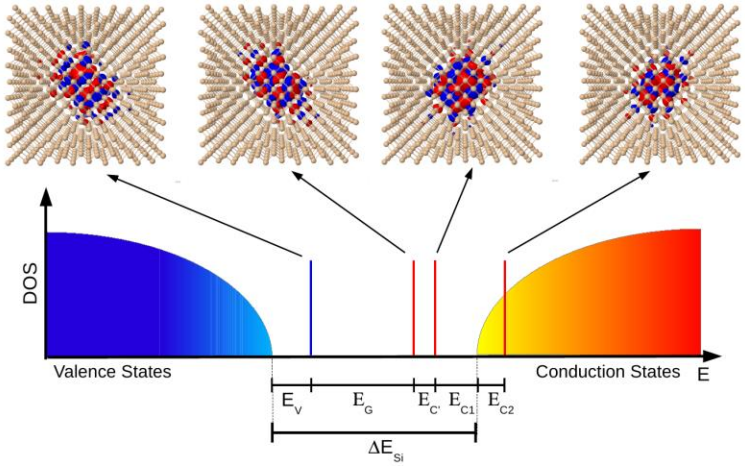
Density Functional Theory (DFT) the workhorse for over three decades

- 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

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

The GW Method: State of the Art

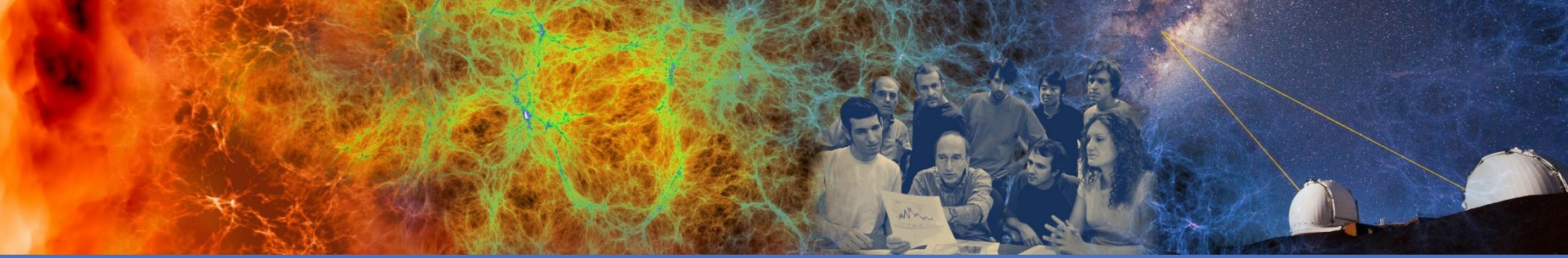
The GW method represents one of the most effective and accurate approach to predict excited-state properties in a wide range of materials

Application of GW to thousands atoms systems still a challenge

Reduce time to solution and extend applicability:

- Develop methods to reduce prefactor and scaling with system size
- Improve performance of existing implementation

In this talk the various strategies employed to accelerate BerkeleyGW on GPU to achieve best performance will be discussed.



The BerkeleyGW Software Package Overview

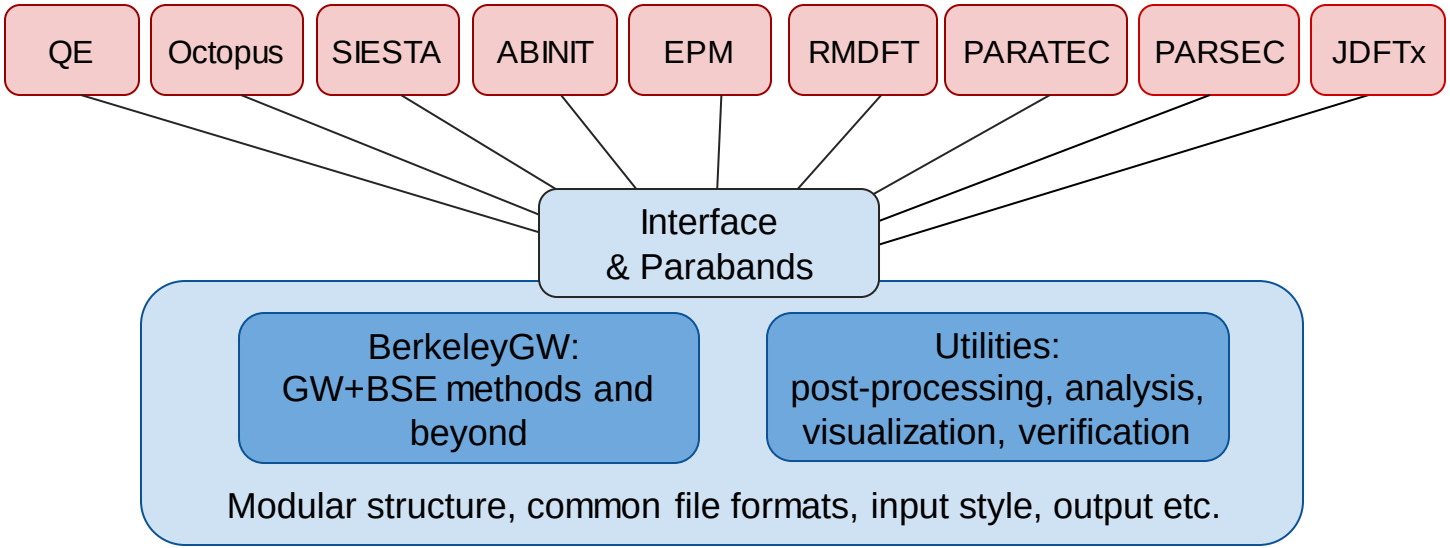
The BerkeleyGW Software Package

BerkeleyGW

Compute the electronic excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond

BerkeleyGW Software Design Vision

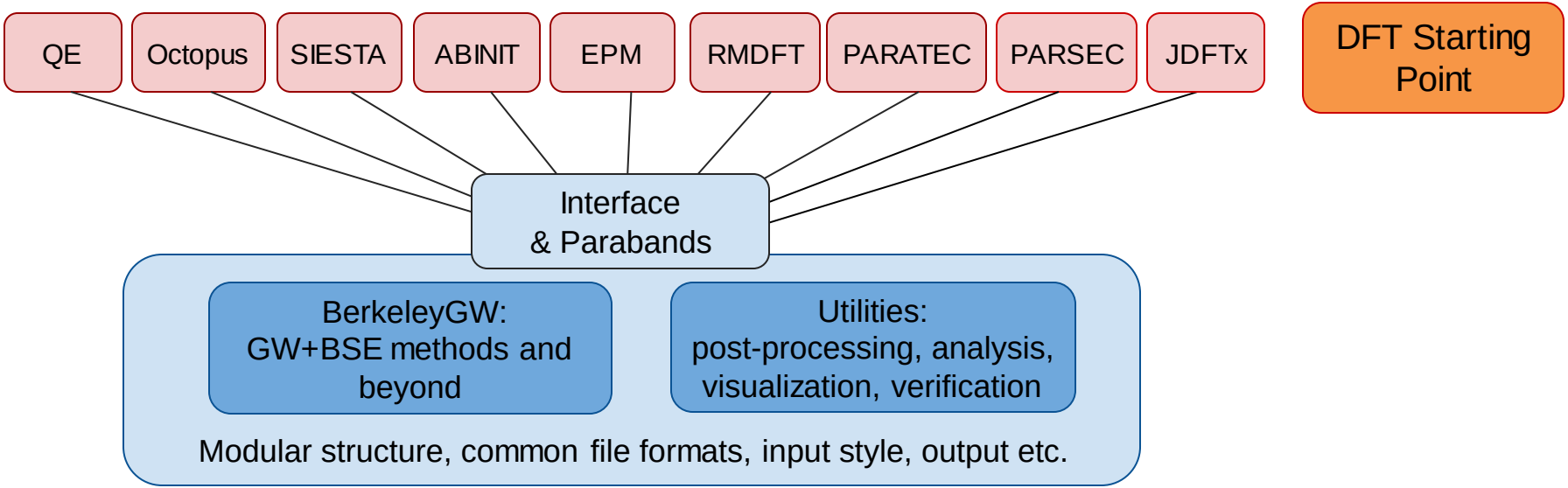
BerkeleyGW compute the electronic excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond



https://berkeleygw.org/

BerkeleyGW Software Design Vision

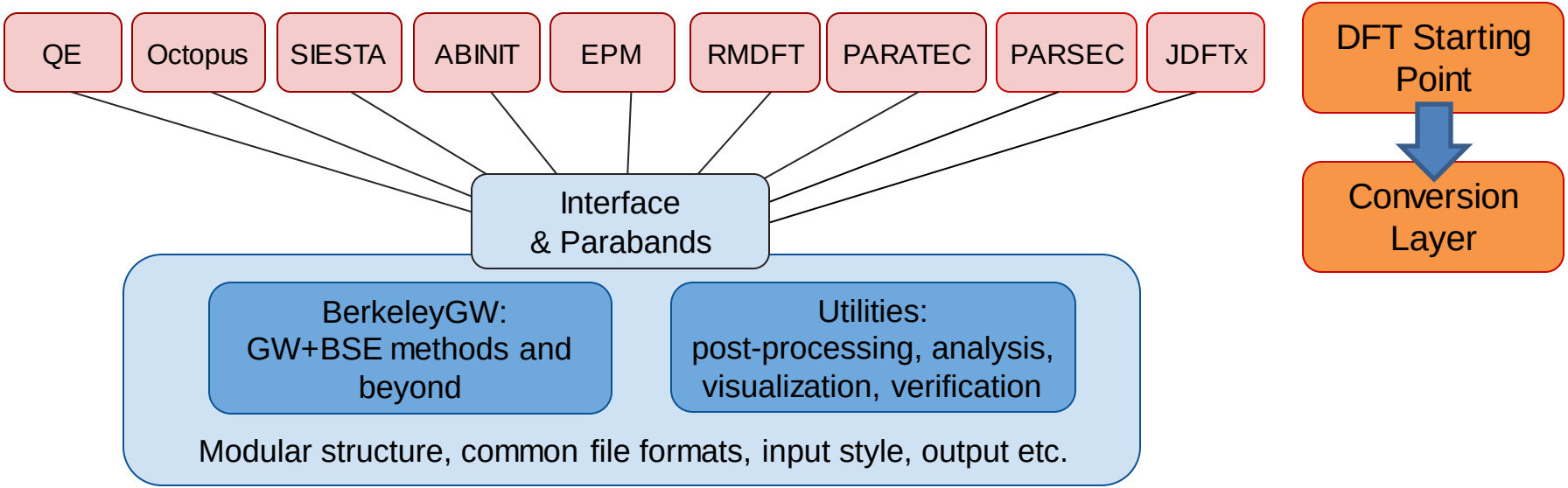
BerkeleyGW compute the electronic excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond



https://berkeleygw.org/

BerkeleyGW Software Design Vision

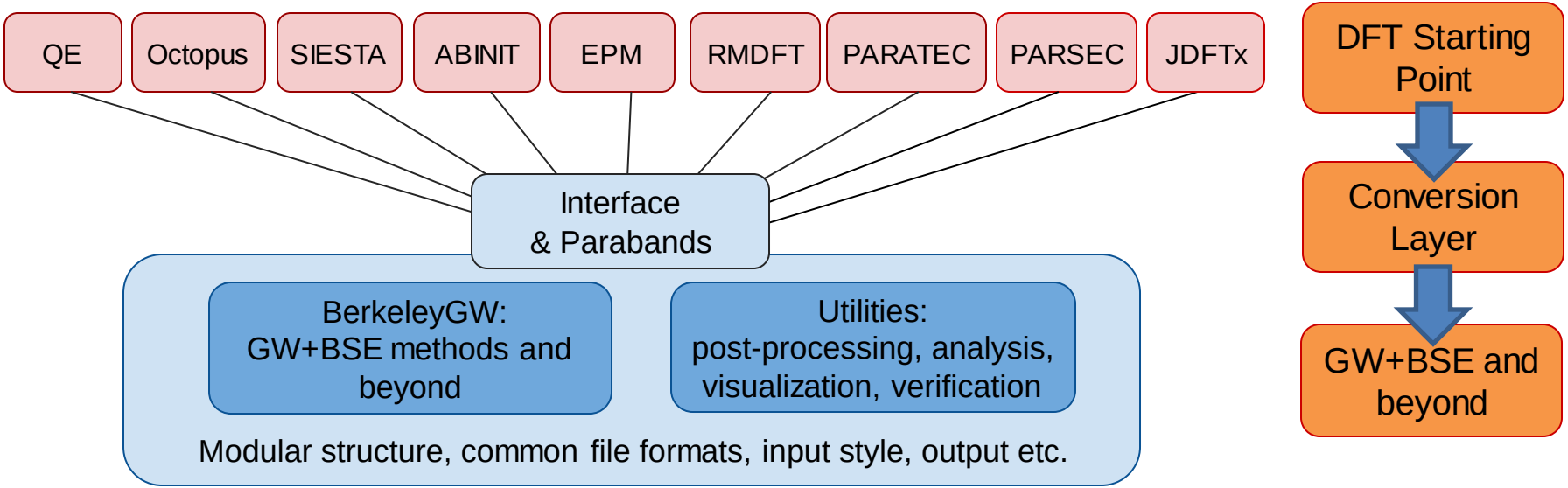
BerkeleyGW compute the electronic excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond



https://berkeleygw.org/

BerkeleyGW Software Design Vision

BerkeleyGW compute the electronic excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond



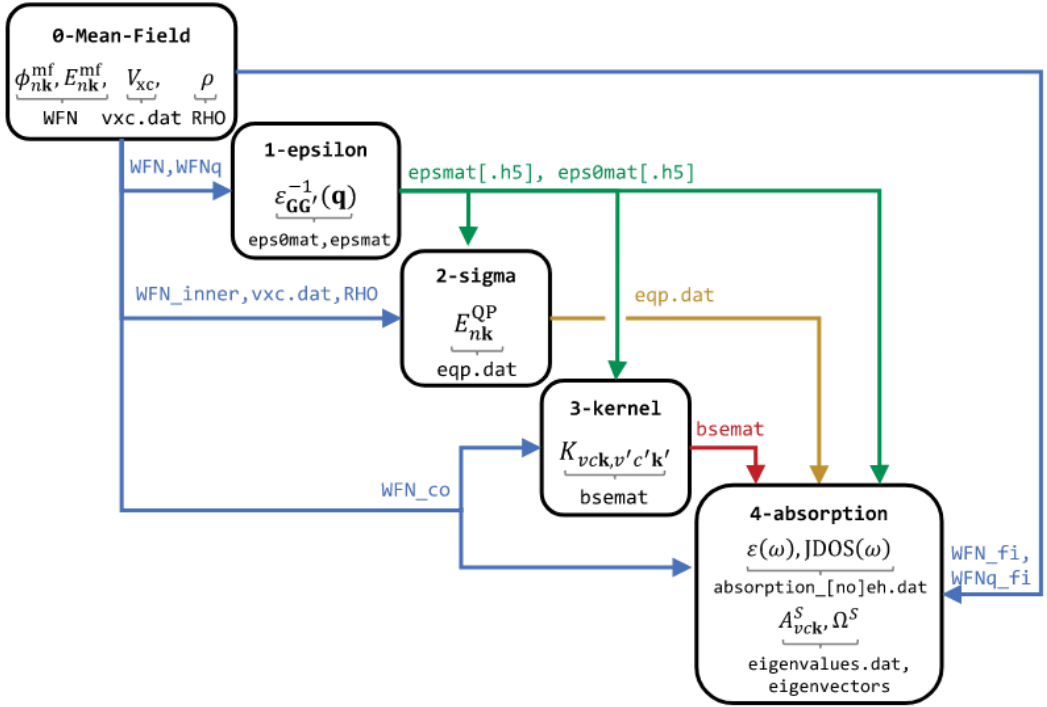
BerkeleyGW: developments are focuses on GW+BSE methodologies

https://berkeleygw.org/

BerkeleyGW Highlights

- Supports a large set of Mean-Field codes
- Supports 3D, 2D, 1D and Molecular Systems. Coulomb Truncation
- Support for Semiconductor, Metallic and Semi-Metallic Systems
- Efficient Algorithms and Use of Libraries for (Pre-) Exascale HPC systems.
- Massively Parallel, up to 100,000 CPUs, and recently up to 10,000 of GPUs.
- Full, 2-component spinor support enabling spin-orbit coupling (SOC)
- Exciton finite momentum Q for exciton band structures

BerkeleyGW Workflow



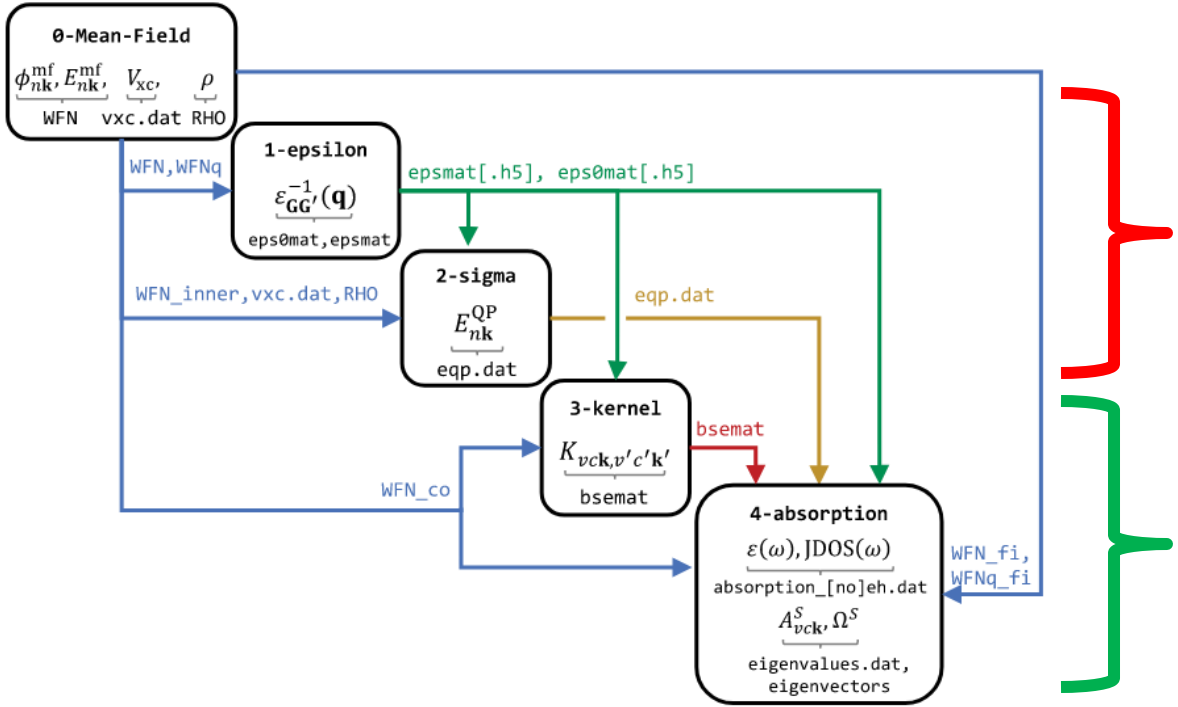
Four major modules: epsilon, sigma, kernel and absorption; *why?*

(1) Each have different data layout, computational cost and memory requirements

(2) Intermediates from each modules reused by others in multiple runs

http://manual.berkeleygw.org/3.0/overview-workflow/

BerkeleyGW Workflow

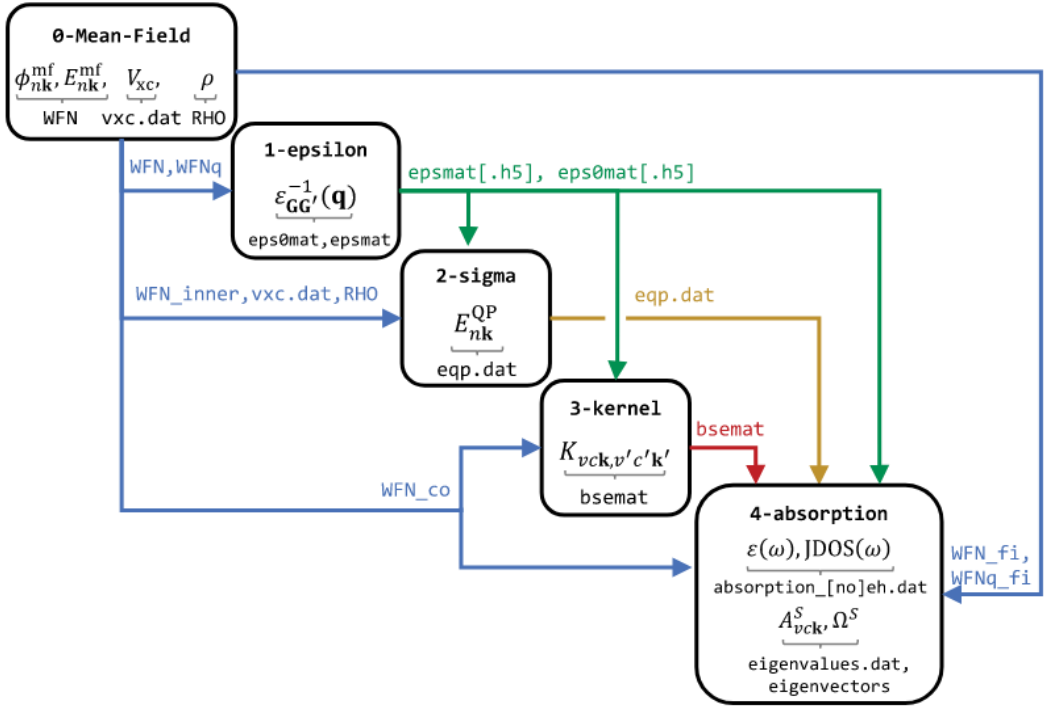


GW Workflow
epsilon + sigma

BSE Workflow
kernel + absorption

http://manual.berkeleygw.org/3.0/overview-workflow/

BerkeleyGW Workflow



Synopsis

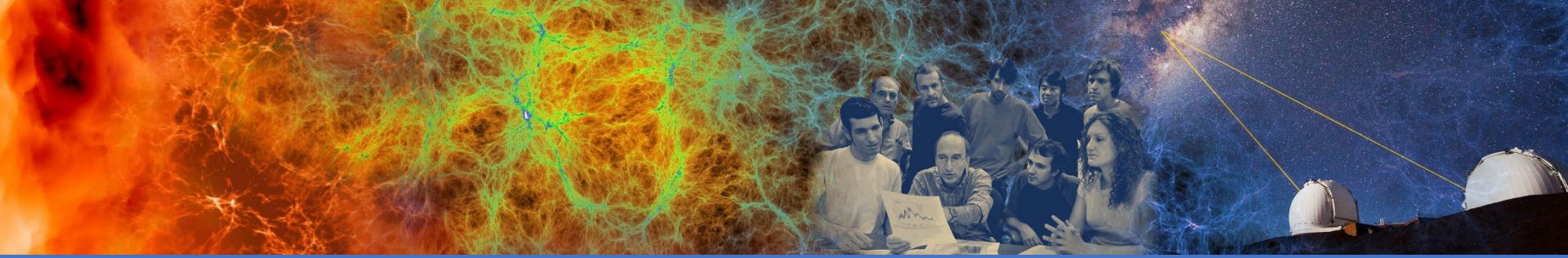
Epsilon: Generate the dielectric function and its frequency dependence

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

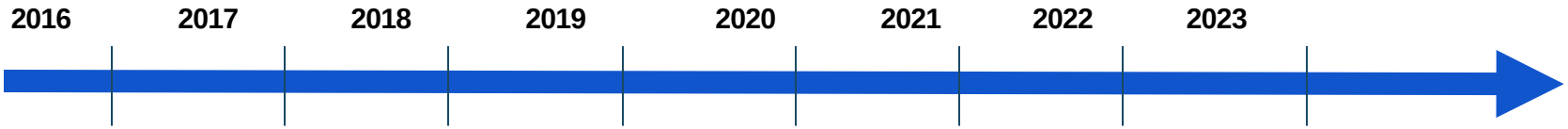
http://manual.berkeleygw.org/3.0/overview-workflow/



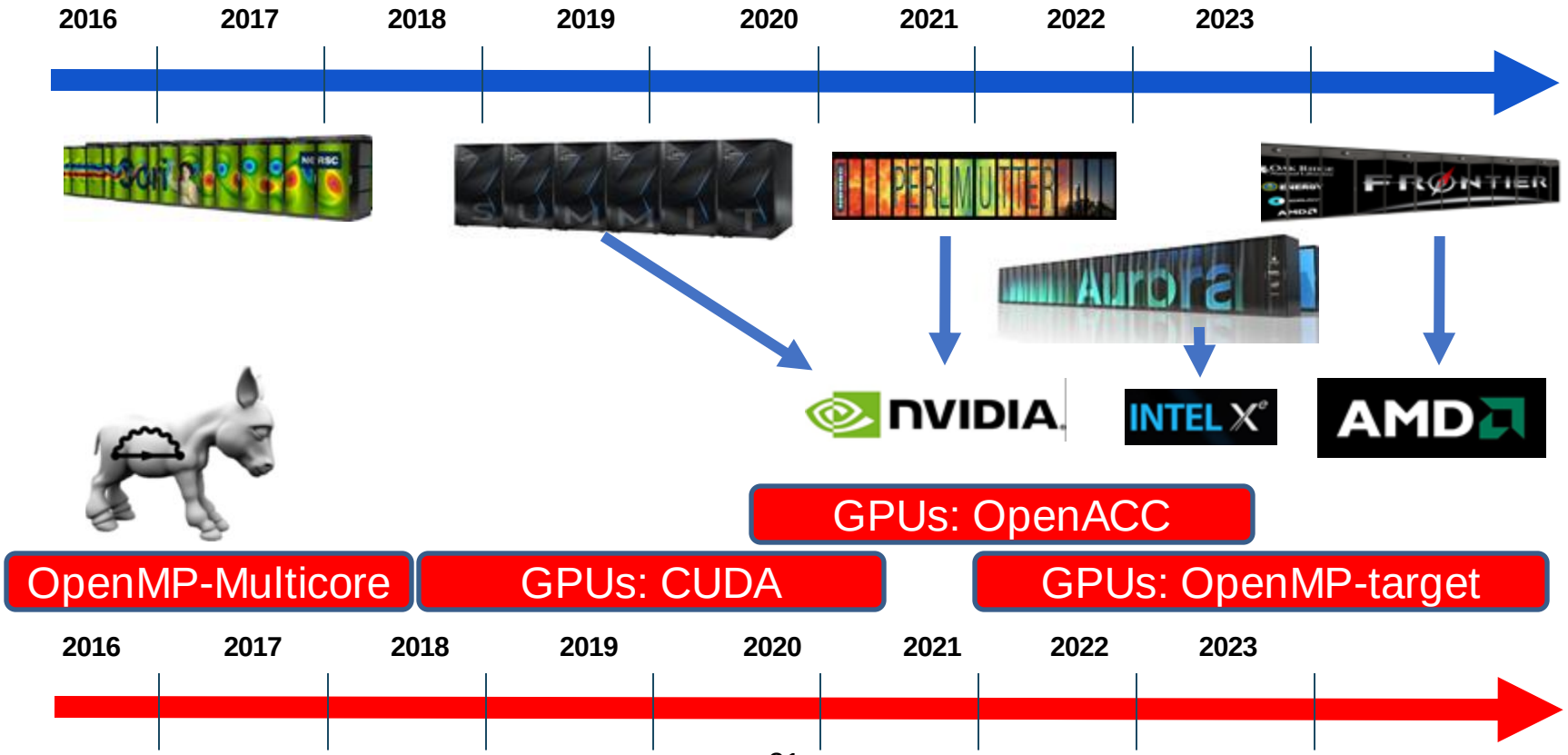
BerkeleyGW on the Path to Exascale

The Path to Exascale

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

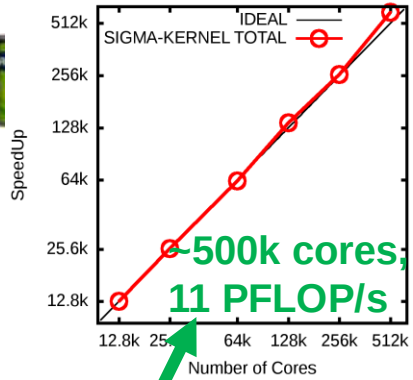


BerkeleyGW on the Path to Exascale



BerkeleyGW on the Path to Exascale

2016 2017 2018 2019 2020 2021 2022 2023



GPUs: OpenACC

OpenMP-Multicore

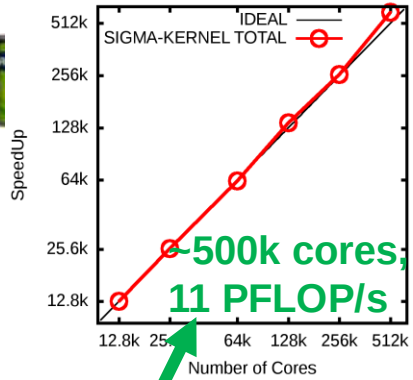
GPUs: CUDA

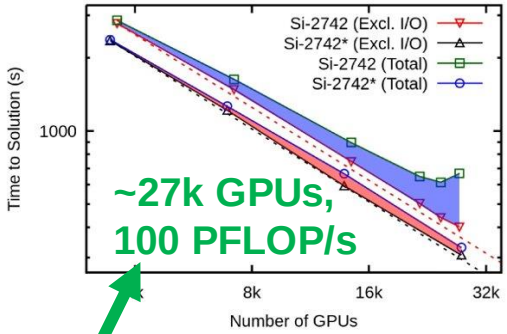
GPUs: OpenMP-target

2016 2017 2018 2019 2020 2021 2022 2023

BerkeleyGW on the Path to Exascale

2016 2017 2018 2019 2020 2021 2022 2023





GPUs: OpenACC

OpenMP-Multicore

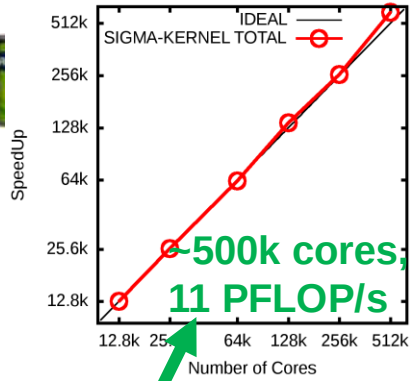
GPUs: CUDA

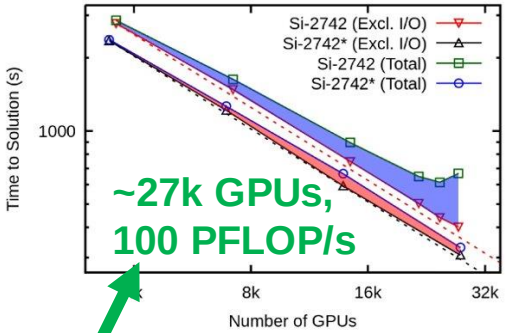
GPUs: OpenMP-target

2016 2017 2018 2019 2020 2021 2022 2023

BerkeleyGW on the Path to Exascale

2016 2017 2018 2019 2020 2021 2022 2023





OpenMP-Multicore

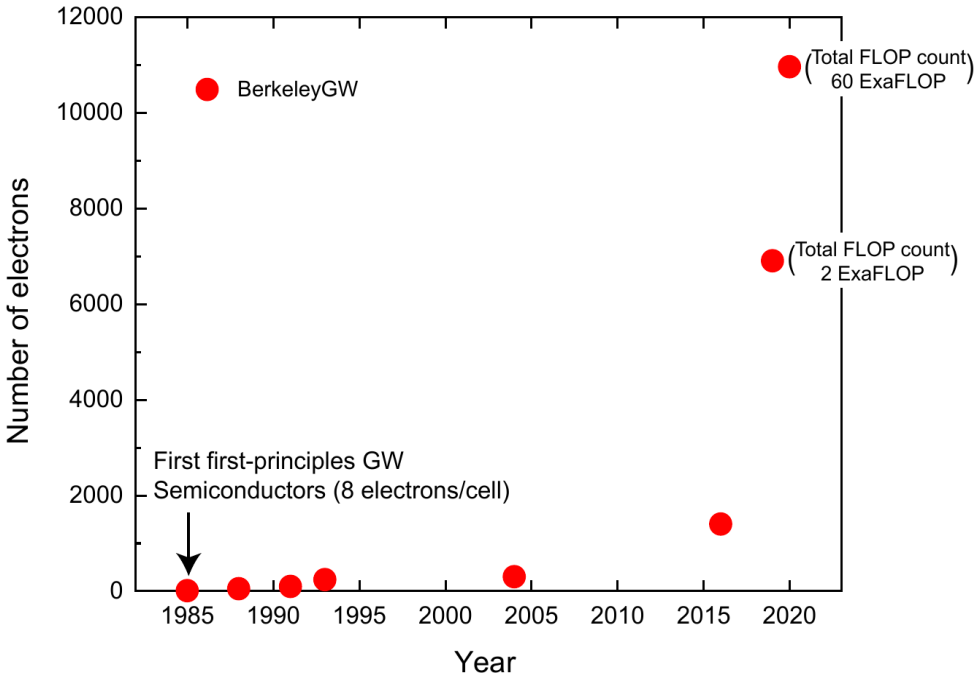
GPUs: CUDA

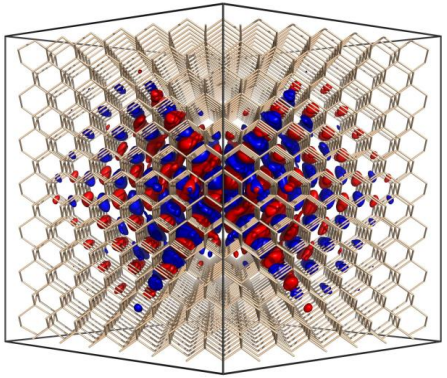
GPUs:

GPUs: OpenMP-target

2016 2017 2018 2019 2020 2021

BerkeleyGW Overview Summary

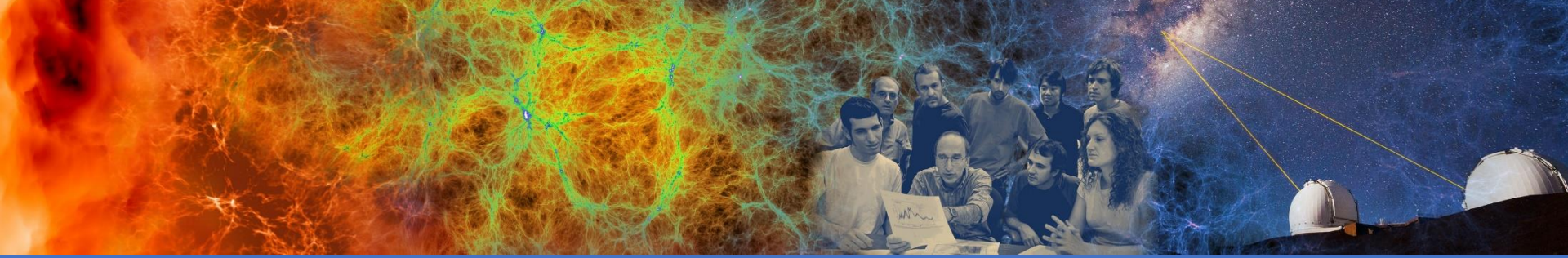




Divacancy defect in silicon
~11,000 electrons and over 2,700 atoms:
time to solution of the ~10s of mins.

2020 ACM Gordon-Bell finalist:
https://dl.acm.org/doi/abs/10.5555/3433701.3433706

By software optimization of BerkeleyGW on leadership class HPC systems the application of GW to systems of increasing size closely follow the Moore's law



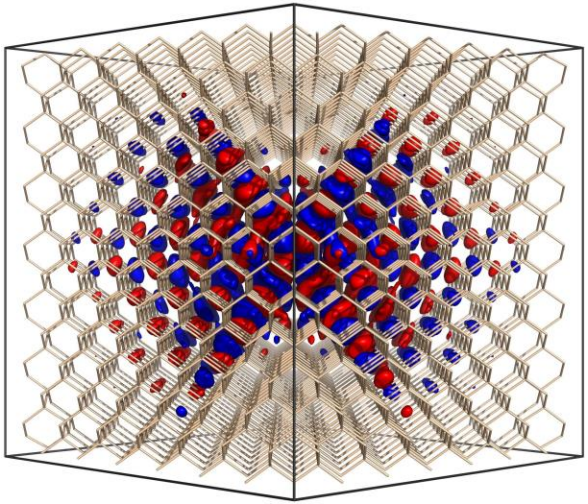
Benchmarks for Performance Results

HPC Systems

- Summit (OLCF)
  4,608 nodes, each with 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter (NERSC)
  1536 nodes, each with 1 AMD Milan CPU and 4 NVIDIA A100 GPUs
- Cori-GPU (NERSC)
  18 nodes, each with 2 Intel Xeon Skylake CPUs and 8 NVIDIA V100 GPUs
- Cori-Haswell (NERSC)
  2,688 Haswell nodes, each with 2 Intel Xeon E5-2698v3 CPUs

Benchmark for Performance Measurement

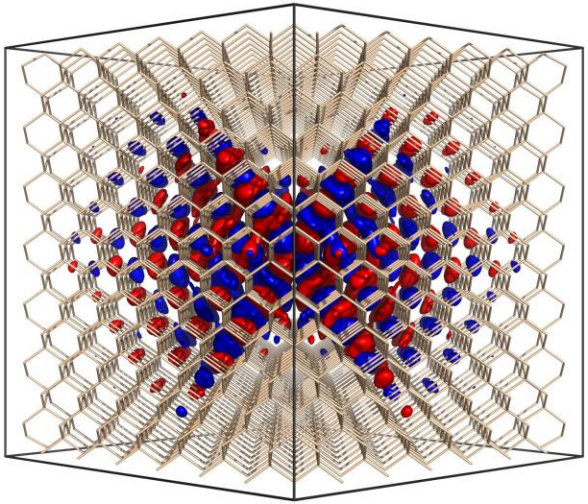
- Real applications: Defects in Semiconductors (silicon, silicon carbide) for QBit
- System size: up to **thousands of atoms**, **11000+ electrons**



Benchmark for Performance Measurement

- Real applications: Defects in Semiconductors (silicon, silicon carbide) for QBit
- System size: up to **thousands of atoms**, **11000+ electrons**

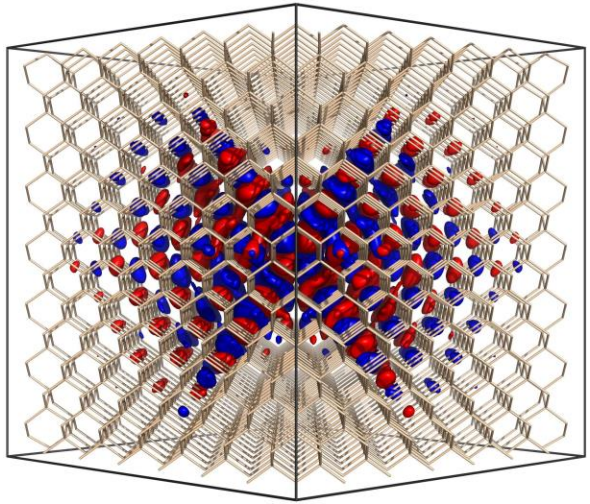
Parameters	Si-214	Si-510	Si-998	SiC-998	Si-2742
N_{spin}	1	1	1	2 (↑/↓)	1
N_G^{ψ}	31,463	74,653	145,837	422,789	363,477
N_G	11,075	26,529	51,627	149,397	141,505
N_b	6,397	15,045	29,346	16,153	80,694
N_v	428	1,020	1,996	1,997/1,995	5,484
N_c	5,969	14,025	27,350	14,156/14,158	75,210
N_Σ	Variable, up to 120 per spin				
Epsilon PFLOPs	2.5	80.5	1164	10,091	66,070
Epsilon Memory (TB)	0.45	6.07	45.1	135	934
Sigma PFLOPs	0.127	1.71	12.6	58.2	260.7
Sigma Memory (GB)	6.19	34.3	133.8	791.4	1006



Benchmark for Performance Measurement

- Real applications: Defects in Semiconductors (silicon, silicon carbide) for QBit
- System size: up to **thousands of atoms**, **11000+ electrons**

Parameters	Si-214	Si-510	Si-998	SiC-998	Si-2742
N_{spin}	1	1	1	2 (↑/↓)	1
N_G^{ψ}	31,463	74,653	145,837	422,789	363,477
N_G	11,075	26,529	51,627	149,397	141,505
N_b	6,397	15,045	29,346	16,153	80,694
N_v	428	1,020	1,996	1,997/1,995	5,484
N_c	5,969	14,025	27,350	14,156/14,158	75,210
N_Σ	Variable, up to 120 per spin				
Epsilon PFLOPs	2.5	80.5	1164	10,091	66,070
Epsilon Memory (TB)	0.45	6.07	45.1	135	934
Sigma PFLOPs	0.127	1.71	12.6	58.2	260.7
Sigma Memory (GB)	6.19	34.3	133.8	791.4	1006



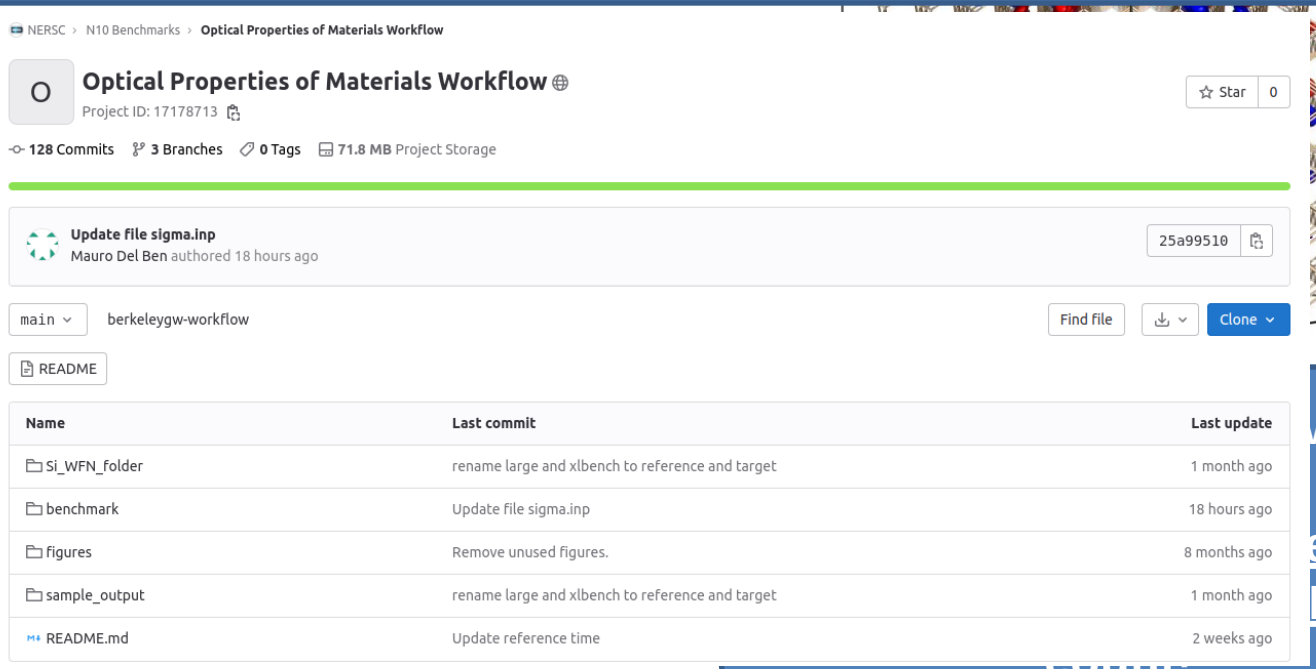
Very Large-scale GW applications:
Requiring **100s of TB memory and 10s of total ExaFLOP count!**

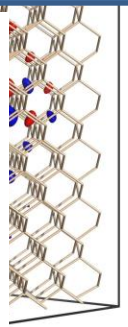
Benchmark for Performance Measurement

https://gitlab.com/NERSC/N10-benchmarks/berkeleygw-workflow

- System s
  11000+ e

Parameters
N_{spin}
N_G^{ψ}
N_G
N_b
N_v
N_c
N_Σ
Epsilon PFLOPs
Epsilon Memory (T
Sigma PFLOPs
Sigma Memory (Gl



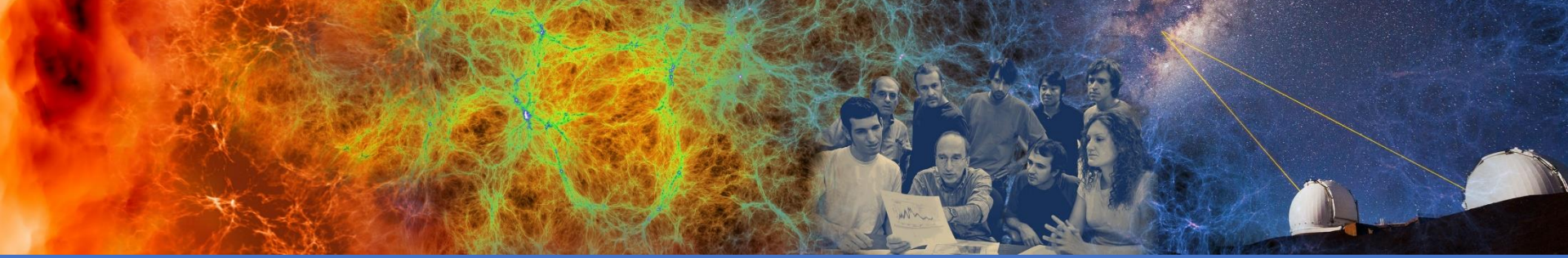


W

emory

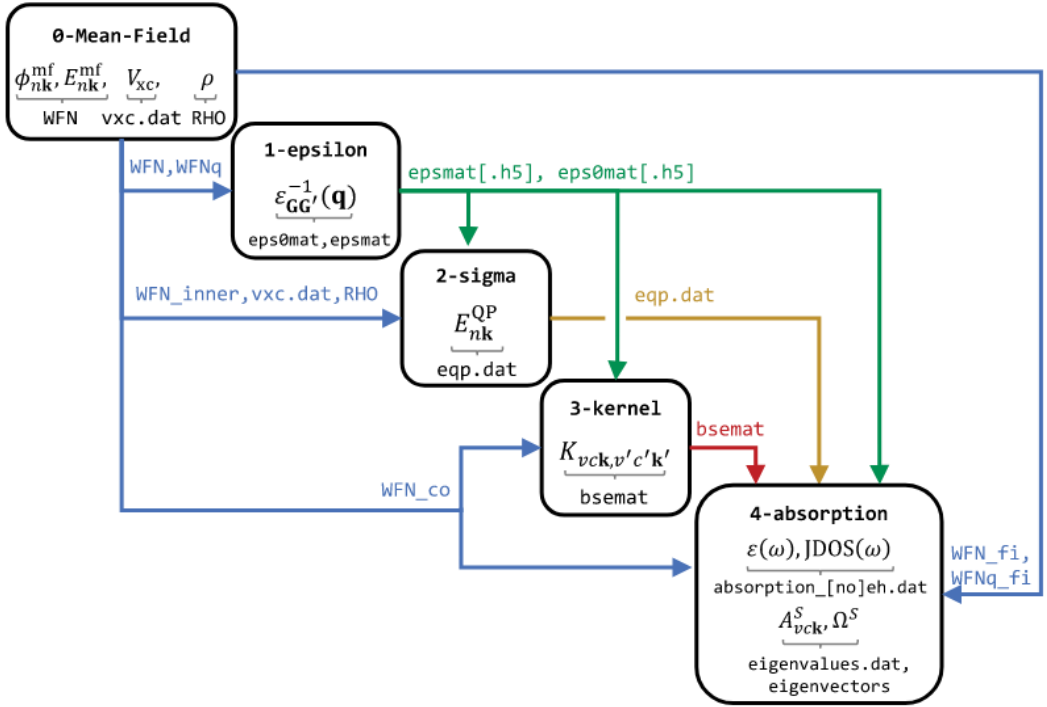
LOP

count!



GW in BerkeleyGW

BerkeleyGW Workflow

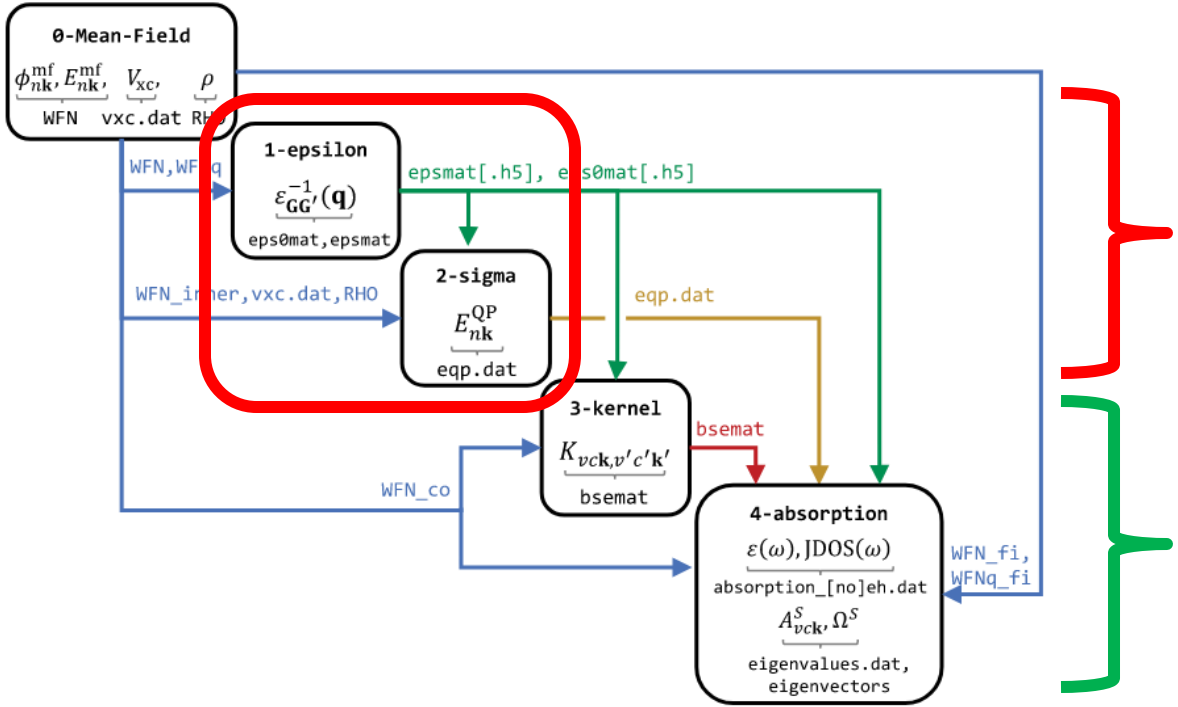


GW Workflow
epsilon + sigma

BSE Workflow
kernel + absorption

http://manual.berkeleygw.org/3.0/overview-workflow/

The GW Workflow: Epsilon + Sigma



GW Workflow
epsilon + sigma

BSE Workflow
kernel + absorption

http://manual.berkeleygw.org/3.0/overview-workflow/

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \boxed{\Sigma(\mathbf{r}, \mathbf{r}'; E_n)}\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

GW Self-Energy Operator Σ : non-Hermitian, non-local, frequency dependent

(Note: In DFT, the role of self-energy is replaced by static and local $V_{\mathrm{xc}}(\mathbf{r})$)

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

GW Self-Energy Operator Σ : non-Hermitian, non-local, frequency dependent

(Note: In DFT, the role of self-energy is replaced by static and local $V_{\mathrm{xc}}(\mathbf{r})$)

High GW Computational Cost in Two Major Bottlenecks:

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

GW Self-Energy Operator Σ : non-Hermitian, non-local, frequency dependent

(Note: In DFT, the role of self-energy is replaced by static and local $V_{\mathrm{xc}}(\mathbf{r})$)

High GW Computational Cost in Two Major Bottlenecks:

- **Epsilon**: Inverse Dielectric Matrix **O(N^4)**

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

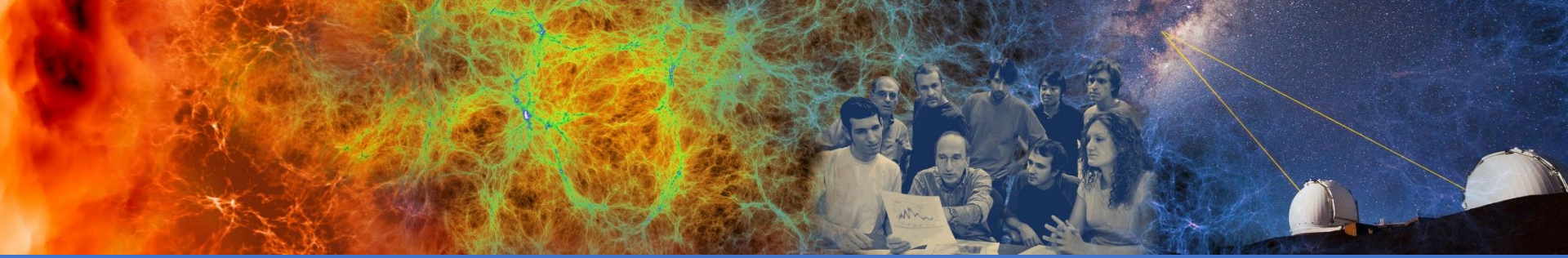
GW Self-Energy Operator Σ : non-Hermitian, non-local, frequency dependent

(Note: In DFT, the role of self-energy is replaced by static and local $V_{\mathrm{xc}}(\mathbf{r})$)

High GW Computational Cost in Two Major Bottlenecks:

- **Epsilon**: Inverse Dielectric Matrix **O(N^4)**
- **Sigma**: Self-Energy Matrix Elements **O(N^4)**

ϵ^{-1} matrix



Epsilon:
System, Node and GPU Implementation

Epsilon: Inverse Dielectric Function (Matrix)

Three major computational steps: input $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

Epsilon: Inverse Dielectric Function (Matrix)

Three major computational steps: input $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

1. Calculate plane-waves matrix elements (FFT's) O(N^3) -> MTXEL

$$M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} | \, e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \, | \psi_{a\mathbf{k}} \rangle$$

Epsilon: Inverse Dielectric Function (Matrix)

Three major computational steps: input $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

1. Calculate plane-waves matrix elements (FFT's) O(N^3) -> MTXEL

$$M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} | \, e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \, | \psi_{a\mathbf{k}} \rangle$$

2. Calculate RPA polarizability (Matrix-Multiplication/ZGEMM) O(N^4) -> CHI-0

$$\chi(\mathbf{q}, \omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega) \mathbf{M}(\mathbf{q})$$

Epsilon: Inverse Dielectric Function (Matrix)

Three major computational steps: input $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

1. Calculate plane-waves matrix elements (FFT's) O(N^3) -> MTXEL

$$M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} | \, e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \, | \psi_{a\mathbf{k}} \rangle$$

2. Calculate RPA polarizability (Matrix-Multiplication/ZGEMM) O(N^4) -> CHI-0

$$\chi(\mathbf{q}, \omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega) \mathbf{M}(\mathbf{q})$$

3. Compute dielectric matrix and its inverse (ScalaPACK) O(N^3)

$$\epsilon^{-1}(\mathbf{q}, \omega_i) = (I - v\chi(\mathbf{q}, \omega_i))^{-1}$$

Epsilon: Inverse Dielectric Function (Matrix)

Three major computational steps: input $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

1. Calculate plane-waves matrix elements (FFT's) O(N^3) -> MTXEL

$$M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} | \, e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \, | \psi_{a\mathbf{k}} \rangle$$

2. Calculate RPA polarizability (Matrix-Multiplication/ZGEMM) O(N^4) -> CHI-0

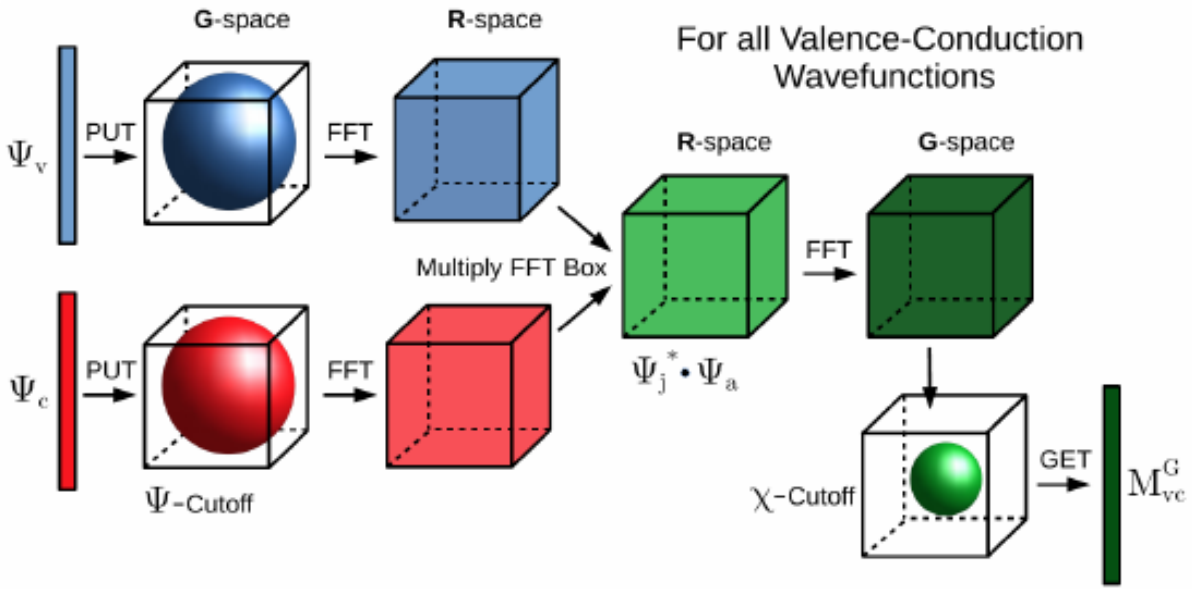
$$\chi(\mathbf{q}, \omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega) \mathbf{M}(\mathbf{q})$$

3. Compute dielectric matrix and its inverse (ScalaPACK) O(N^3)

$$\epsilon^{-1}(\mathbf{q}, \omega_i) = (I - v\chi(\mathbf{q}, \omega_i))^{-1}$$

For large scale applications the evaluation of the polarizability (CHI-0) is by far the most computationally intensive part of the calculation:
large distributed matrix-multiplication over fat and short matrices

Epsilon: Matrix Elements, FFTs (MTXEL)

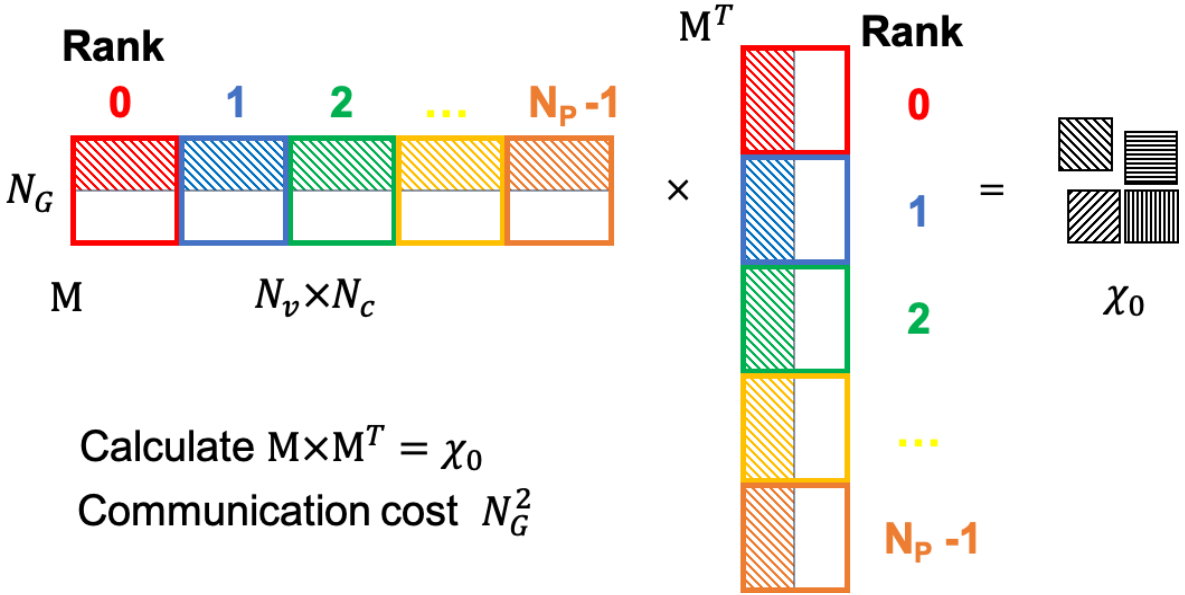


Unfavorable
O(N) data vs
O(N lnN) FLOPs

Two nested loops executing a sequence of simple kernels

- All wavefunction offloaded to GPU before the loops
- Inner most loop broken over blocks
- Async memory transfer over batches

Epsilon: The CHI-0 Kernel

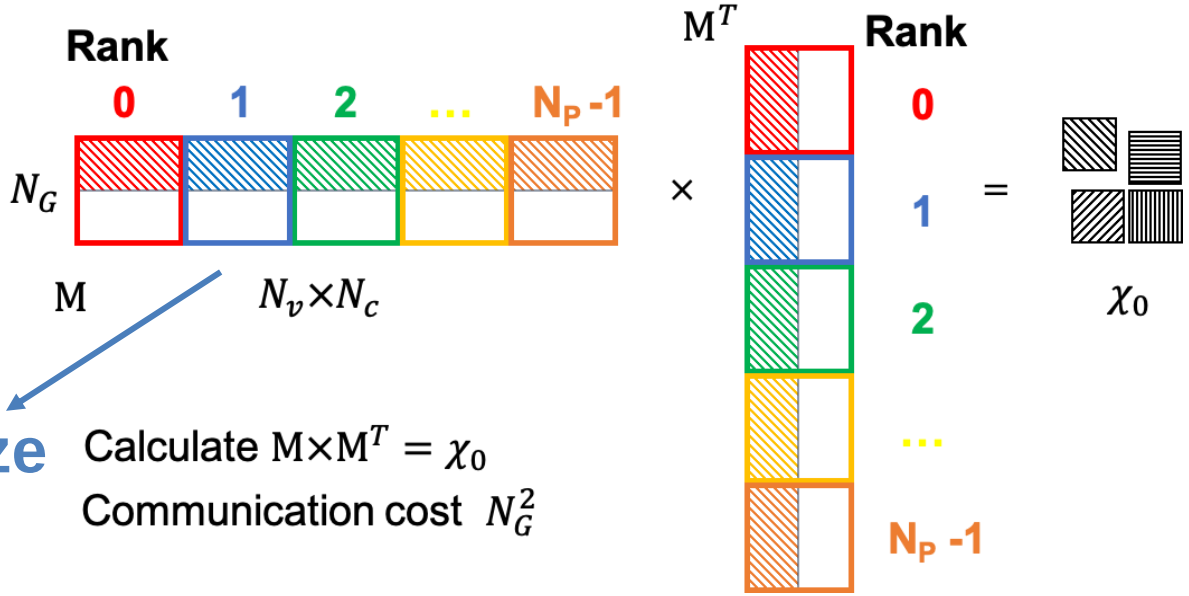


Calculate $M \times M^T = \chi_0$

Communication cost N_G^2

Data layout for **M** matrix in CHI-0 kernel

Epsilon: The CHI-0 Kernel



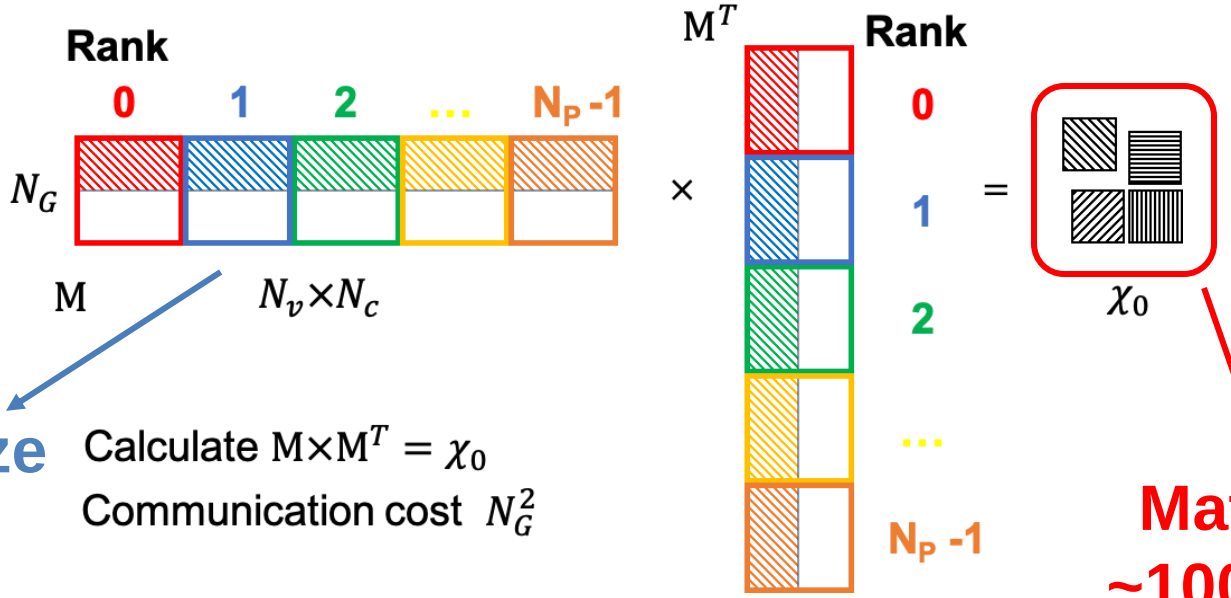
Matrix size ~100k x 100M

Calculate $M \times M^T = \chi_0$

Communication cost N_G^2

Data layout for **M** matrix in CHI-0 kernel

Epsilon: The CHI-0 Kernel



Matrix size ~100k x 100M

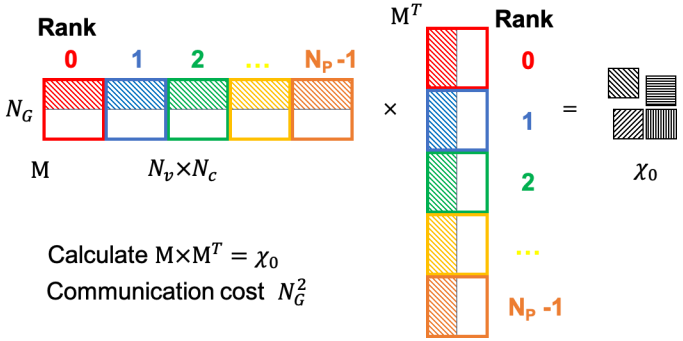
Calculate $M \times M^T = \chi_0$

Communication cost N_G^2

Matrix size ~100k x 100k

Data layout for **M** matrix in CHI-0 kernel

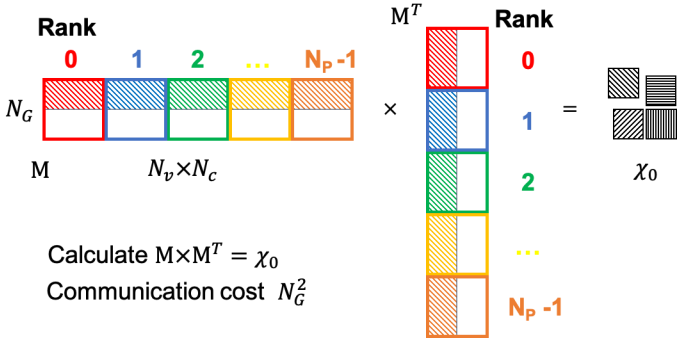
Epsilon: The CHI-0 Kernel



Data layout for **M** matrix in CHI-0 kernel

- Offload Data Preparation
  Accelerate data preparation (potential memory bottleneck)
- Batching Mechanism
  Avoid hitting OOM on GPU
- Non-blocking Cyclic Communication
  Overlap Computation (GPU) and MPI communication (CPU)

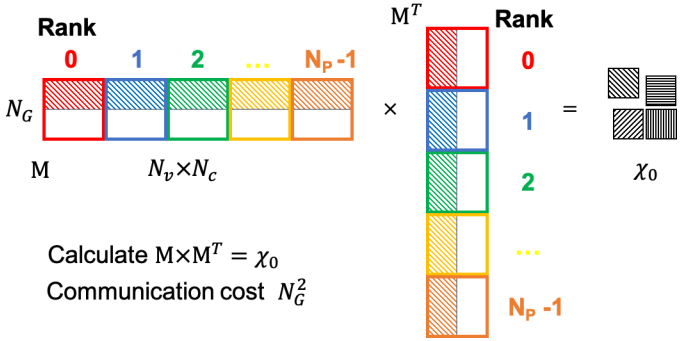
Epsilon: The CHI-0 Kernel



Data layout for **M** matrix in CHI-0 kernel

- **Offload Data Preparation**
  Accelerate data preparation (potential memory bottleneck)
- Batching Mechanism
  Avoid hitting OOM on GPU
- Non-blocking Cyclic Communication
  Overlap Computation (GPU) and MPI communication (CPU)

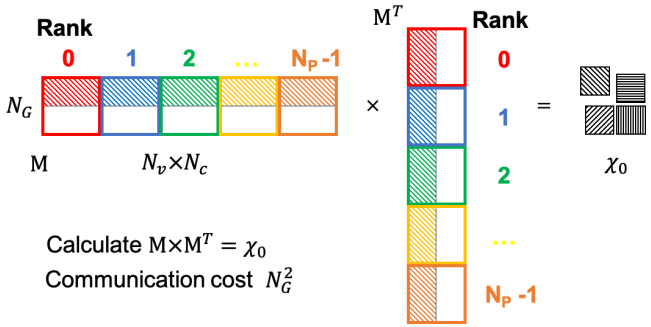
Epsilon: The CHI-0 Kernel



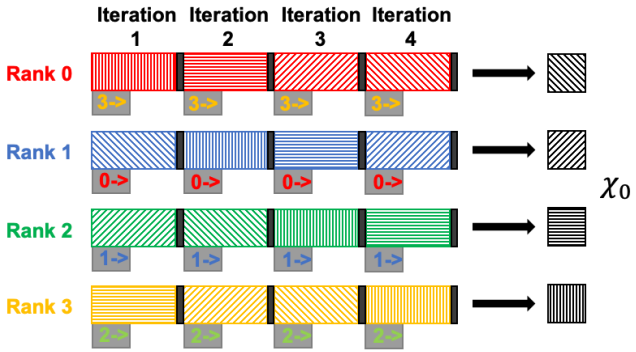
Data layout for **M** matrix in CHI-0 kernel

- Offload Data Preparation
  Accelerate data preparation (potential memory bottleneck)
- **Batching Mechanism**
  Avoid hitting OOM on GPU
- Non-blocking Cyclic Communication
  Overlap Computation (GPU) and MPI communication (CPU)

Epsilon: The CHI-0 Kernel



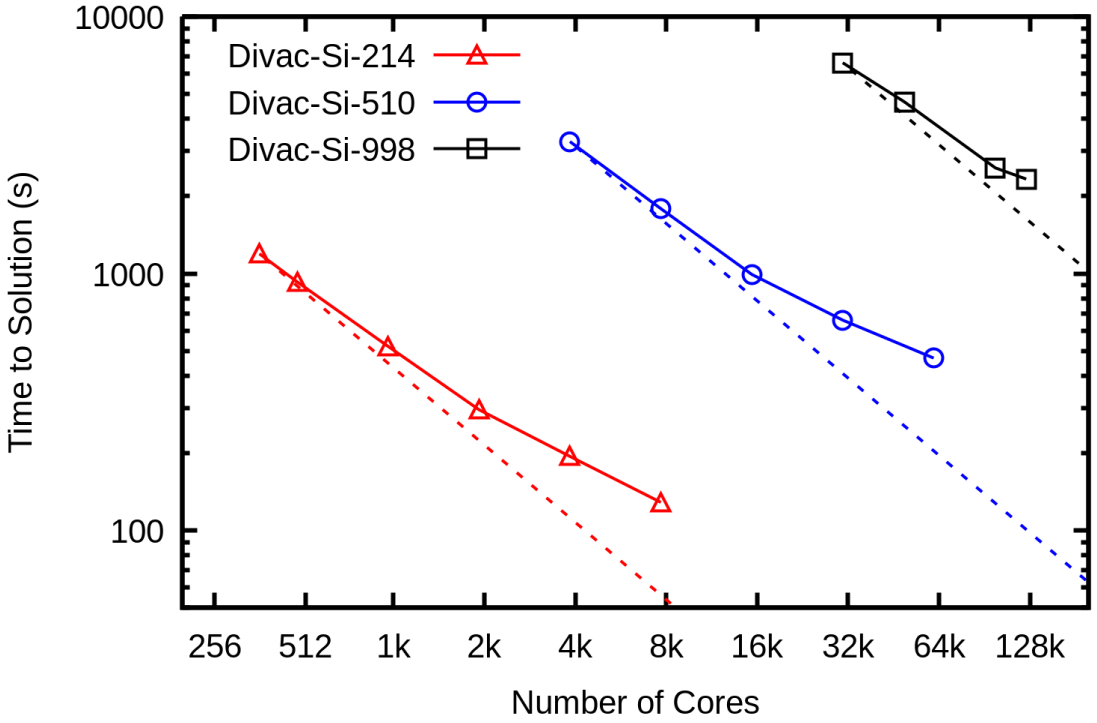
Data layout for **M** matrix in CHI-0 kernel



Comput./Commun. pattern for non-blocking cyclic scheme

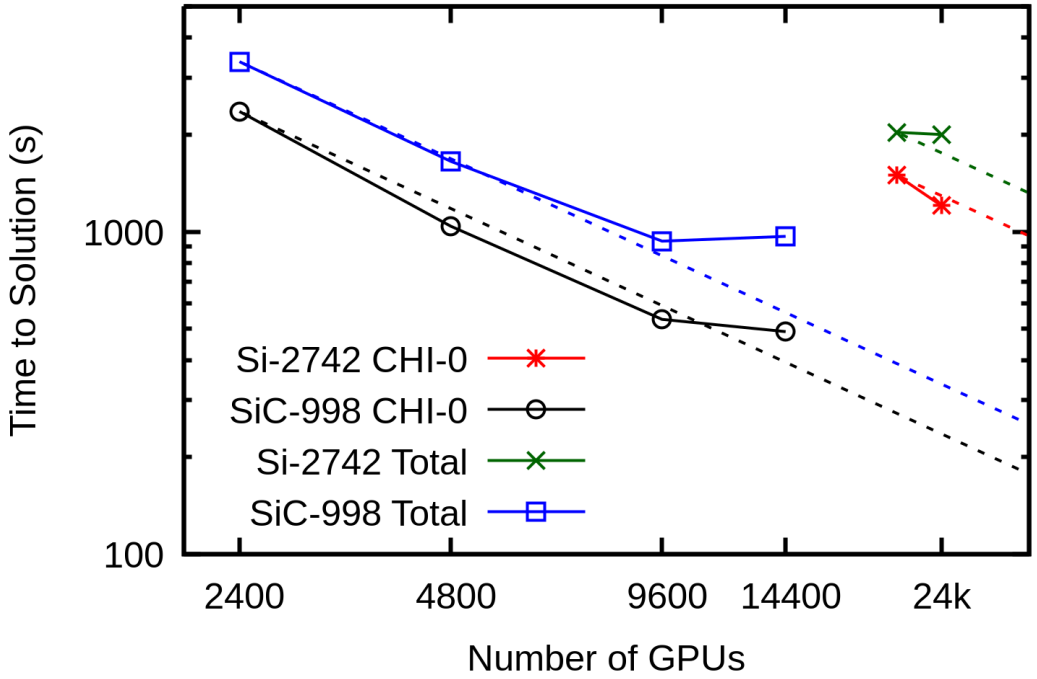
- Offload Data Preparation
  Accelerate data preparation (potential memory bottleneck)
- Batching Mechanism
  Avoid hitting OOM on GPU
- **Non-blocking Cyclic Communication**
  Overlap Computation (GPU) and MPI communication (CPU)

Epsilon: Performance



Strong Scaling of epsilon Full Frequency multi-core implementation (NERSC)

Epsilon: Performance on Summit (GPU)



Epsilon scales linearly well to thousands of GPUs

Strong Scaling of epsilon measured on Summit@OLCF (Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs)

Epsilon: CPU vs GPU

	MTXEL	CHI-0	Invert	Total
CPU Only	616	1120	10.3	1794
GPU Full-Offload	24.4	47.7	9.6	96.3

Runtime comparison in seconds on Cori-GPU with 2 nodes, with GPU and CPU only for Si-214

Epsilon: CPU vs GPU

	MTXEL	CHI-0	Invert	Total
CPU Only	616	1120	10.3	1794
GPU Full-Offload	24.4	47.7	9.6	96.3

25x **23.5x**

Runtime comparison in seconds on Cori-GPU with 2 nodes, with GPU and CPU only for Si-214

Epsilon: CPU vs GPU

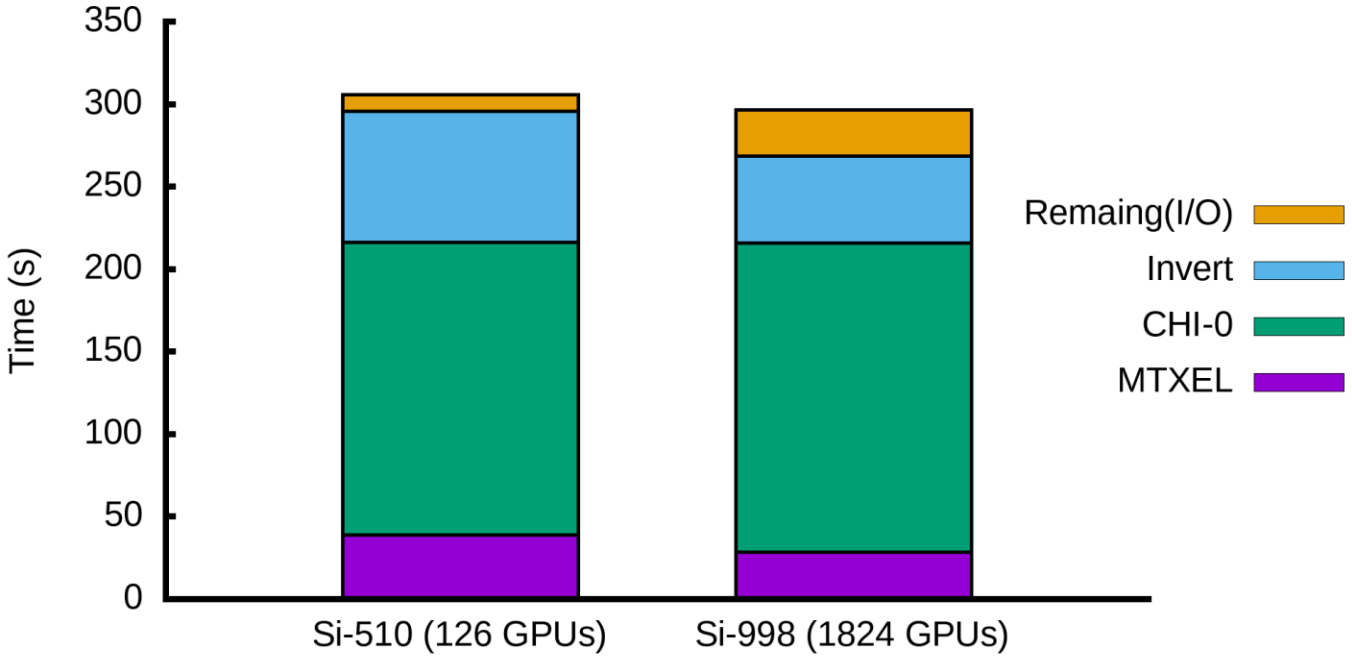
18.6x speedup (overall)

	MTXEL	CHI-0	Invert	Total
CPU Only	616	1120	10.3	1794
GPU Full-Offload	24.4	47.7	9.6	96.3

25x **23.5x** **18.6x**

Runtime comparison in seconds on Cori-GPU with 2 nodes, with GPU and CPU only for Si-214

Epsilon: Weak Scaling



Only the $O(N^4)$ part considered

Epsilon: Shift of Bottleneck on GPU vs CPU

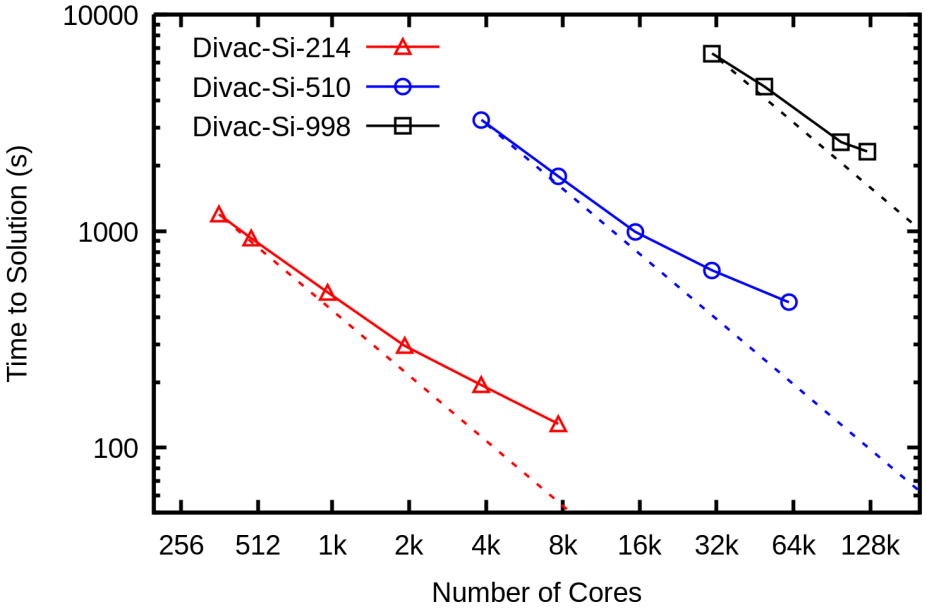
Computational Kernels			
Name	Description	FLOPs	Memory
`MTXEL`	Compute Matrix Elements $M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} \vert e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \vert \psi_{a\mathbf{k}} \rangle$	$O(N^3 \log N)$	$O(N^3)$
`CHI-Sum`	Compute RPA polarizability $\chi(\mathbf{q},\omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega)\mathbf{M}(\mathbf{q})$	$O(N^4)$	$O(N^3)$
`Inversion`	Compute inverse dielectric Function $\epsilon^{-1}(\mathbf{q},\omega_i) = (I - v\chi(\mathbf{q},\omega_i))^{-1}$	$O(N^3)$	$O(N^2)$

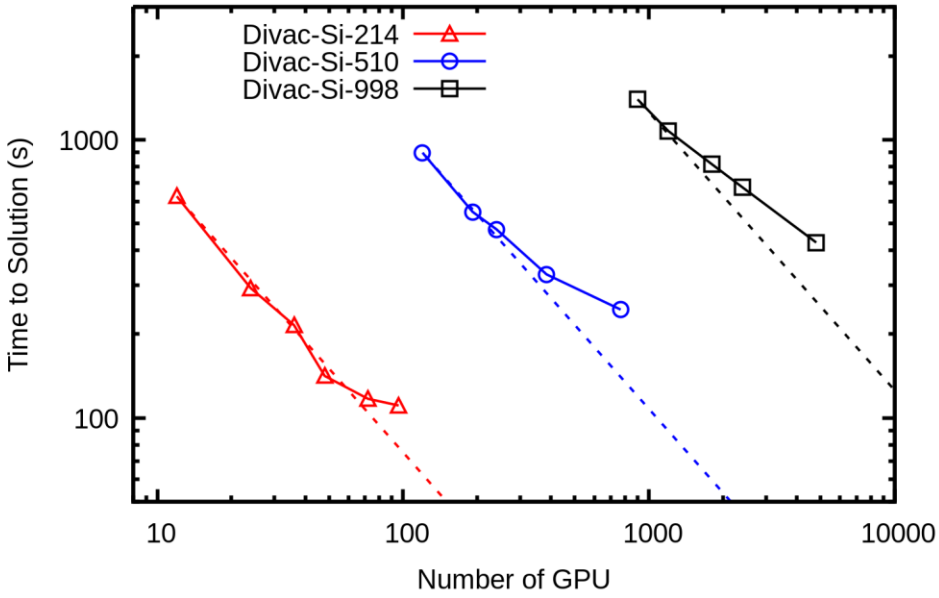
Epsilon: Shift of Bottleneck on GPU vs CPU

Computational Kernels			
Name	Description	FLOPs	Memory
`MTXEL`	Compute Matrix Elements $M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} \vert e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \vert \psi_{a\mathbf{k}} \rangle$	$O(N^3 \log N)$	$O(N^3)$
`CHI-Sum`	Compute RPA polarizability $\chi(\mathbf{q},\omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega)\mathbf{M}(\mathbf{q})$	$O(N^4)$	$O(N^3)$
`Inversion`	Compute inverse dielectric Function $\epsilon^{-1}(\mathbf{q},\omega_i) = (I - v\chi(\mathbf{q},\omega_i))^{-1}$	$O(N^3)$	$O(N^2)$

From a computational point of view Epsilon code scales $O(N^4)$ in FLOPs and $O(N^3)$ in memory wrt system size

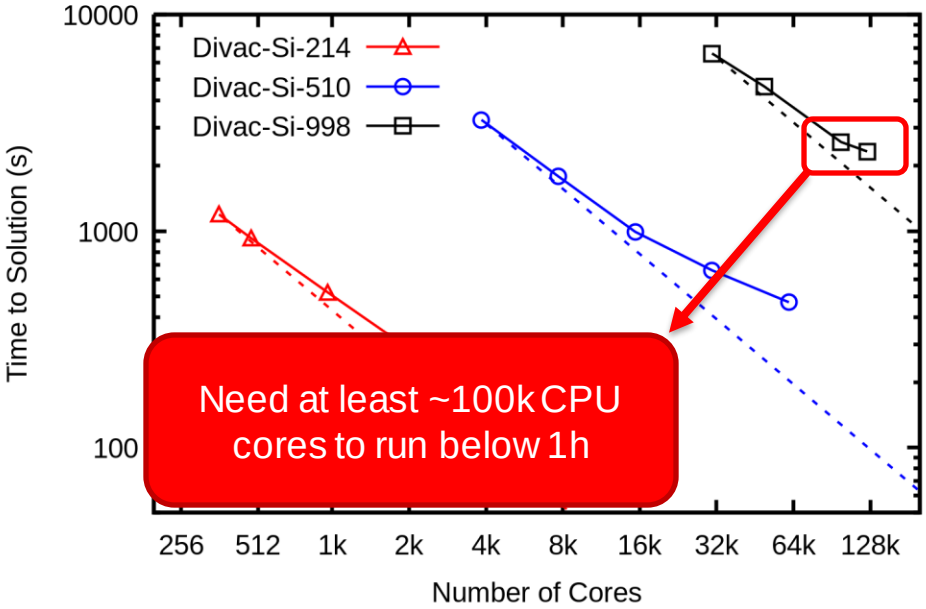
Epsilon FLOP vs Memory: GPU vs CPU

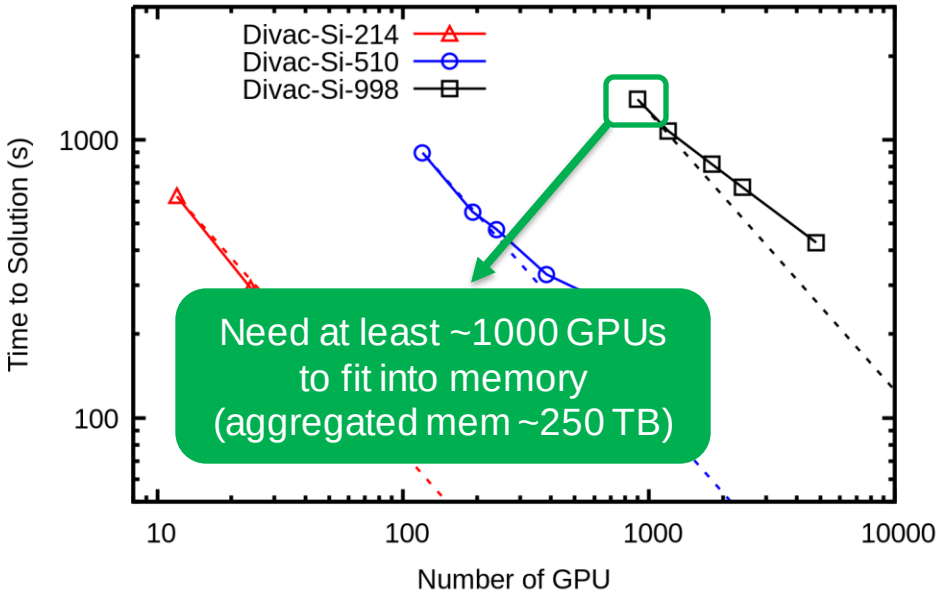




Strong scaling of Full Frequency calculation at NERSC (Edison) and OLCF (Summit) for systems of increasing size.

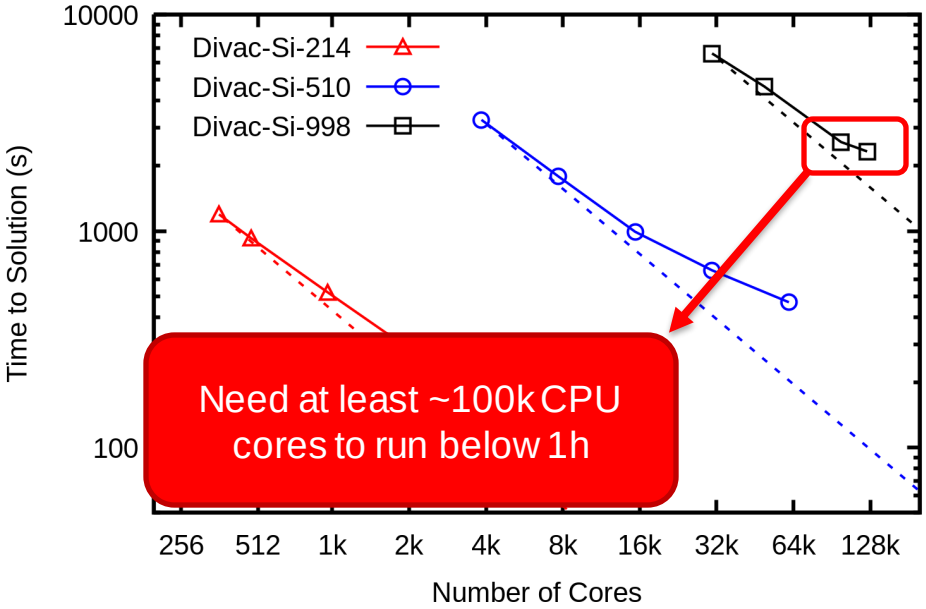
Epsilon FLOP vs Memory: GPU vs CPU

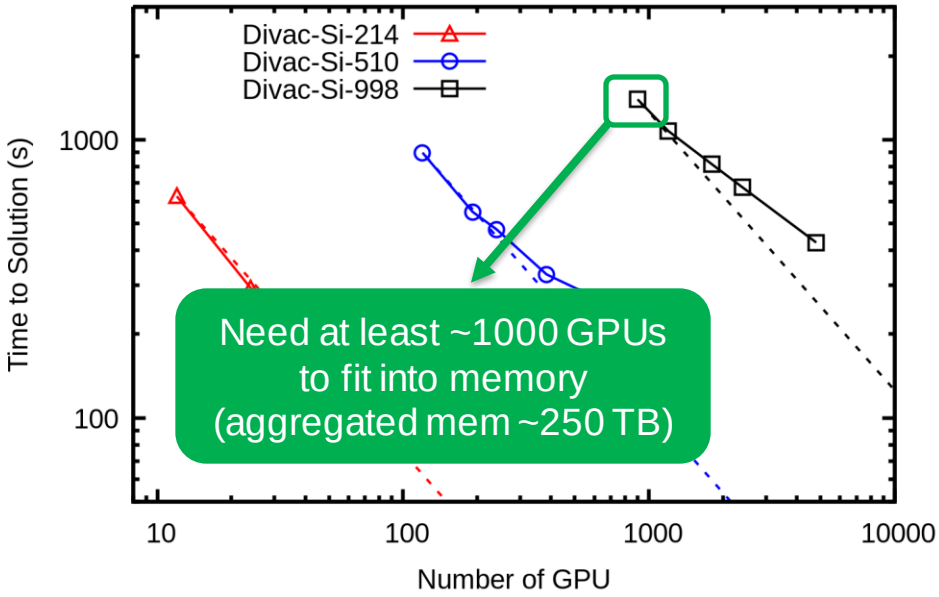




Strong scaling of Full Frequency calculation at NERSC (Edison) and OLCF (Summit) for systems of increasing size.

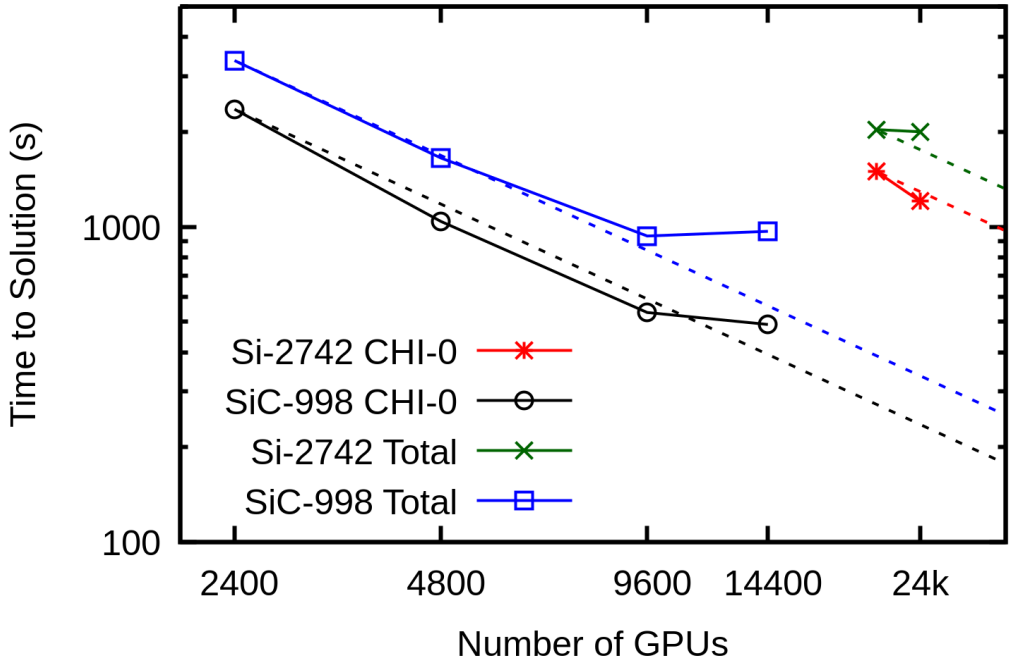
Epsilon FLOP vs Memory: GPU vs CPU





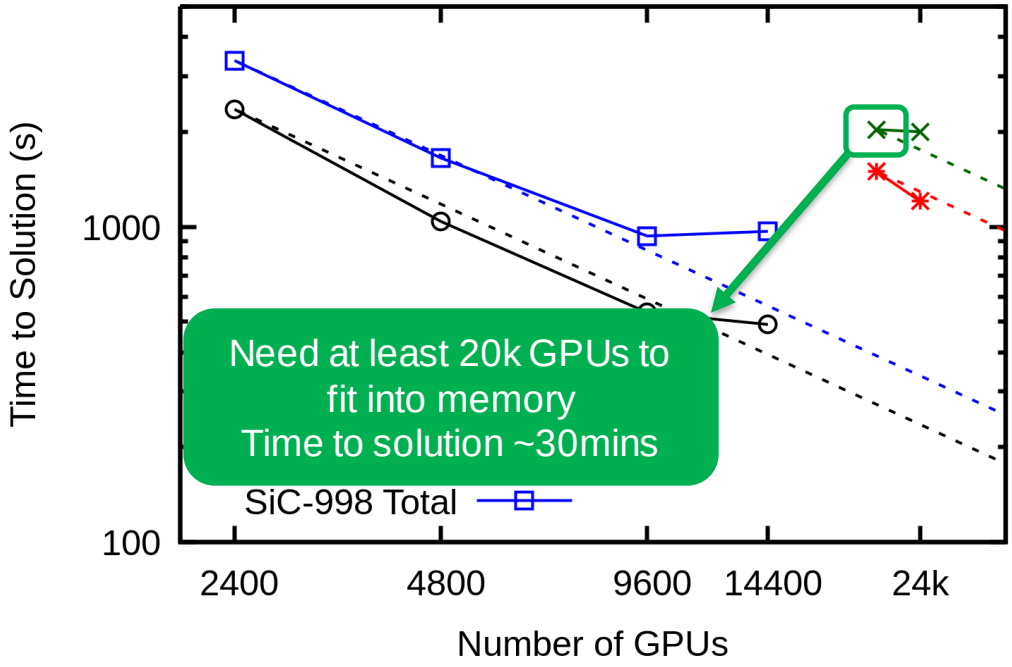
An order of magnitude acceleration on GPU architectures vs multicore has shifted the bottleneck from FLOPs to Memory

Epsilon FLOP vs Memory: GPU vs CPU



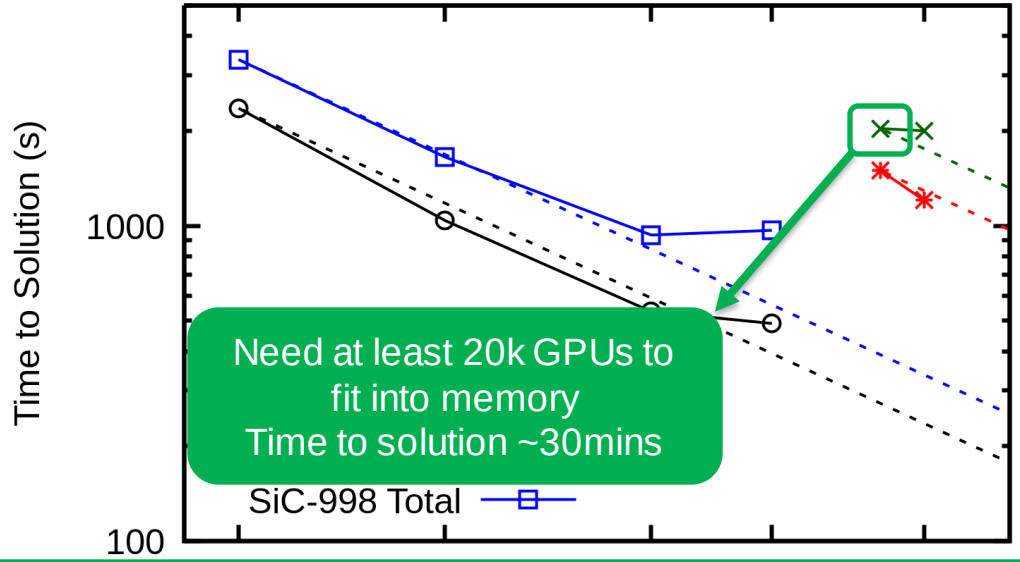
Strong scaling of static CHI-0 at OLCF (Summit) for systems of increasing size.

Epsilon FLOP vs Memory: GPU vs CPU



Strong scaling of static CHI-0 at OLCF (Summit) for systems of increasing size.

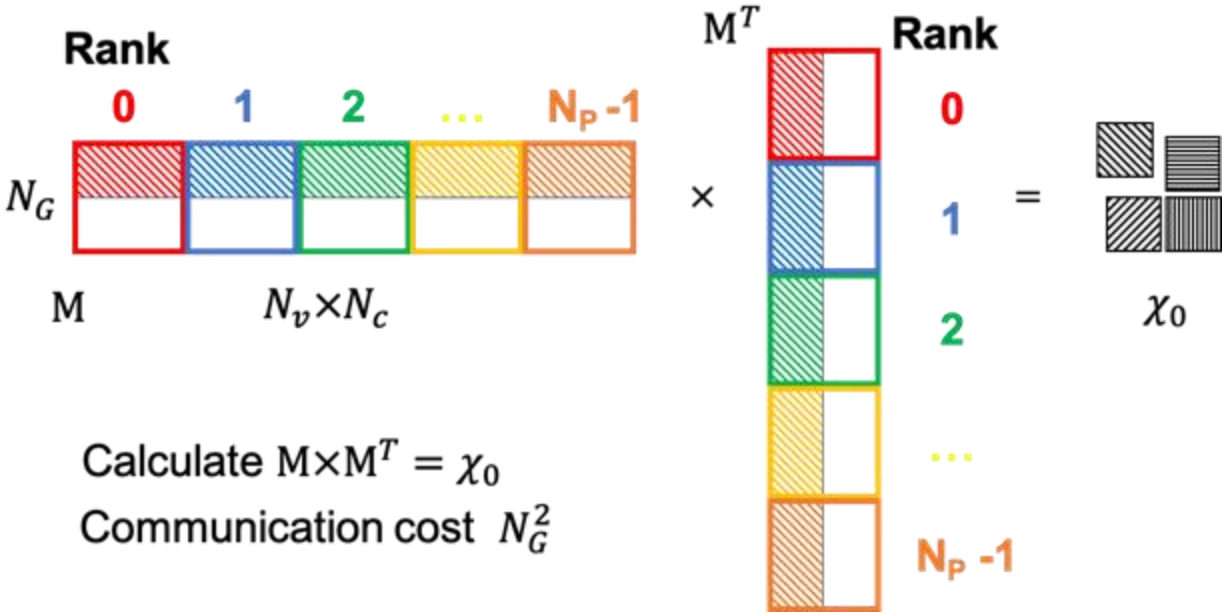
Epsilon FLOP vs Memory: GPU vs CPU



Summit has ~27k GPUs total, we need to address the memory bottleneck to:

- Push the limit forward to larger systems
- Allow for medium/large scale calculations to be possible without the need of massive machine/allocation

The NVblock Algorithm



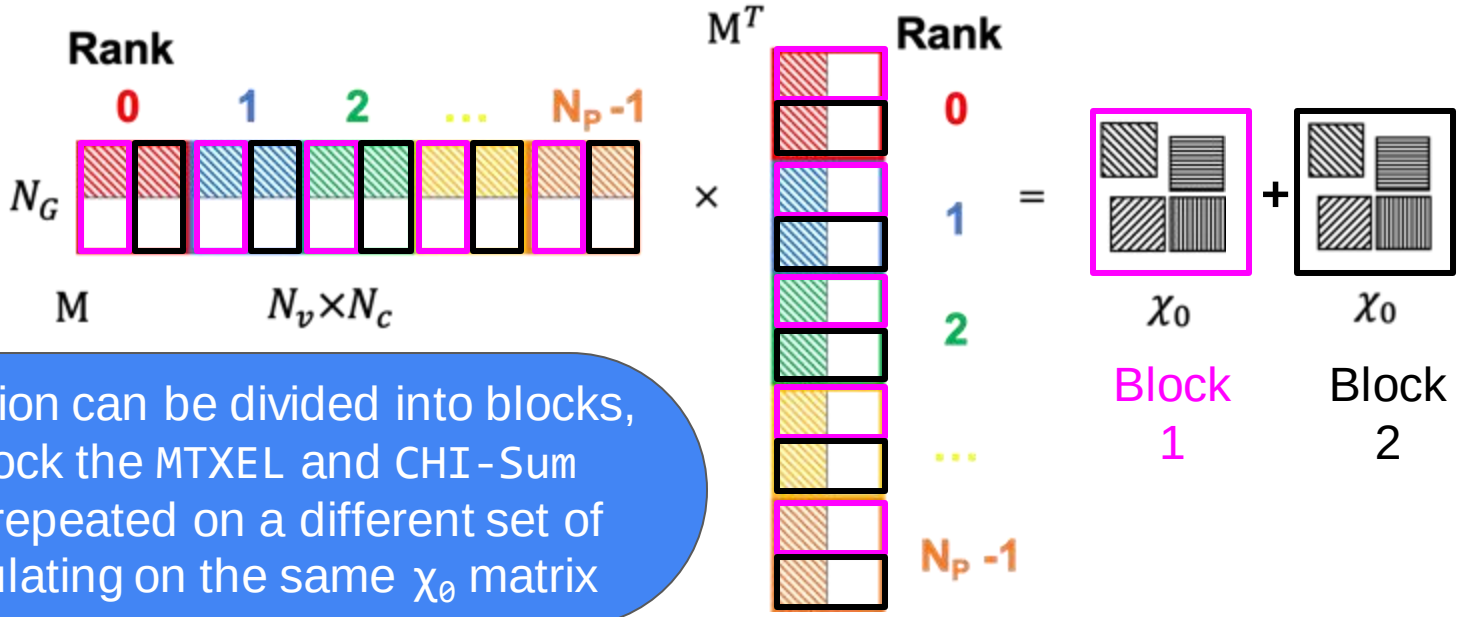
Calculate $M \times M^T = \chi_0$

Communication cost N_G^2

Original Implementation:

- `MTXEL` kernel computes all matrix elements (`gme` array) organized into a short-fat matrix
- `CHI-Sum` kernel contracts the matrix elements to compute the polarizability
- All `gme` are necessary by `CHI-Sum` -> $O(N^3)$ memory bottleneck

The NVblock Algorithm



The computation can be divided into blocks, for each block the `MTXEL` and `CHI-Sum` kernels are repeated on a different set of `gme` accumulating on the same χ_0 matrix

Original Implementation:

- `MTXEL` kernel computes all matrix elements (`gme` array) organized into a short-fat matrix
- `CHI-Sum` kernel contracts the matrix elements to compute the polarizability
- All `gme` are necessary by `CHI-Sum` -> $O(N^3)$ memory bottleneck

The NVblock Algorithm

Implementation to alleviate the $O(N^3)$ memory bottleneck in epsilon:

- Computation is divided over **N** blocks of **V**alence bands (***NV block***)
- Block size depends on available memory
- Each batch repeated the same communication/computation pattern

The NVblock Algorithm

Implementation to alleviate the $O(N^3)$ memory bottleneck in epsilon:

- Computation is divided over **N** blocks of **V**alence bands (***NV block***)
- Block size depends on available memory
- Each batch repeated the same communication/computation pattern

Given a Np MPI ranks and a max memory per rank:

- Define number of blocks (Nb) depending on max memory per rank
- For a given max memory per rank: Nb x Np = constant
- Memory: $O(N^3/Nb/Np)$
- Communication: $O(N^2/Nb)$
- FLOPs: $O(N^4/Np)$

The NVblock Algorithm

Implementation to alleviate the $O(N^3)$ memory bottleneck in epsilon:

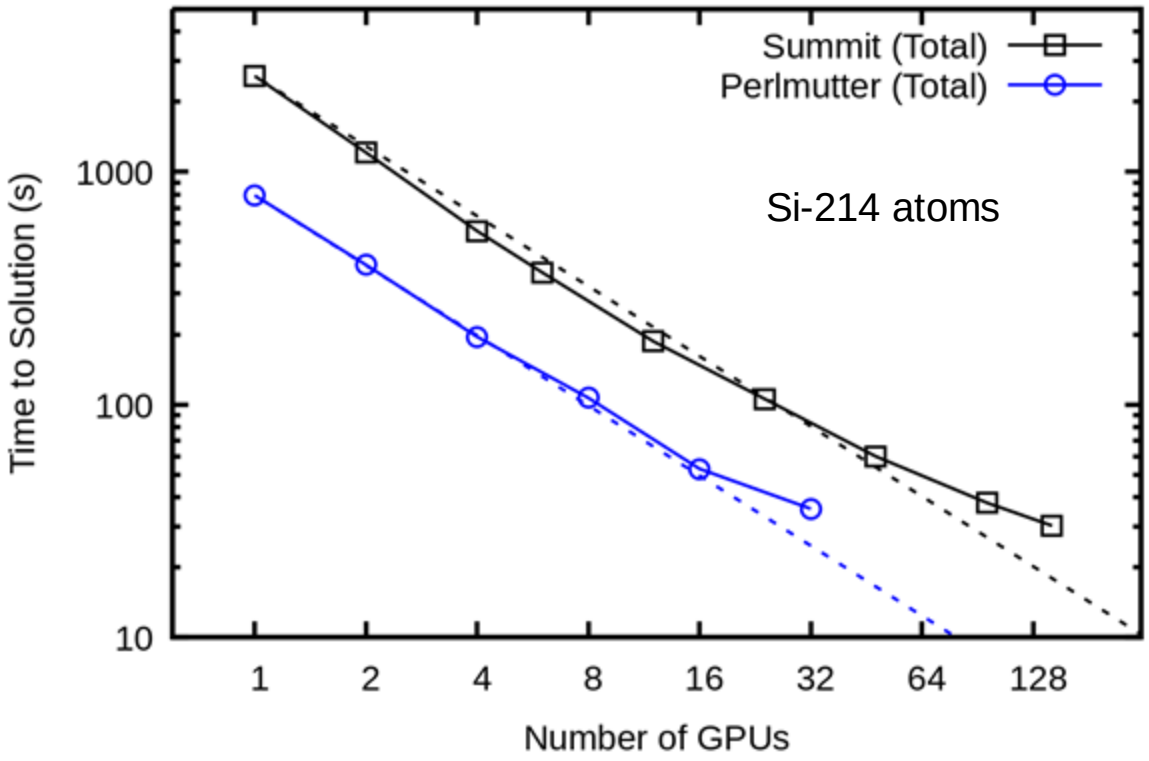
- Computation is divided over **N** blocks of **V**alence bands (***NV block***)
- Block size depends on available memory
- Each batch repeated the same communication/computation pattern

Given a Np MPI ranks and a max memory per rank:

- Define number of blocks (Nb) depending on max memory per rank
- For a given max memory per rank: Nb x Np = constant
- Memory: $O(N^3/Nb/Np)$
- Communication: $O(N^2/Nb)$
- FLOPs: $O(N^4/Np)$

In the limit of Nb = Nv bands the memory requirements becomes $O(N^2)$

The NVblock Algorithm: Strong Scaling Performance

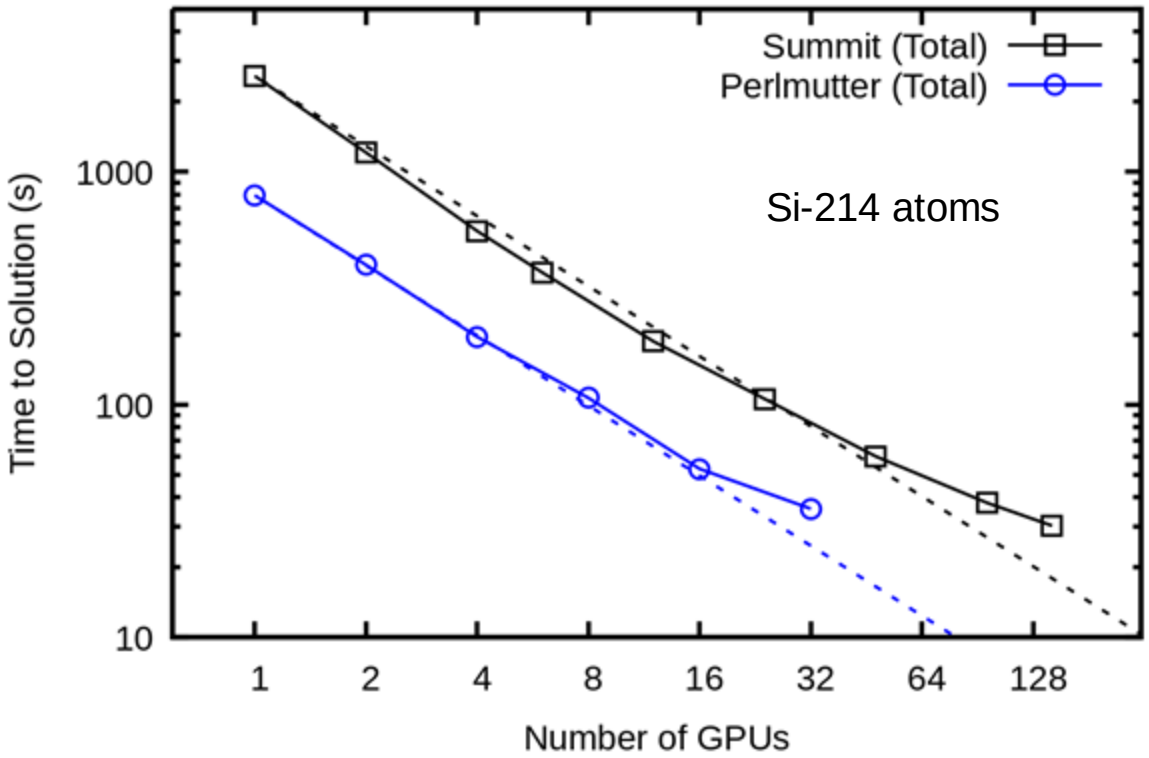


- Calculation can be fit on a single GPU (On summit originally at least 12 GPUs were necessary)
- Excellent parallel scalability over a wide range of GPUs
- Perlmutter vs Summit: 3.3x speed-up:
  - Better FLOP rate power
  - ~2x more memory per GPU

Strong Scaling of epsilon measured on Summit@OLCF and Perlmutter@NERSC

- Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter Node: 1 AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs

The NVblock Algorithm: Strong Scaling Performance

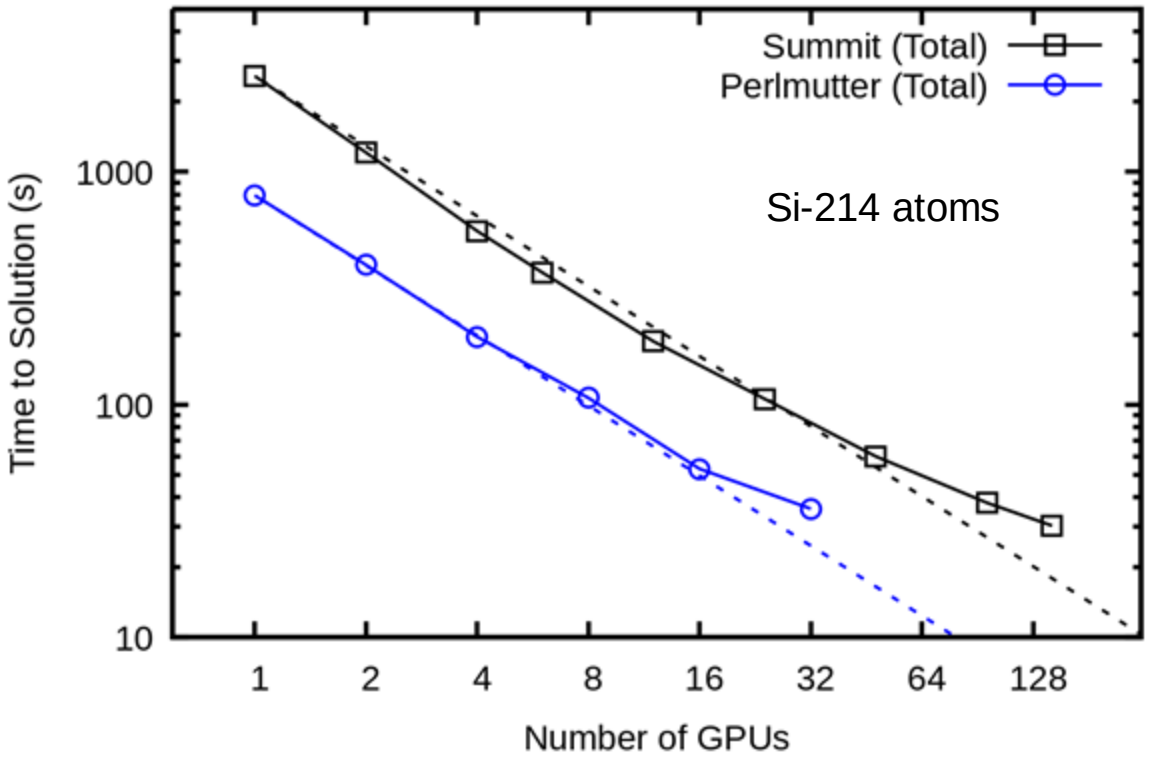


- **Calculation can be fit on a single GPU (On summit originally at least 12 GPUs were necessary)**
- Excellent parallel scalability over a wide range of GPUs
- Perlmutter vs Summit: 3.3x speed-up:
  - Better FLOP rate power
  - ~2x more memory per GPU

Strong Scaling of epsilon measured on Summit@OLCF and Perlmutter@NERSC

- Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter Node: 1 AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs

The NVblock Algorithm: Strong Scaling Performance

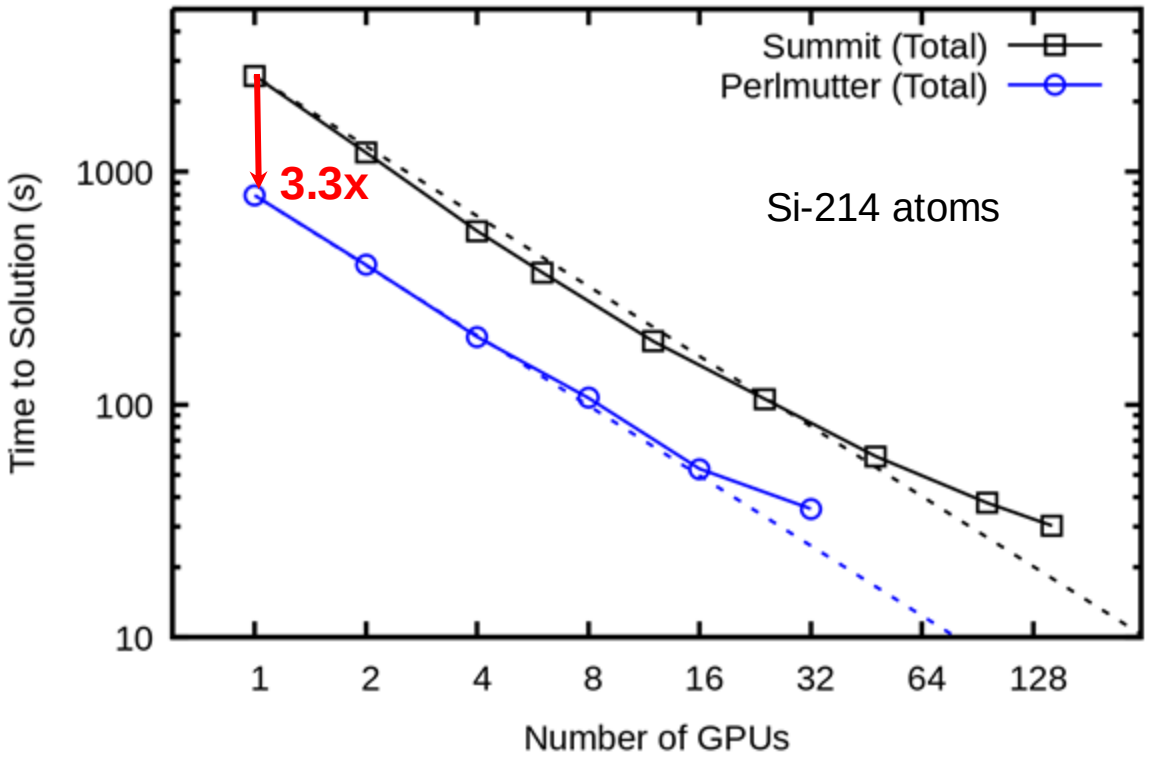


- Calculation can be fit on a single GPU (On summit originally at least 12 GPUs were necessary)
- **Excellent parallel scalability over a wide range of GPUs**
- Perlmutter vs Summit: 3.3x speed-up:
  - Better FLOP rate power
  - ~2x more memory per GPU

Strong Scaling of epsilon measured on Summit@OLCF and Perlmutter@NERSC

- Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter Node: 1 AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs

The NVblock Algorithm: Strong Scaling Performance

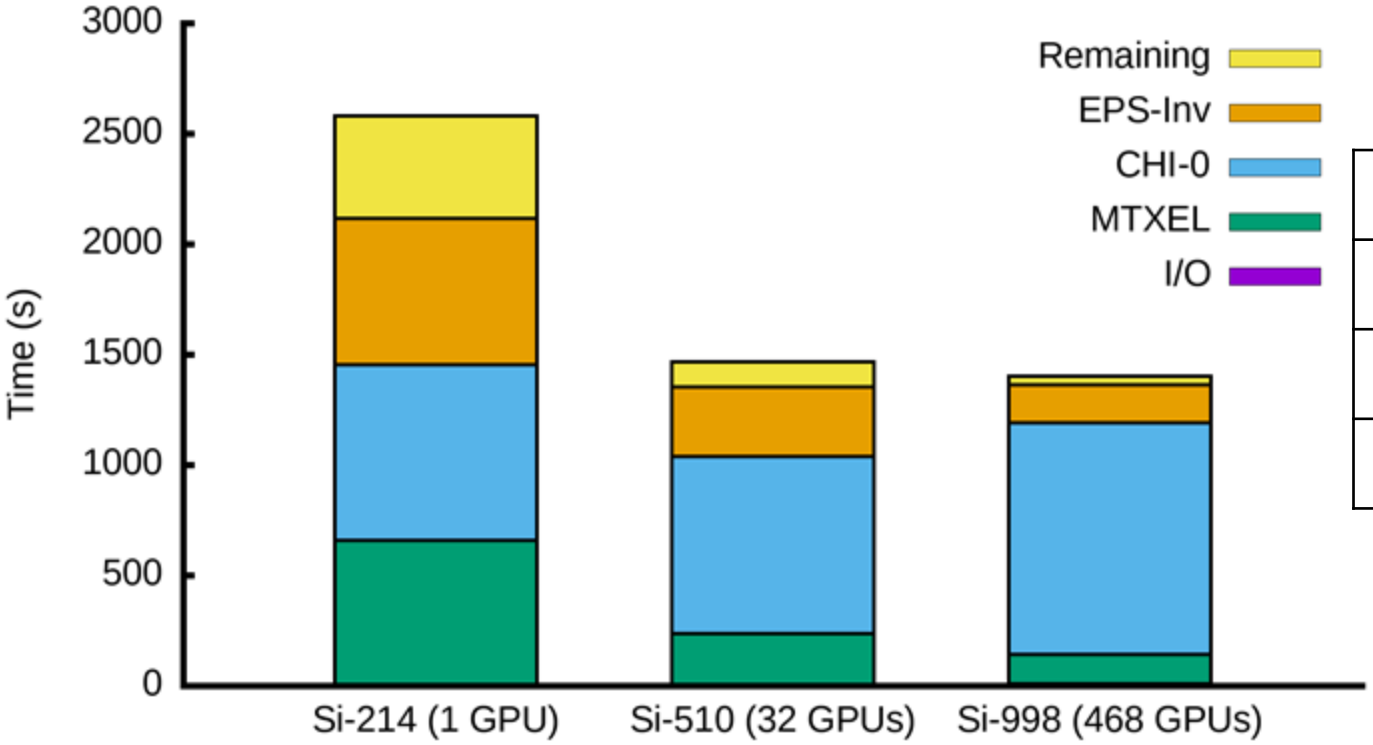


- Calculation can be fit on a single GPU (On summit originally at least 12 GPUs were necessary)
- Excellent parallel scalability over a wide range of GPUs
- **Perlmutter vs Summit: 3.3x speed-up:**
  - **Better FLOP rate power**
  - **~2x more memory per GPU**

Strong Scaling of epsilon measured on Summit@OLCF and Perlmutter@NERSC

- Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter Node: 1 AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs

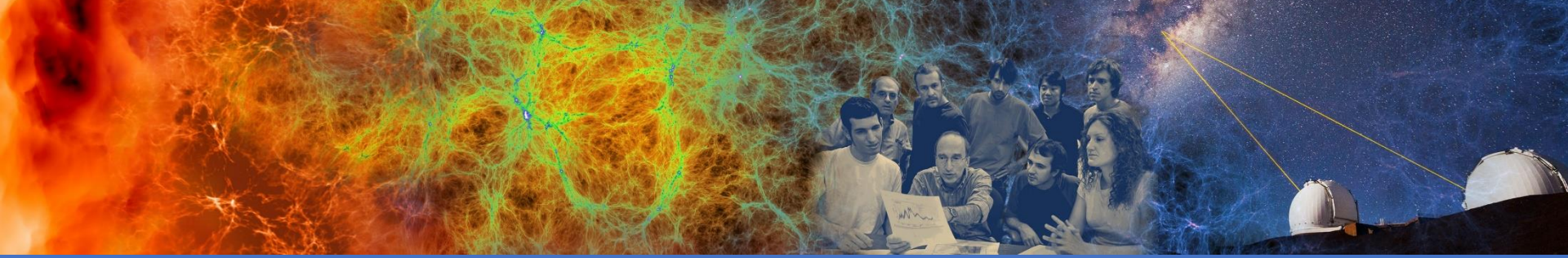
The NVblock Algorithm: Weak Scaling Performance



	CHI-Sum - $O(N^4)$
Si-214	795
Si-510	801
Si-998	1047

Time in seconds.

Weak Scaling of epsilon measured on Summit@OLCF. Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs.



Sigma:
System, Node and GPU Implementation

The GW Workflow: Epsilon + Sigma

Dynamical properties of electrons as solution of Dyson's equation:

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\,\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

GW Self-Energy Operator Σ : non-Hermitian, non-local, frequency dependent

(Note: In DFT, the role of self-energy is replaced by static and local $V_{\mathrm{xc}}(\mathbf{r})$)

High GW Computational Cost in Two Major Bottlenecks:

- **Epsilon**: Inverse Dielectric Matrix **O(N^4)**
- **Sigma**: Self-Energy Matrix Elements **O(N^4)**

ϵ^{-1} matrix

Sigma: Quasiparticle Properties

Compute a set (100-1000) of Self-Energy matrix elements to solve the Dyson equation

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

Each Self-Energy matrix element:

$$\Sigma_{lm}(E) = \frac{i}{2\pi}\int_0^\infty d\omega \sum_n \sum_{GG'} M_{nl}^{-G} \frac{\epsilon_{GG'}^{-1}(\omega)\cdot v(G')}{E - E_n - \omega} M_{nm}^{-G'}$$

Sigma: Quasiparticle Properties

Compute a set (100-1000) of Self-Energy matrix elements to solve the Dyson equation

$$h_0(\mathbf{r})\phi_n(\mathbf{r}) + \int \Sigma(\mathbf{r}, \mathbf{r}'; E_n)\phi_n(\mathbf{r}')\mathrm{d}\mathbf{r}' = E_n\phi_n(\mathbf{r})$$

Each Self-Energy matrix element:

$$\Sigma_{lm}(E) = \frac{i}{2\pi}\int_0^\infty d\omega \sum_n \sum_{GG'} M_{nl}^{-G} \frac{\epsilon_{GG'}^{-1}(\omega)\cdot v(G')}{E - E_n - \omega} M_{nm}^{-G'}$$

Frequency treatment

- Generalized Plasmon Pole (GPP) model
  - Analytical approximation to the frequency dependence
  - Require only the static dielectric matrix
- Full-Frequency (FF) model
  - Analytical integration over frequency (Contor-Deformation)
  - Require frequency dependent dielectric matrix

Breaking Down the Portability Strategy

Sigma

Compute a set of (100-1000) Self-Energy matrix elements

$$\Sigma_{lm} = \sum_{n}^{N_b} \sum_{GG'}^{N_G} M^{l*}_{Gn} [P(E_n)]_{GG'} M^{m}_{G'n}$$

The **GPP kernel**: a big data reduction across different matrices with a complex matrix-vector interdependence

Sigma: The GPP Kernel

Compute a set of (100-1000) Self-Energy matrix elements

Two Level Parallelization Strategy:

- Inter-Pool parallelization (*independent self-energy matrix elements*)
- Intra-Pool parallelization (*same self-energy matrix elements*)

Sigma: The GPP Kernel

Inter-Pool Parallelization

16 MPI tasks, 2 pools, 4 Self-Energies: $\{\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4\}$

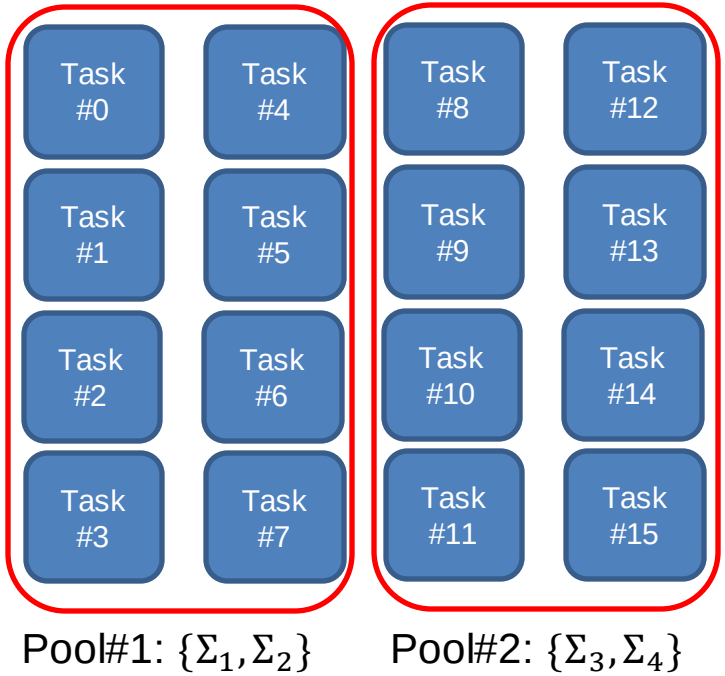
Two Level Parallelization Strategy:

- **Inter-Pool parallelization** (*independent self-energy matrix elements*)
- Intra-Pool parallelization (*same self-energy matrix elements*)

Sigma: The GPP Kernel

Inter-Pool Parallelization

16 MPI tasks, 2 pools, 4 Self-Energies: $\{\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4\}$

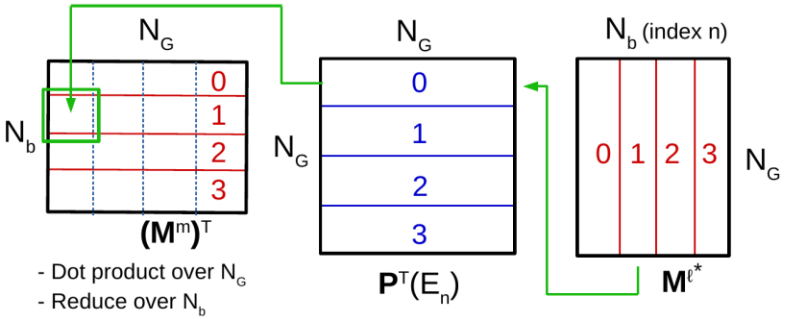


Two Level Parallelization Strategy:

- **Inter-Pool parallelization** (*independent self-energy matrix elements*)
- Intra-Pool parallelization (*same self-energy matrix elements*)

Sigma: The GPP Kernel

Intra-Pool data layout

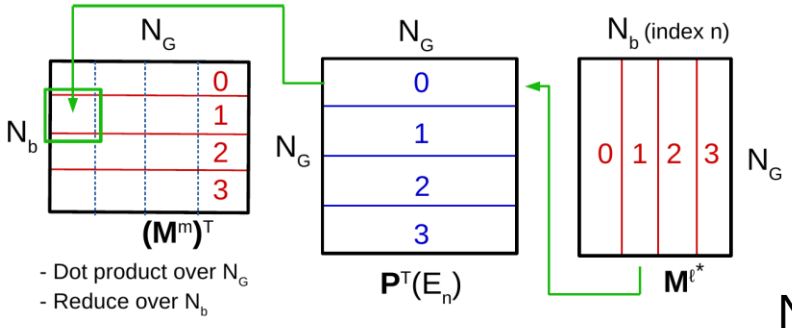


Two Level Parallelization Strategy:

- Inter-Pool parallelization (*independent self-energy matrix elements*)
- **Intra-Pool parallelization** (*same self-energy matrix elements*)

Sigma: The GPP Kernel

Intra-Pool data layout



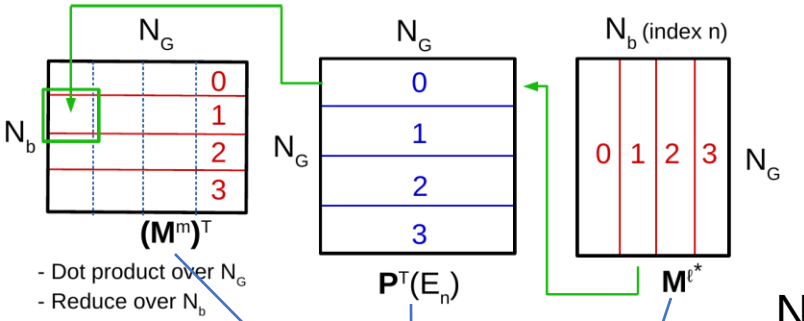
Two Level Parallelization Strategy:

- Inter-Pool parallelization (*independent self-energy matrix elements*)
- **Intra-Pool parallelization** (*same self-energy matrix elements*)

Non-Blocking Cyclic communication: **Overlap MPI communication (host) with computation (GPU)**

Sigma: The GPP Kernel

Intra-Pool data layout



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

Two Level Parallelization Strategy:

- Inter-Pool parallelization (*independent self-energy matrix elements*)
- Intra-Pool parallelization (*same self-energy matrix elements*)

Non-Blocking Cyclic communication: Overlap MPI communication (host) with computation (GPU)

GPP-Kernel: Compute Intensive part entirely on GPU

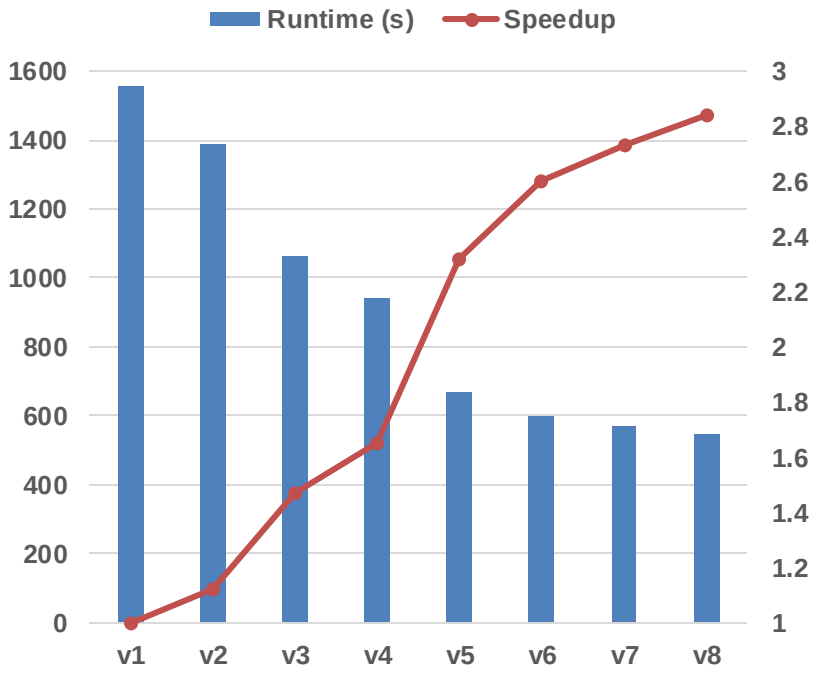
GPP Kernel: Computational Characteristics

```
# 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

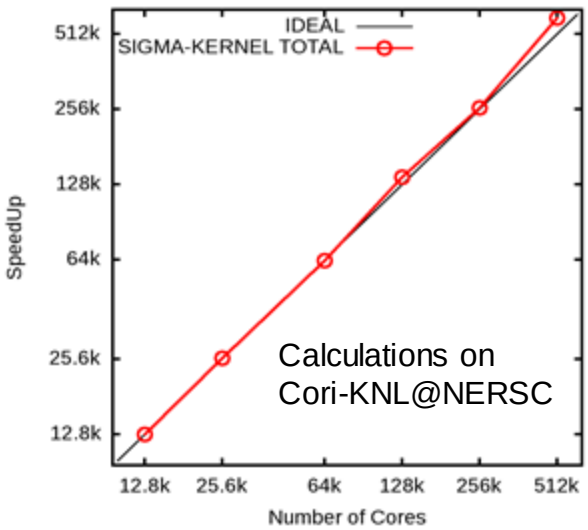
Optimization Path

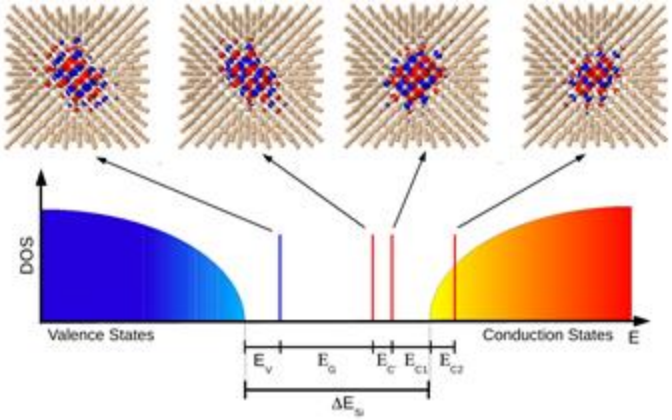
1. Baseline*
2. Replace divides with reciprocals
3. Replace square roots with power of 2
4. Replace divides and square roots
5. Loop re-ordering
6. Further increase occupancy
7. Cache blocking
8. Add more arrays to shared memory



*Collapse 3 of the other loops

Sigma: Performance (FF)





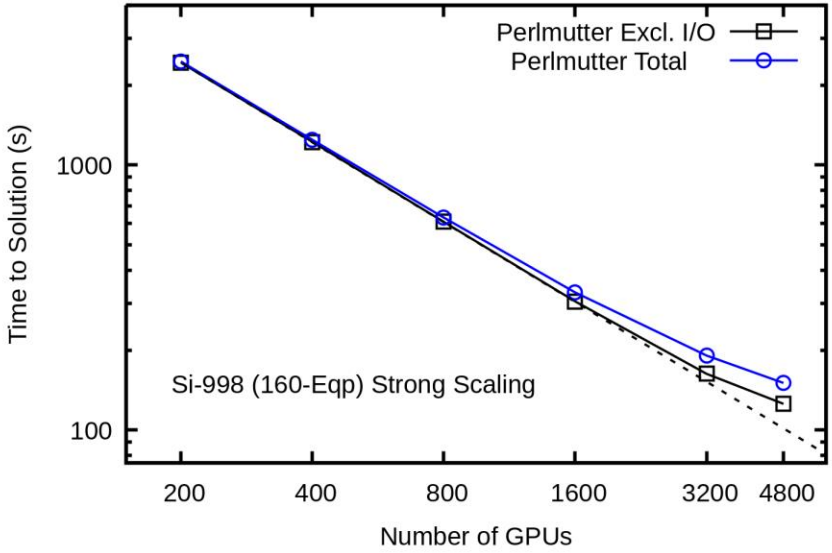
	998 Si	1726 Si
Number KNL Nodes	9600	9500
Number of Cores	633,600	627,000
Number E_{qp} Evaluated	48	38
Time to solution (s)	160	201
Peta FLOP/s	11.8	11.2
% Peak AVX Performance	47	45

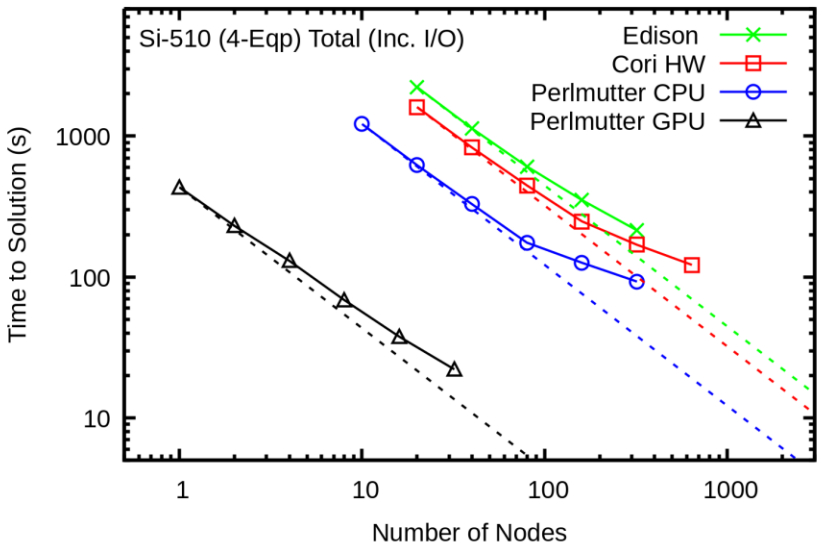
Divacancy in Silicon

- Large reconstruction
- Supercell containing 998/1726 atoms
- Calculation parameters:
  N_G = 100k ; N_c = 37k ; N_b = 12k ; Nfreq = 35

M. Del Ben, F.H. da Jornada, A. Canning, N. Wichmann, K. Raman, R. Sasanka, C. Yang, S.G. Louie and J. Deslippe, *Comput. Phys. Commun.* **235**, 187-195 (2019)

Sigma: Performance on Perlmutter (GPU)

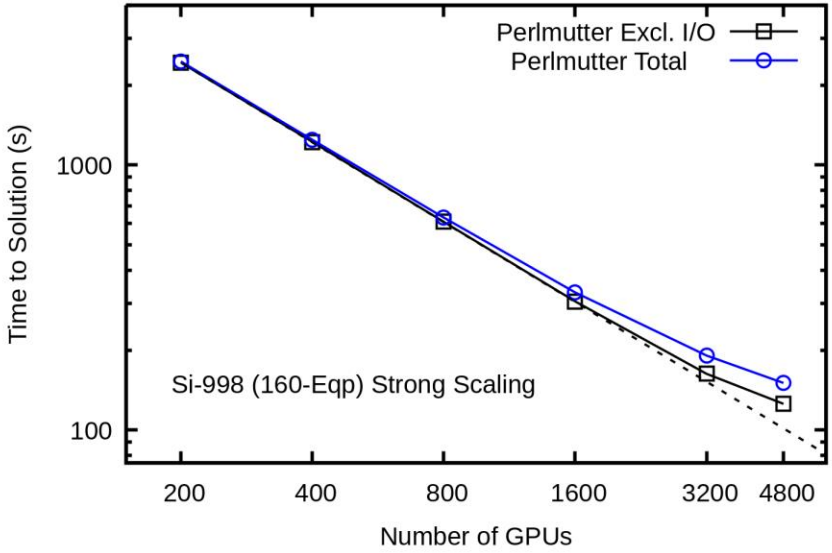


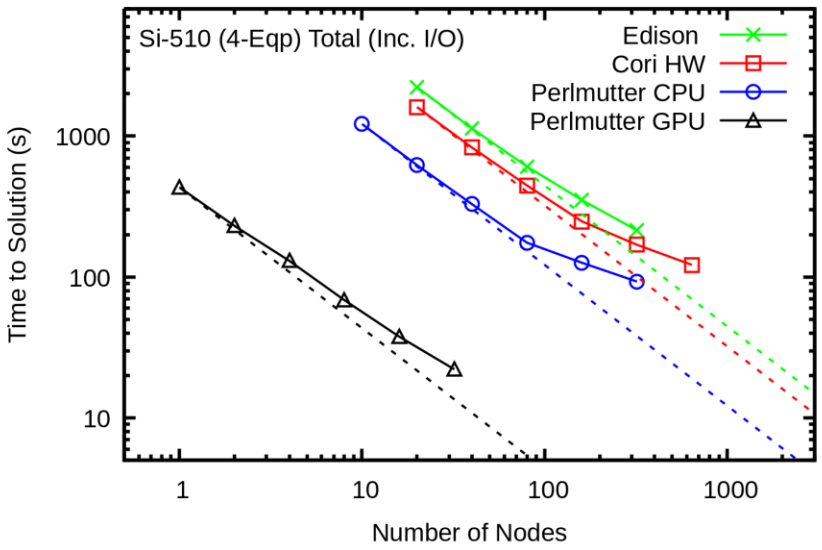


Strong Scaling for Sigma measured on Perlmutter@NERSC (Cray Shasta, Node: 2 AMD Milan + 4-A100 GPUs)

- Left: Strong scaling to (almost) entire Perlmutter
- Right: Comparison between CPU (Cori-Haswell) and GPU (Perlmutter)

Sigma: Performance on Perlmutter (GPU)

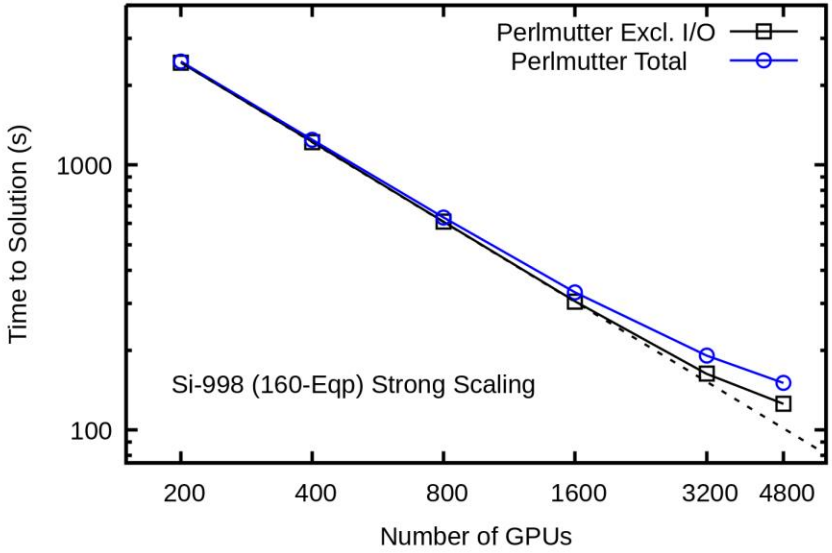


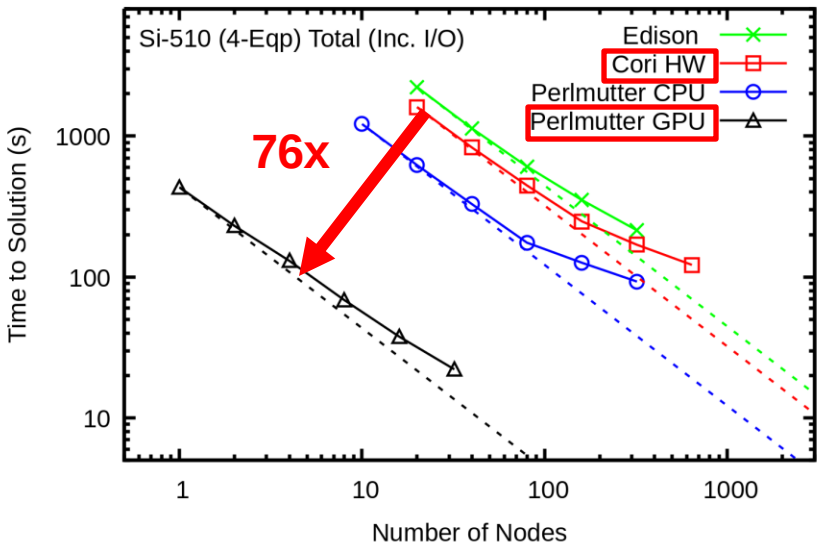


Strong Scaling for Sigma measured on Perlmutter@NERSC (Cray Shasta, Node: 2 AMD Milan + 4-A100 GPUs)

- **Left: Strong scaling to (almost) entire Perlmutter**
- Right: Comparison between CPU (Cori-Haswell) and GPU (Perlmutter)

Sigma: Performance on Perlmutter (GPU)

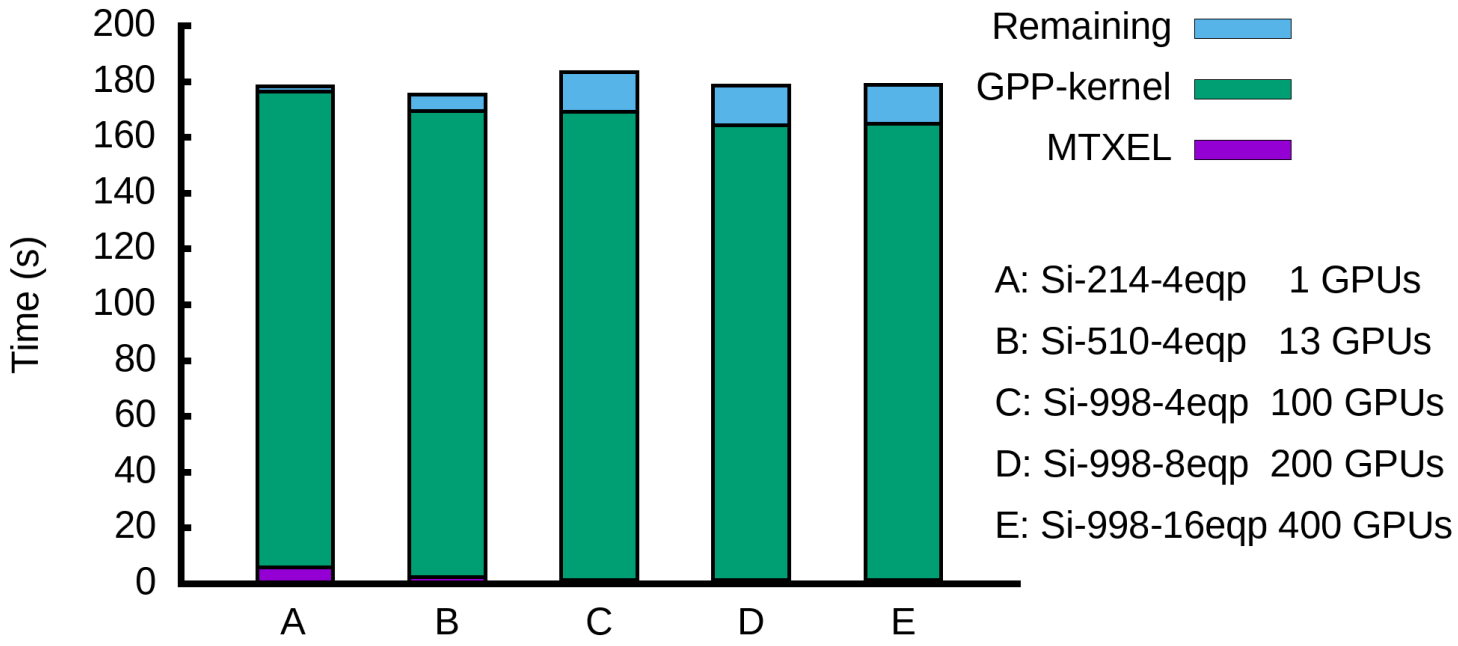




Strong Scaling for Sigma measured on Perlmutter@NERSC (Cray Shasta, Node: 2 AMD Milan + 4-A100 GPUs)

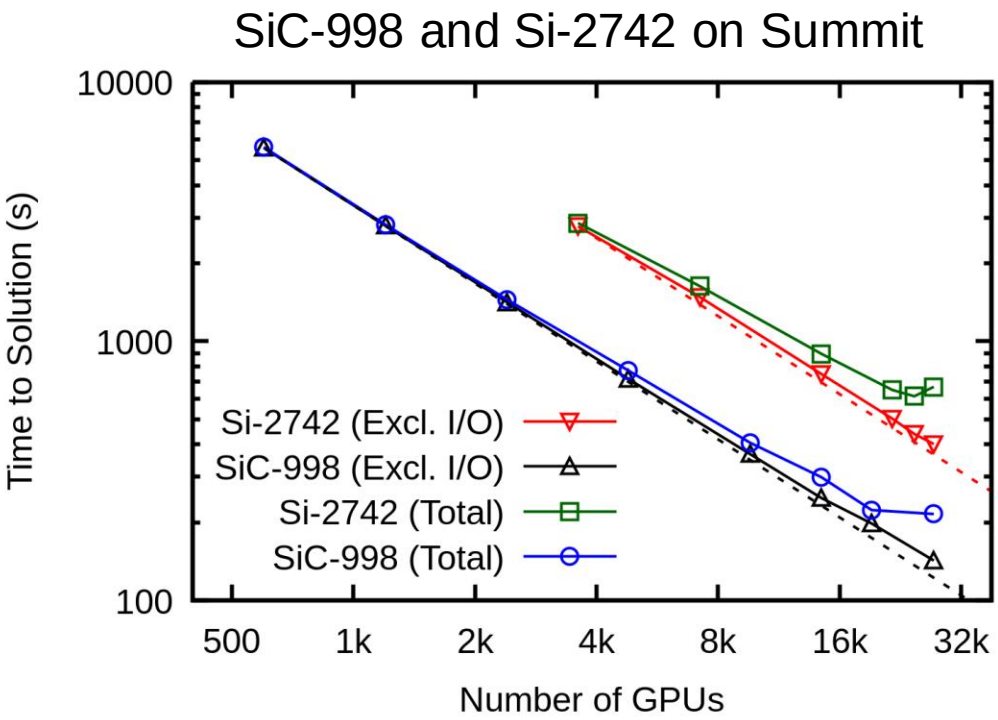
- Left: Strong scaling to (almost) entire Perlmutter
- **Right: Comparison between CPU (Cori-Haswell) and GPU (Perlmutter)**

Sigma Weak Scaling



Only the $O(N^4)$ part considered

Sigma: Strong Scaling (Summit)



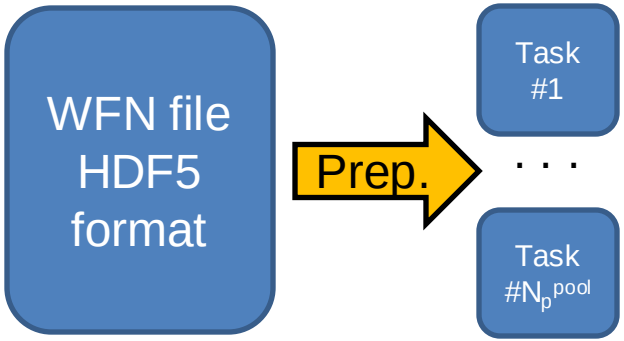
Scaling up to 4560 Summit nodes (27,360 GPU's) 99% of Summit.

I/O Optimization

- Exploit node local solid-state memory (SSD)
  - Prepare data → distributed form, at no cost
  - Prestage data → node local SSD
  - Runtime → Each rank read directly from local SSD

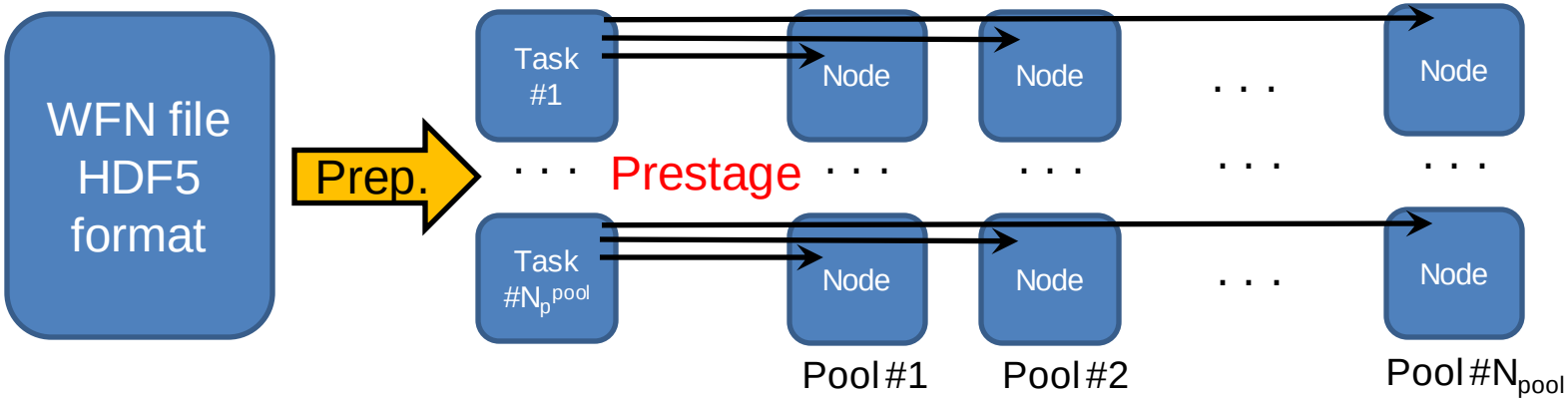
I/O Optimization

- Exploit node local solid-state memory (SSD)
  - **Prepare data** → distributed form, at no cost
  - Prestage data → node local SSD
  - Runtime → Each rank read directly from local SSD



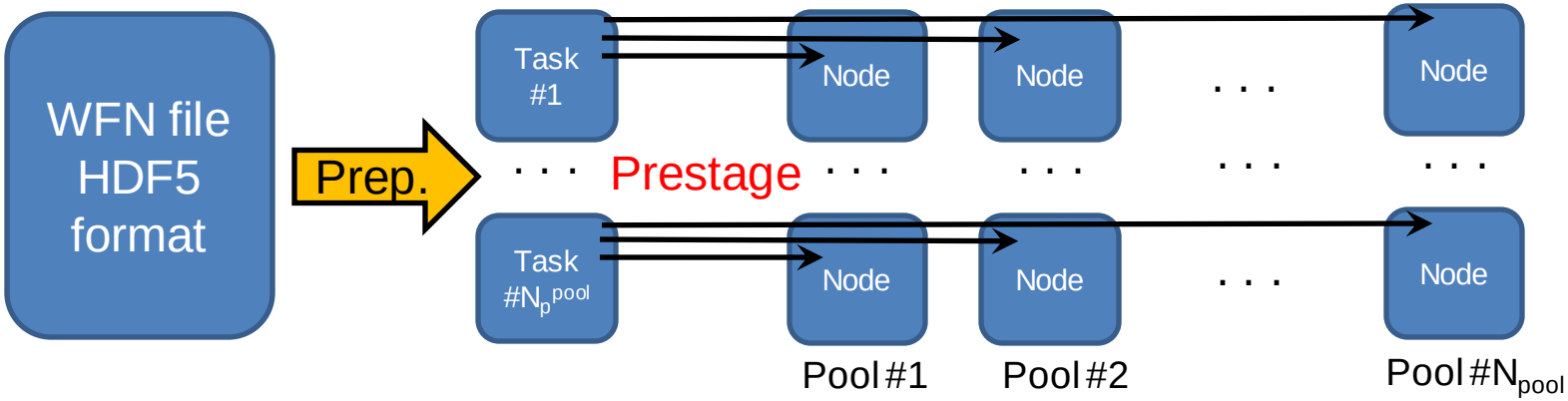
I/O Optimization

- Exploit node local solid-state memory (SSD)
  - Prepare data → distributed form, at no cost
  - **Prestage data** → node local SSD
  - Runtime → Each rank read directly from local SSD

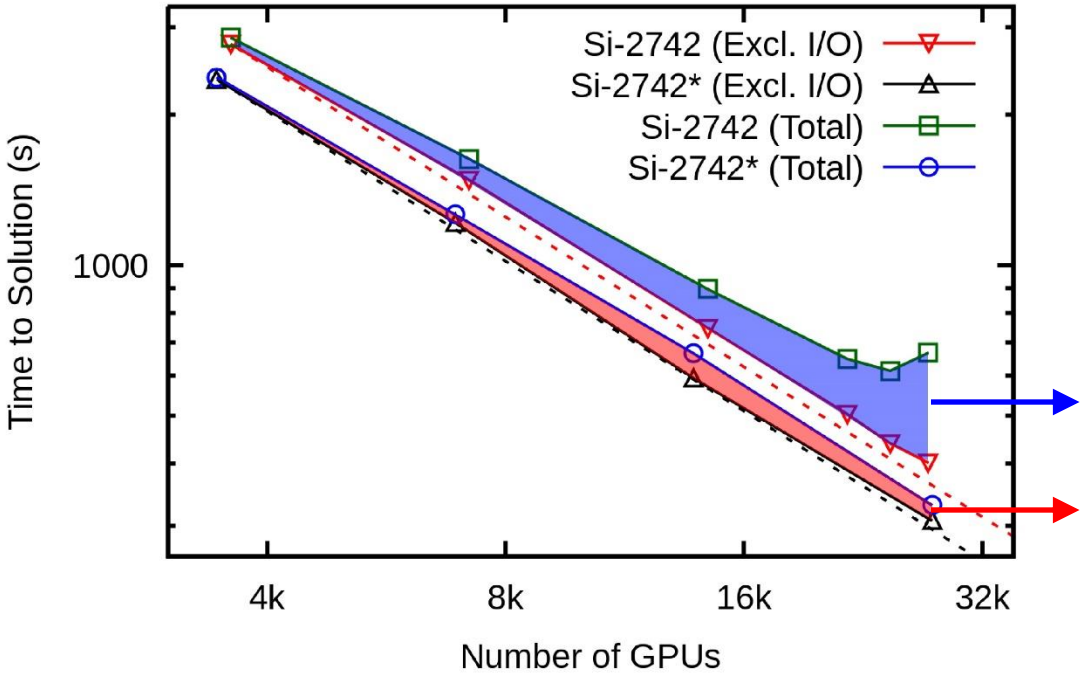


I/O Optimization

- Exploit node local solid-state memory (SSD)
  - Prepare data → distributed form, at no cost
  - Prestage data → node local SSD
  - **Runtime** → Each rank read directly from local SSD



BerkeleyGW: Full Application at Scale



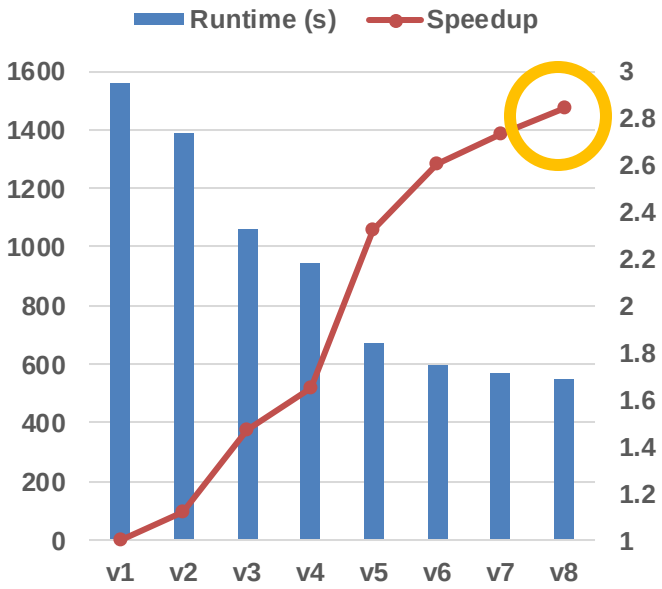
I/O Optimization (SSD):
266s to 23s

(Blue shade) **no** Prestage to SSD

(Red shade) **with** Prestage to SSD

*Results with optimized I/O include some further optimization in GPP-kernel (v8)

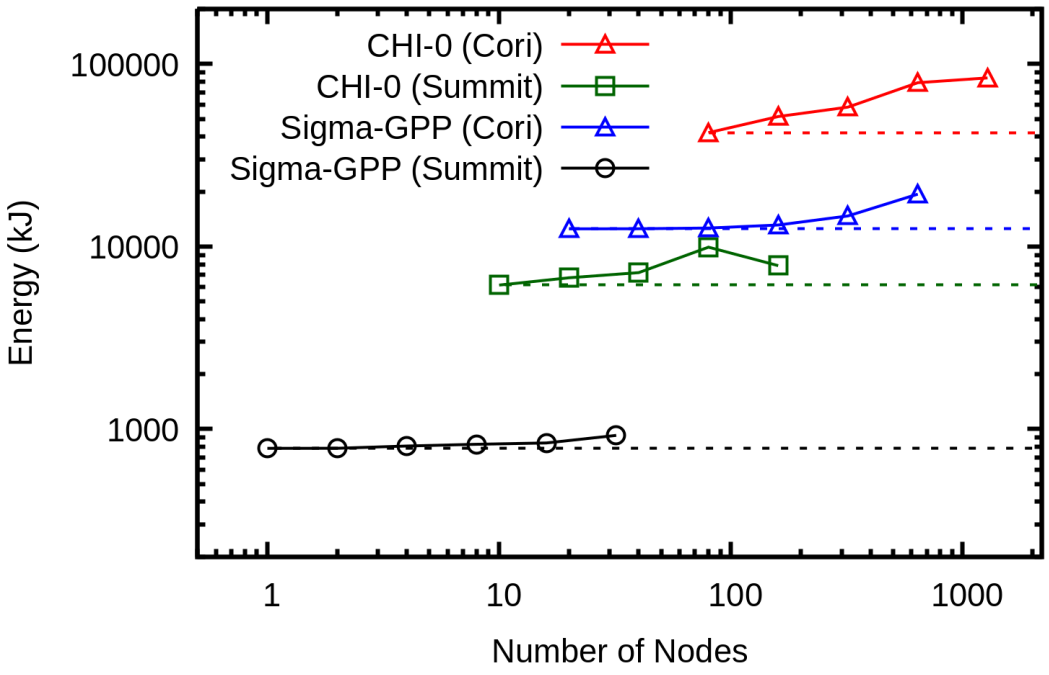
BerkeleyGW: Full System Run



Application: 105.9 PFLOP/s

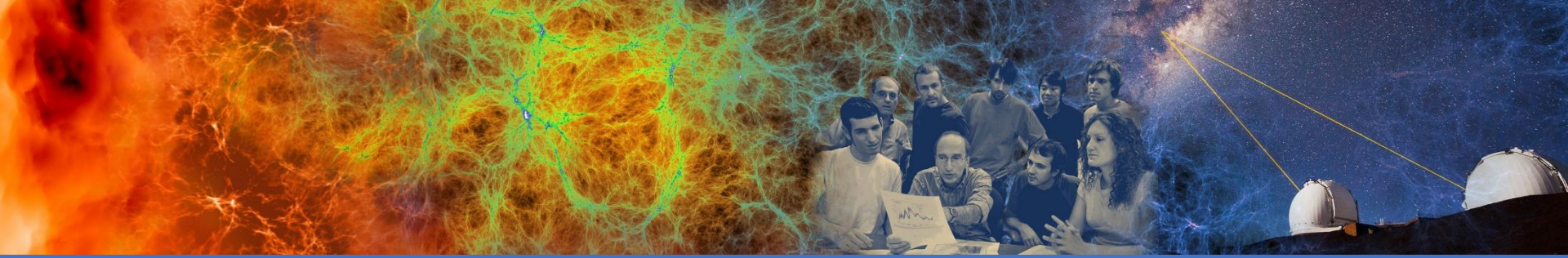
Application	BerkeleyGW
Benchmark	Si-2742
# Eqp	256
# of GPUs	27,648 (full Summit)
Pool Size	108 GPUs
Compute Time	592 s
I/O Time	39 s
Throughput	105.9 PFLOP/s
% of R_{peak}	52.7% of 200.79 PFLOP/s

Energy Efficiency



- Compare two implementations across architectures in term of energy consumption per calculation
- Average Energy per node 2,200 kW Summit, 400 kW Cori HW
- Total energy = Avg Power * Time to solution
- Ideal scaling is constant energy increasing the number of nodes

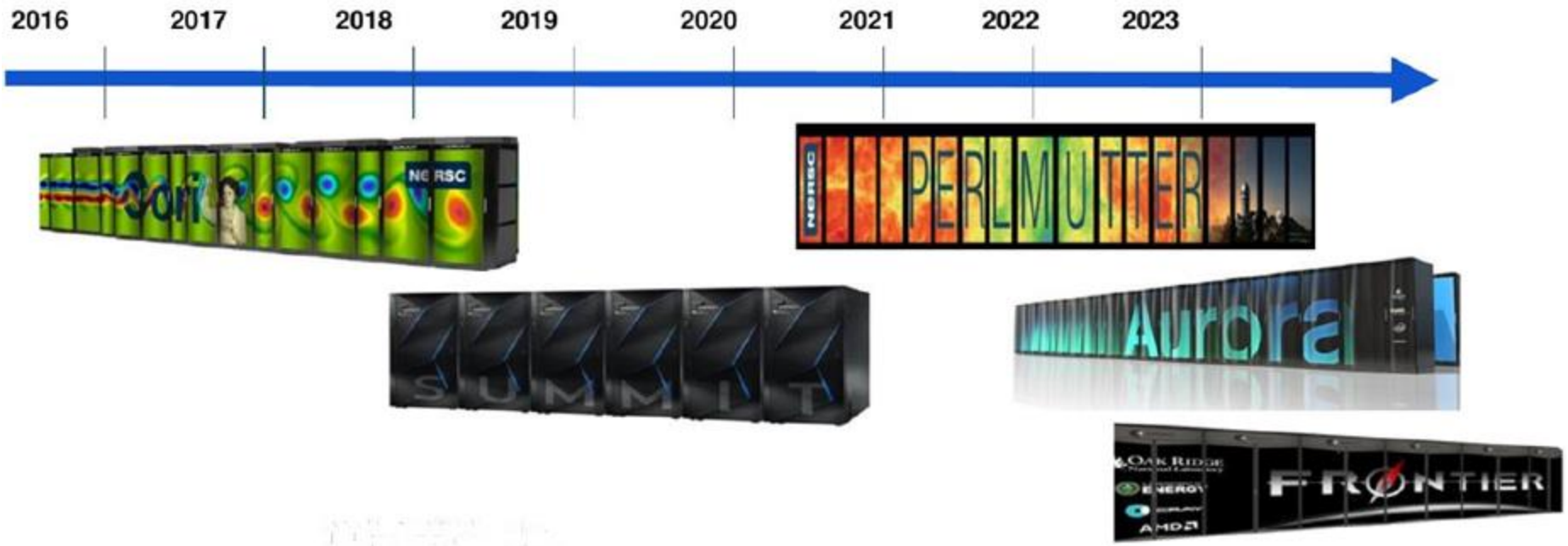
Roughly one order of magnitude improvement in energy per flop efficiency



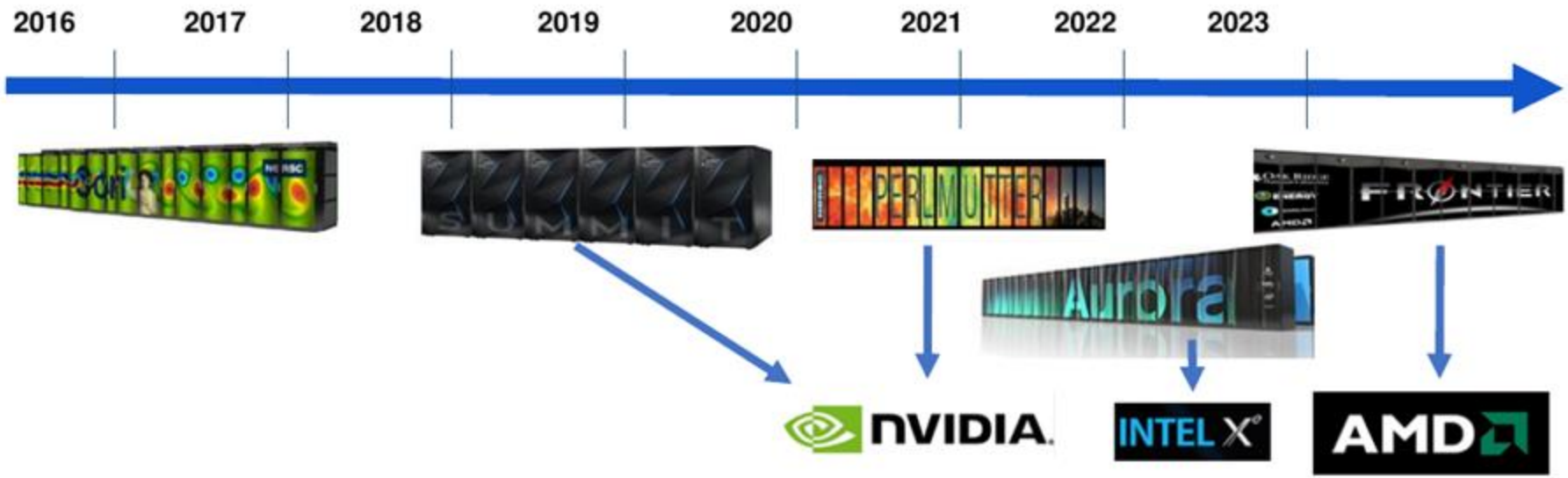
Portability on Exascale Systems

Path to Exascale

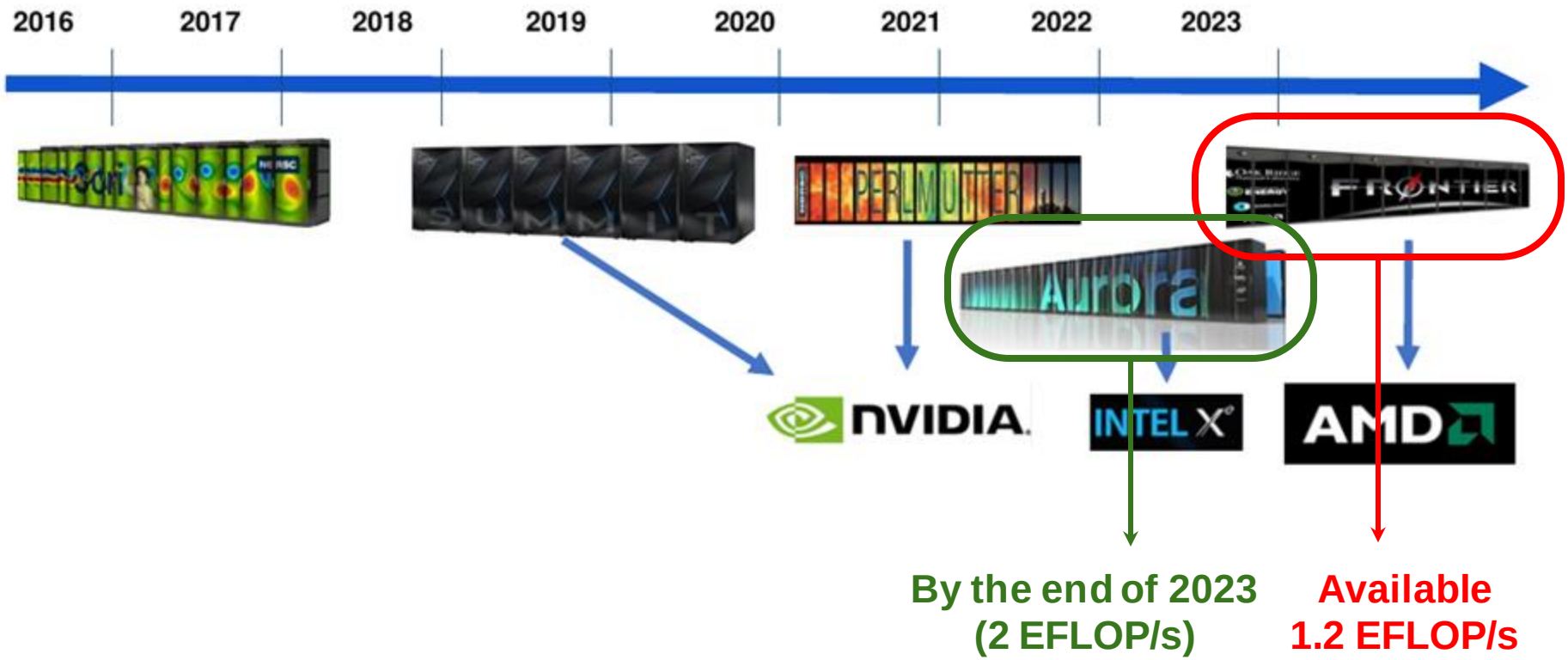
Exascale for DOE Office of Science Means GPU accelerated systems.



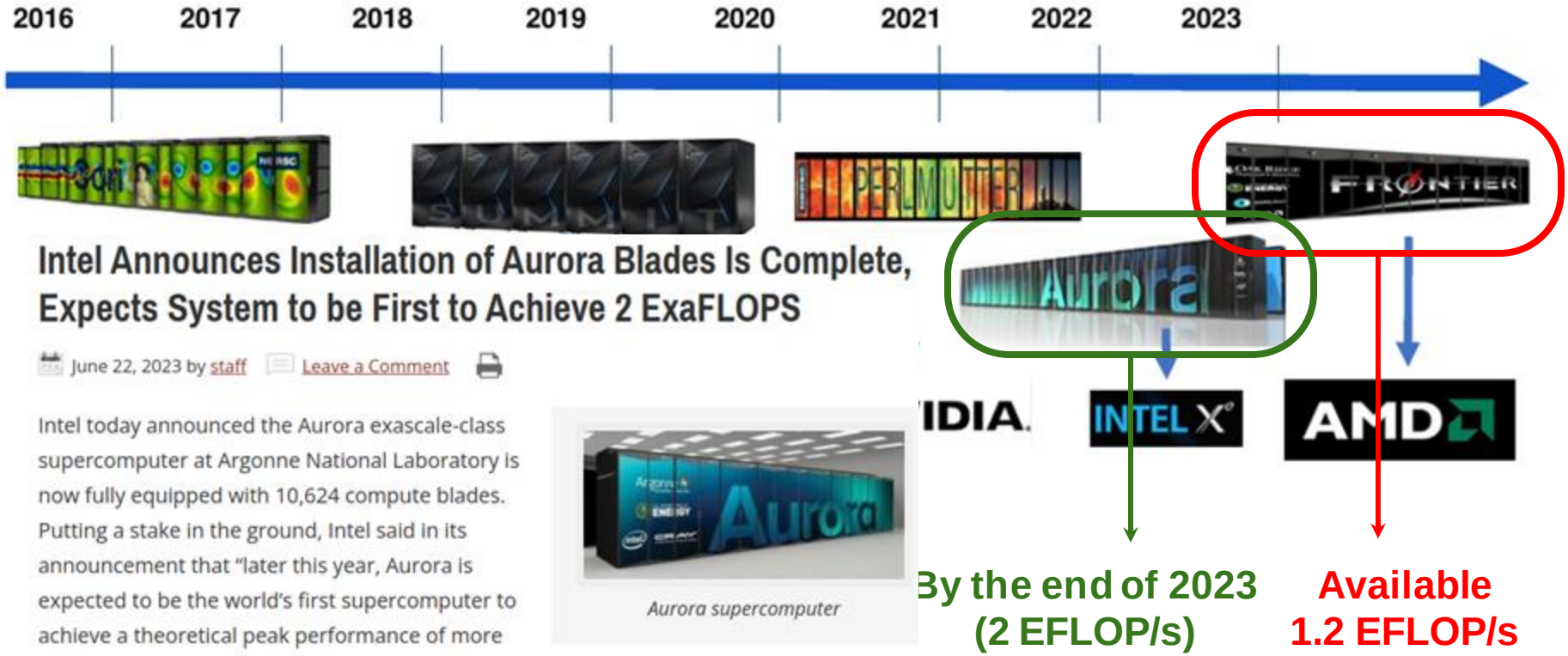
Exascale is Already Here



Exascale is Already Here



Exascale is Already Here



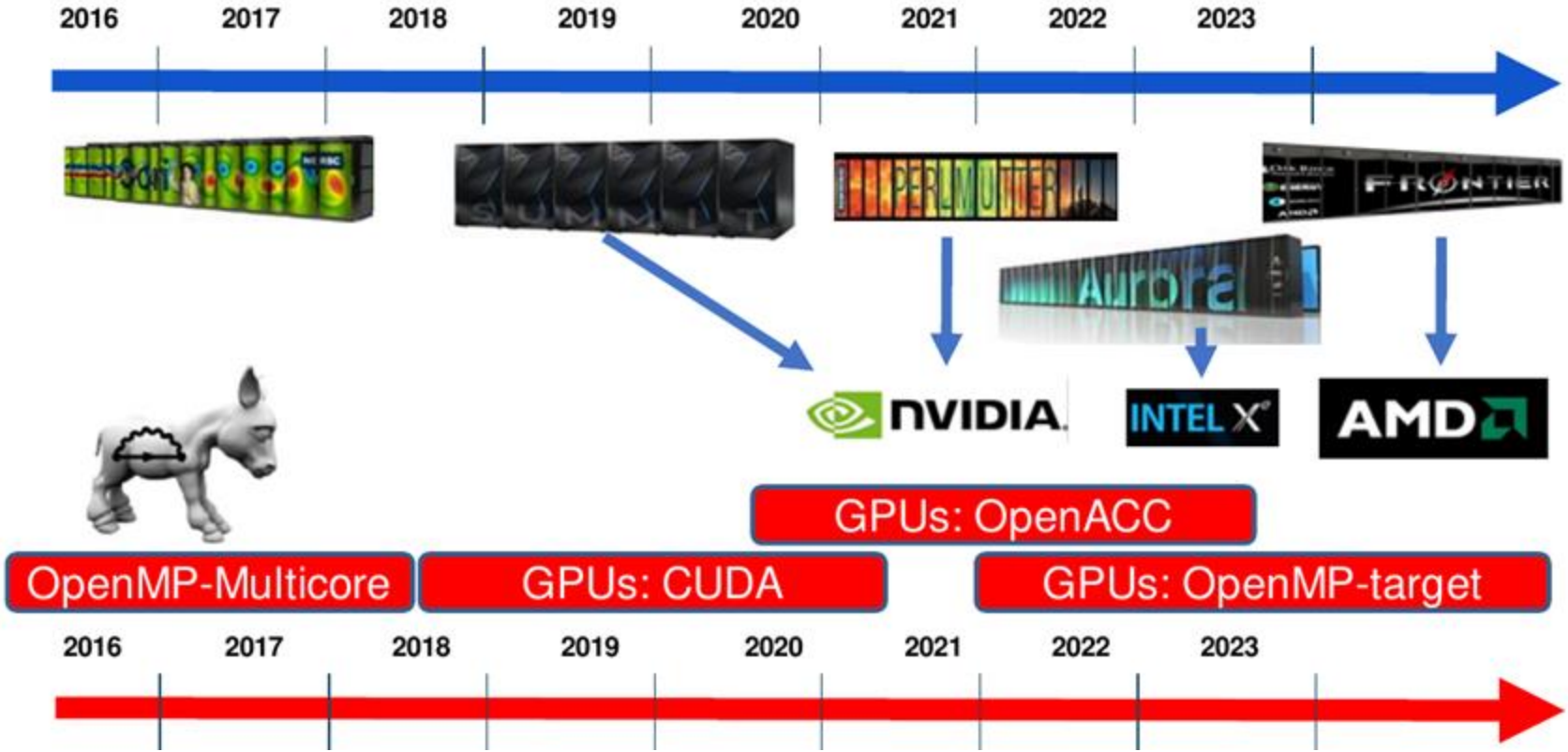
Intel Announces Installation of Aurora Blades Is Complete, Expects System to be First to Achieve 2 ExaFLOPS

June 22, 2023 by staff Leave a Comment

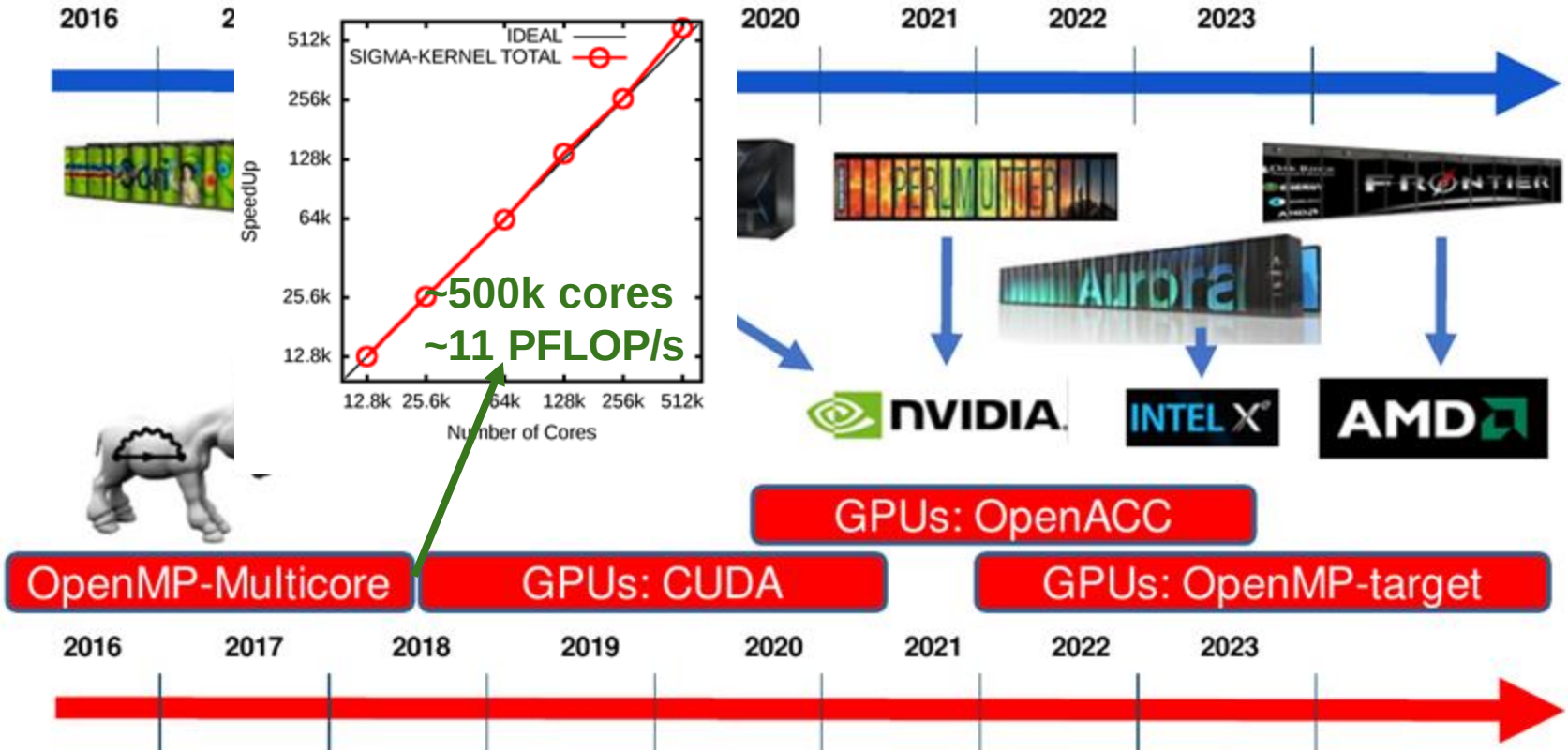
Intel today announced the Aurora exascale-class supercomputer at Argonne National Laboratory is now fully equipped with 10,624 compute blades. Putting a stake in the ground, Intel said in its announcement that "later this year, Aurora is expected to be the world's first supercomputer to achieve a theoretical peak performance of more than 2 exaflops ... when it enters the TOP500 list."

Aurora supercomputer

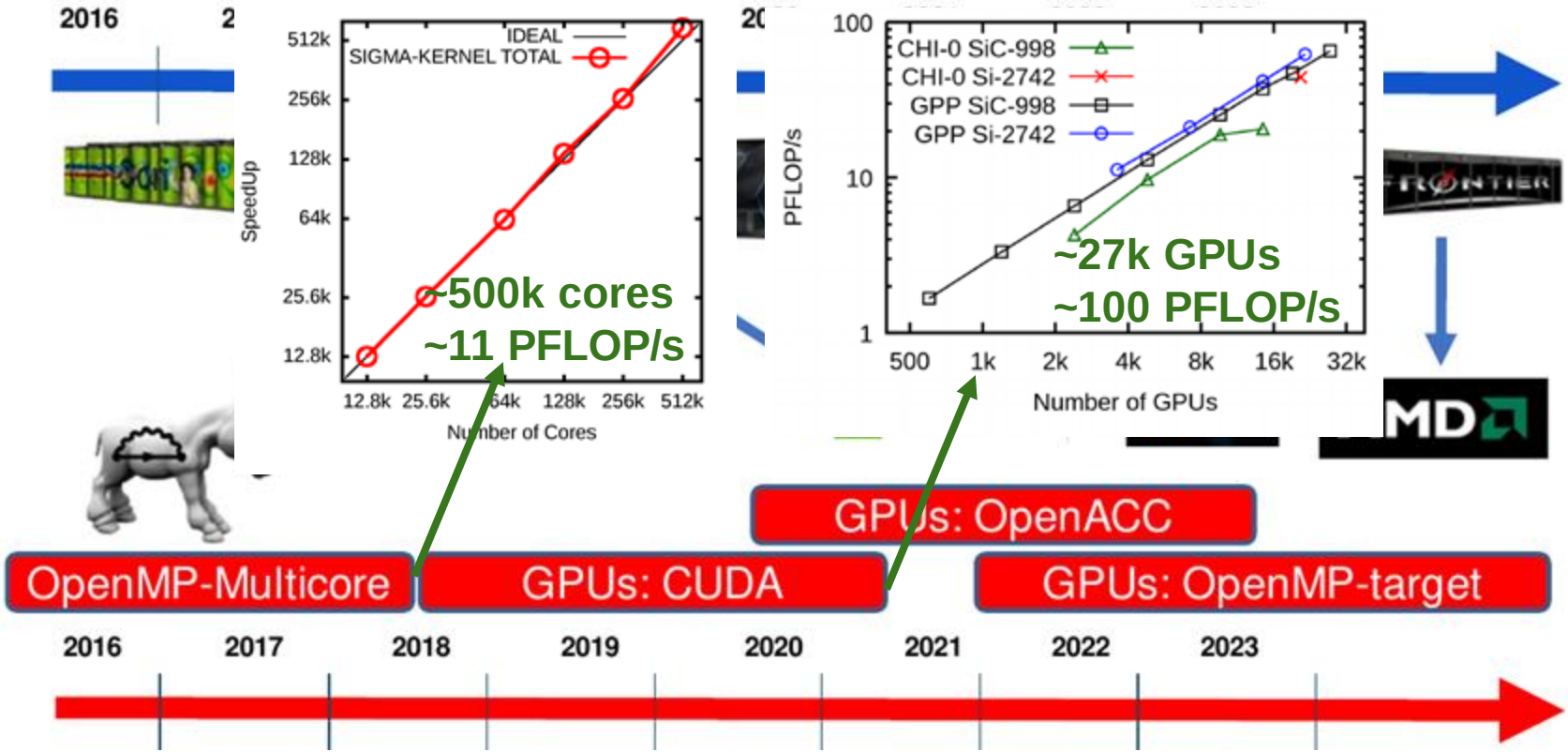
BerkeleyGW on the Path to Exascale



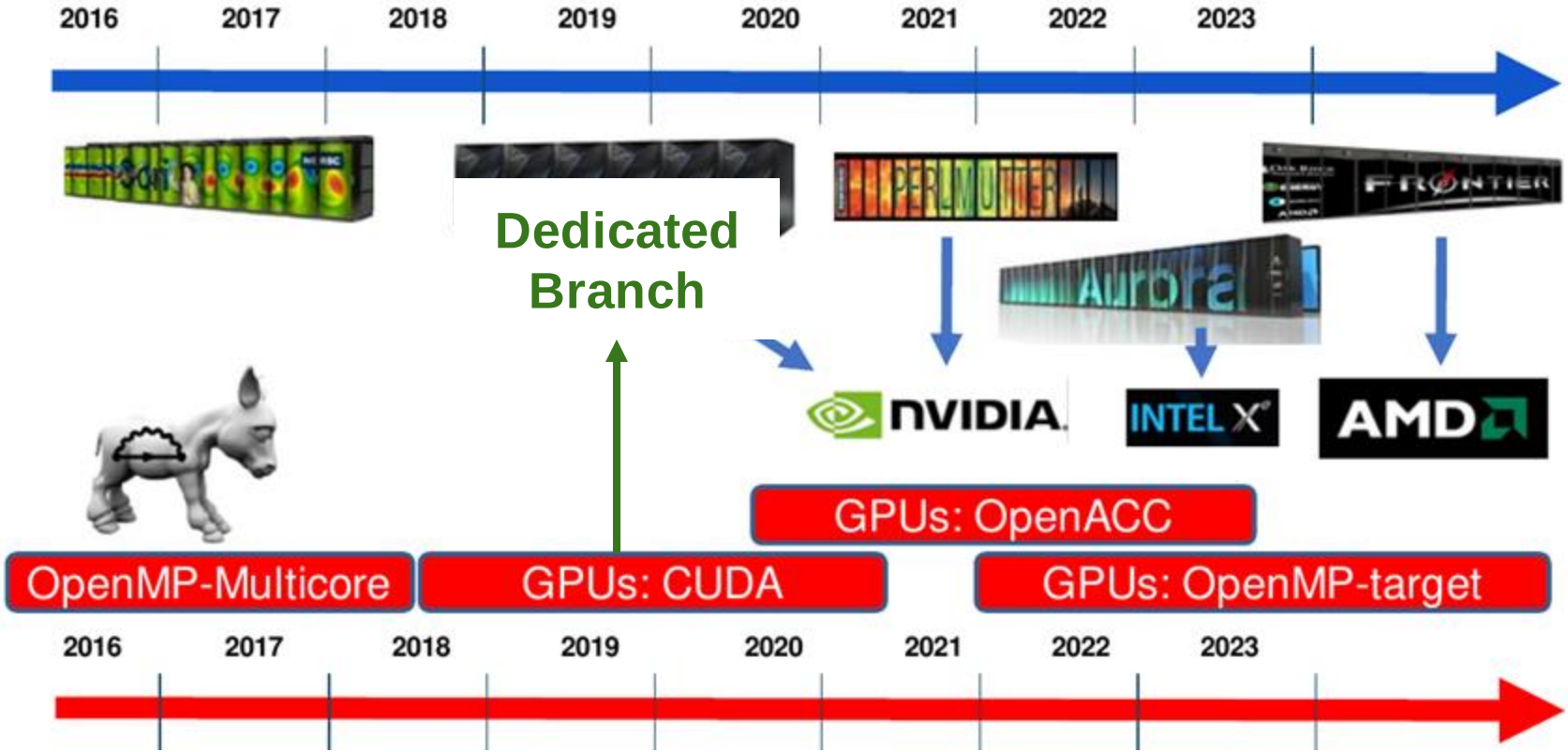
BerkeleyGW on the Path to Exascale



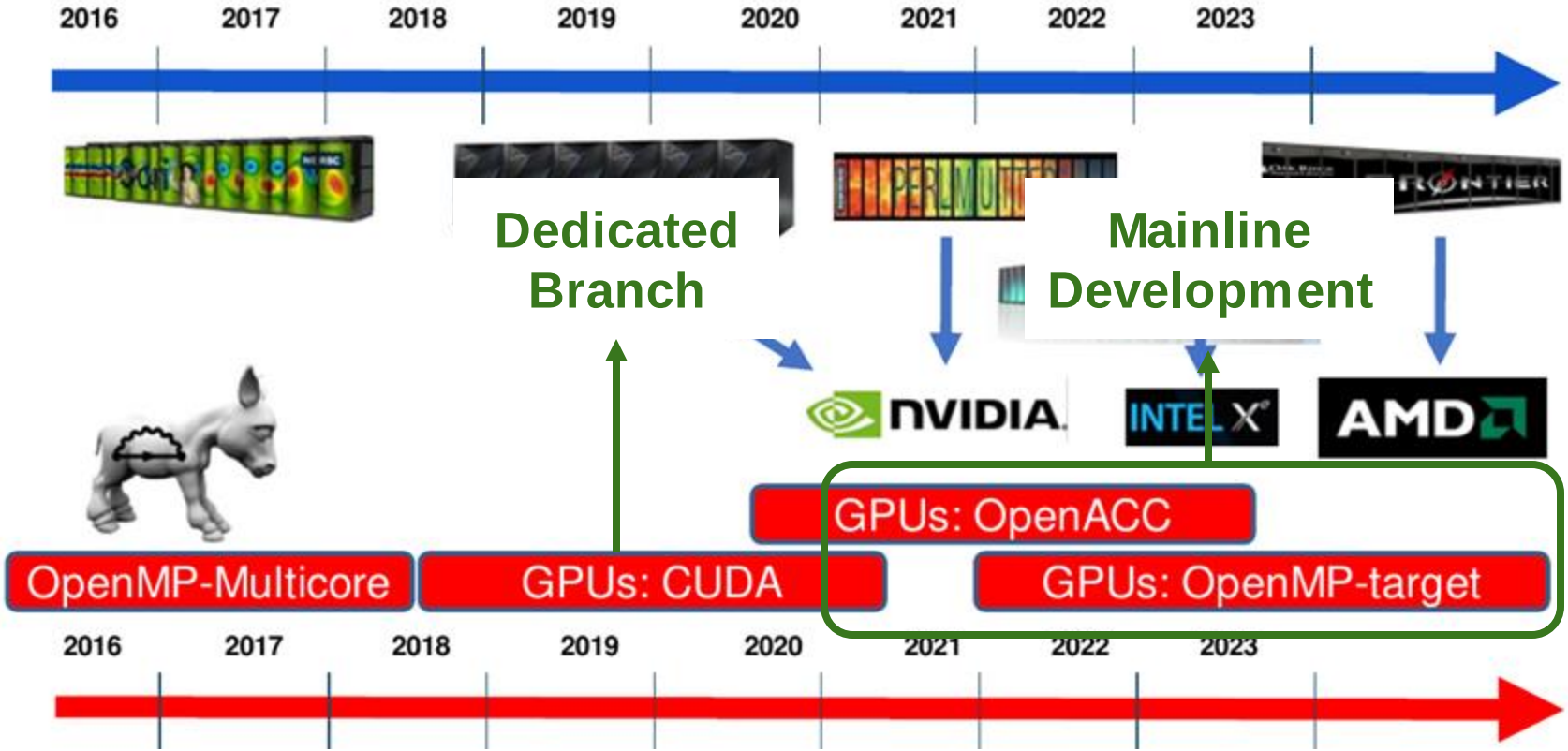
BerkeleyGW on the Path to Exascale



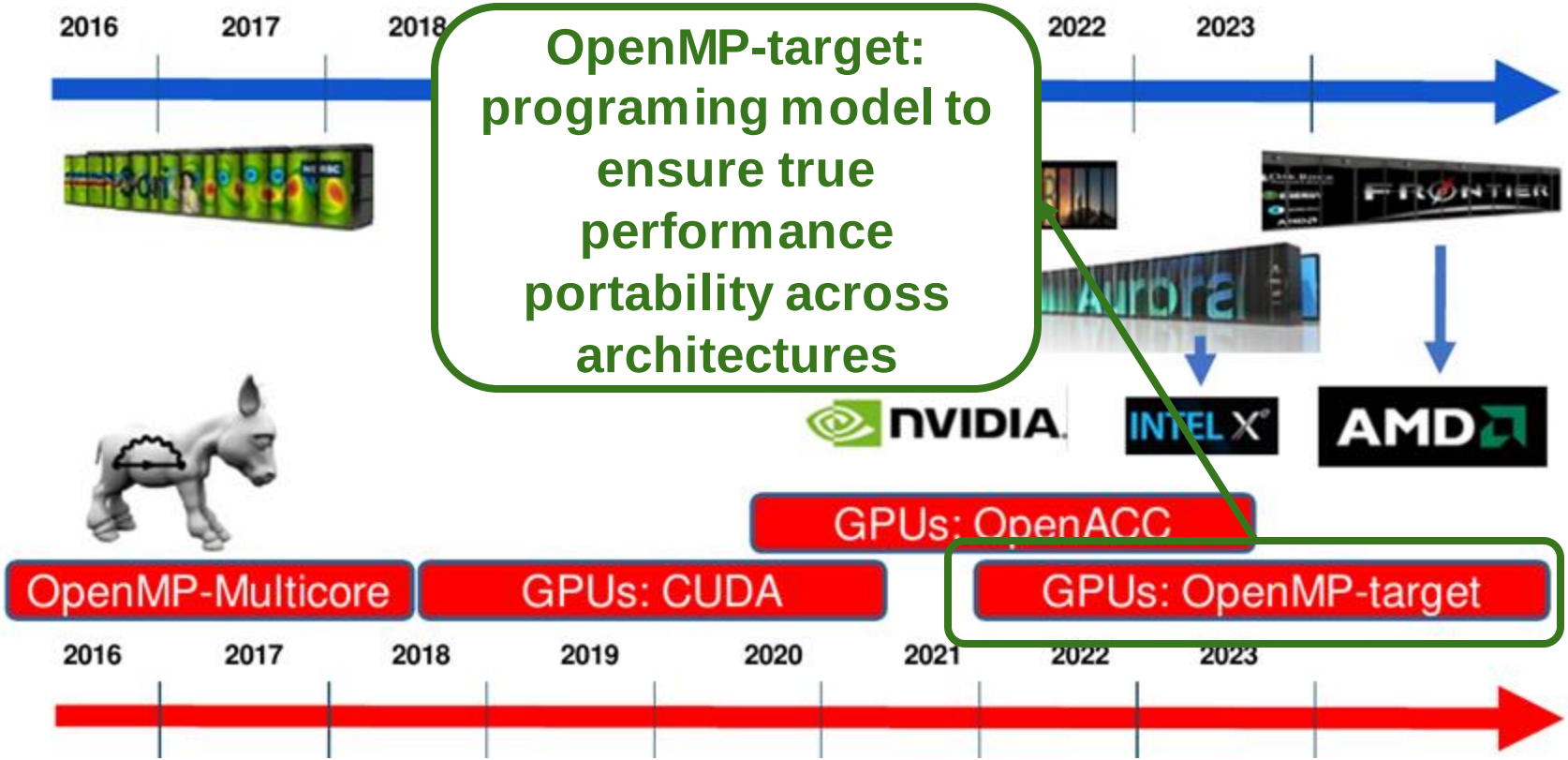
BerkeleyGW on the Path to Exascale



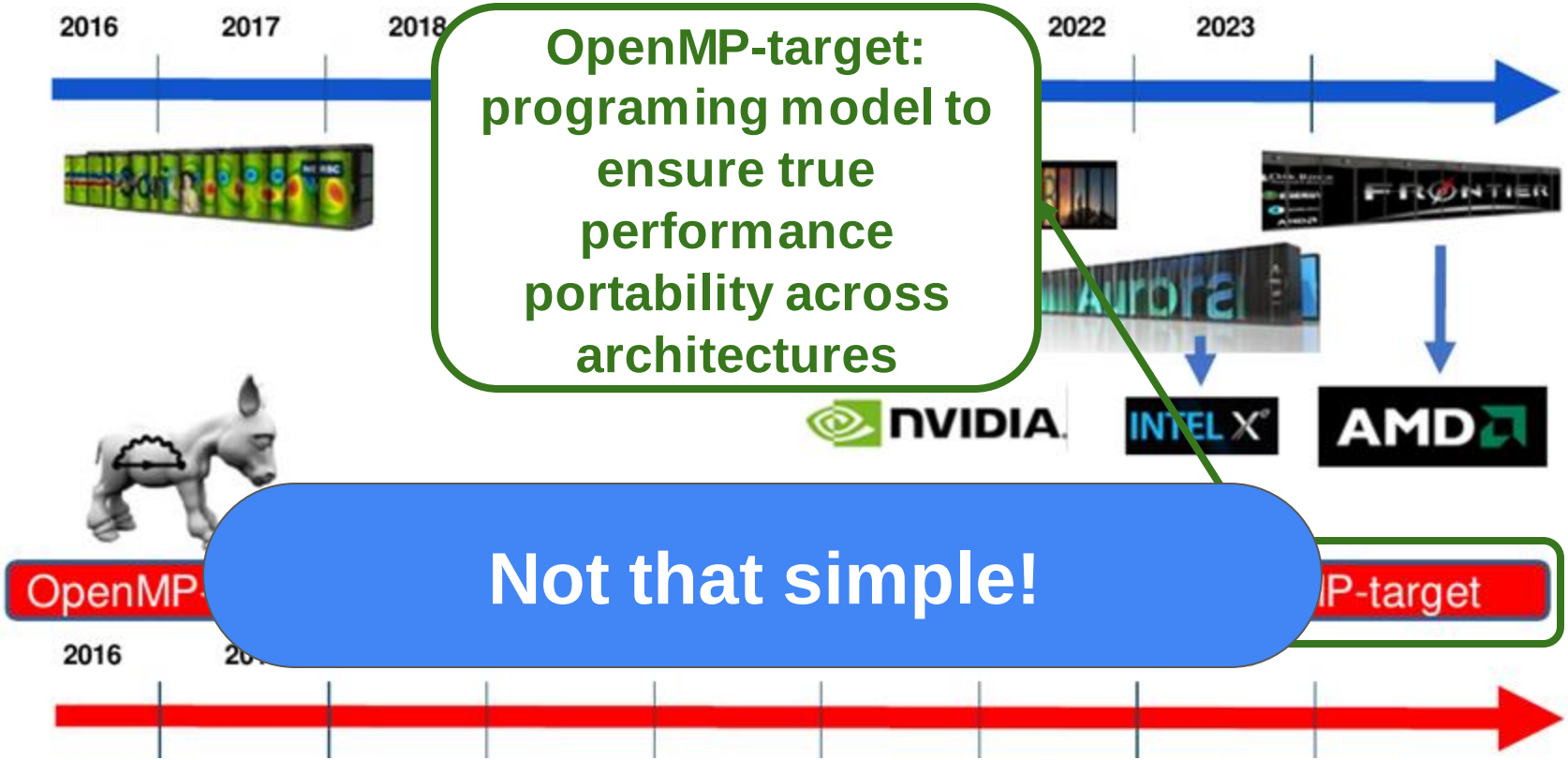
BerkeleyGW on the Path to Exascale



BerkeleyGW on the Path to Exascale



BerkeleyGW on the Path to Exascale



Challenges to achieve true performance portability

- Different hardware have different characteristics
  - May be leveraged by vendor specific programming model (CUDA, HIP, SYCL)
- Portable programming model for offload to device:
  - OpenACC
  - OpenMP-target
- Libraries: non standard/unified Application Programming Interface (API):
  - NVHPC API
  - CUDA API
  - HIP API
  - OneAPI
- Compiler support:
  - Different compilers may behave very differently depending on the choices above
  - Compilers may support or lack various features of the standards
  - Things can easily change and break from one version to another

Portability Status for BerkeleyGW: Compilers/APIs

Offload Programming Model		OpenACC		OpenMP-Target		
Library API		NVHPC	HIP	NVHPC	HIP	OneAPI
Compiler	NVHPC	Yes	Crash	Yes	Crash	
	Cray	NO	Yes	NO	Yes	
	Intel	NO	NO	NO	NO	Functional
	AMD	NO		NO	Crash	
	GNU	NO		NO		

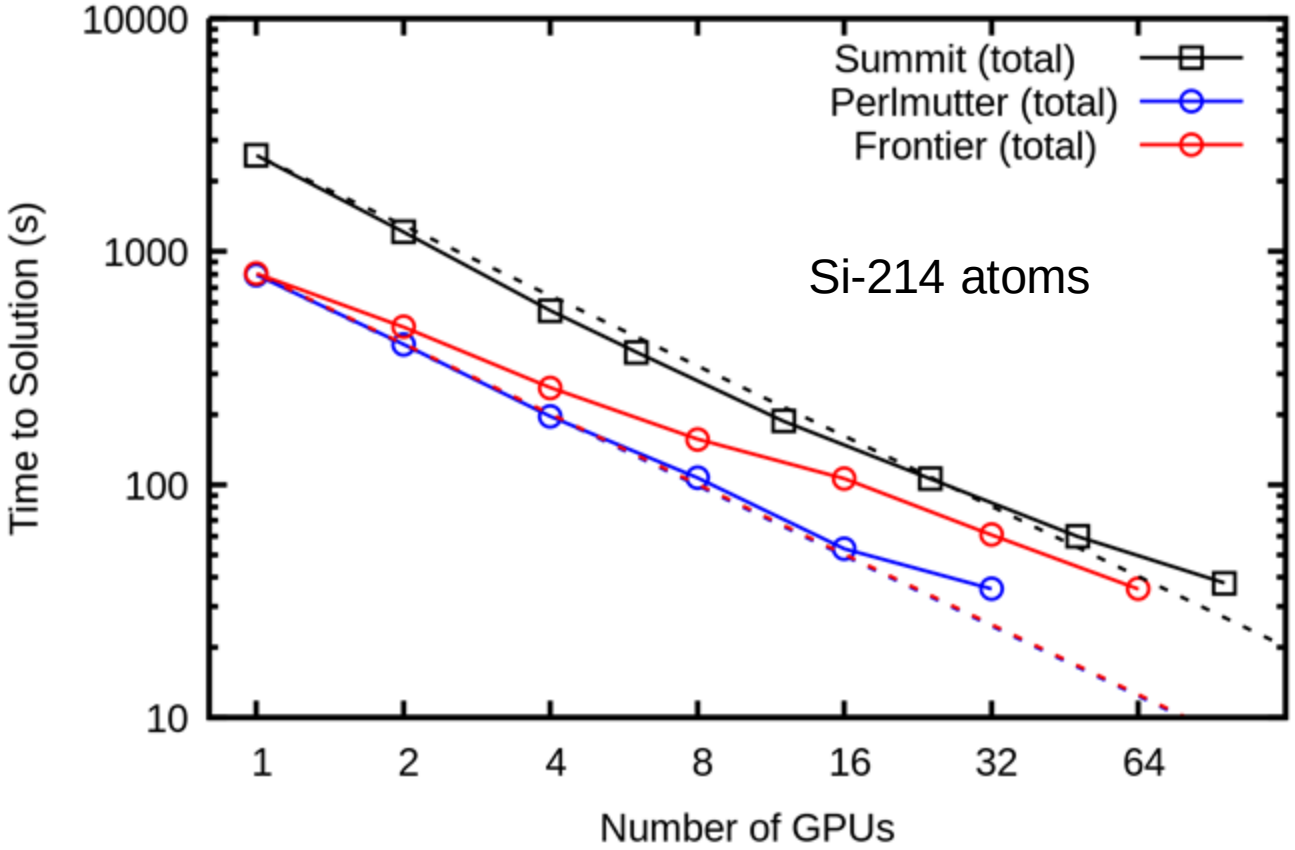
Yes	Production Quality
Functional	Compile and run with some issues at runtime, some kernels do not offload
Crash	Compile but crash, no workarounds
NO	Don't/won't work
	Not tested

Portability Status for BerkeleyGW: Frontier

- Cray compiler, HIP API
- Modules: `cray-fftw; cray-hdf5-parallel; craype-accel-amd-gfx90a; rocm`
- Test: Silicon Carbide, 3x3x3 k-grid (coarse), 5x5x5 k-grid (fine)

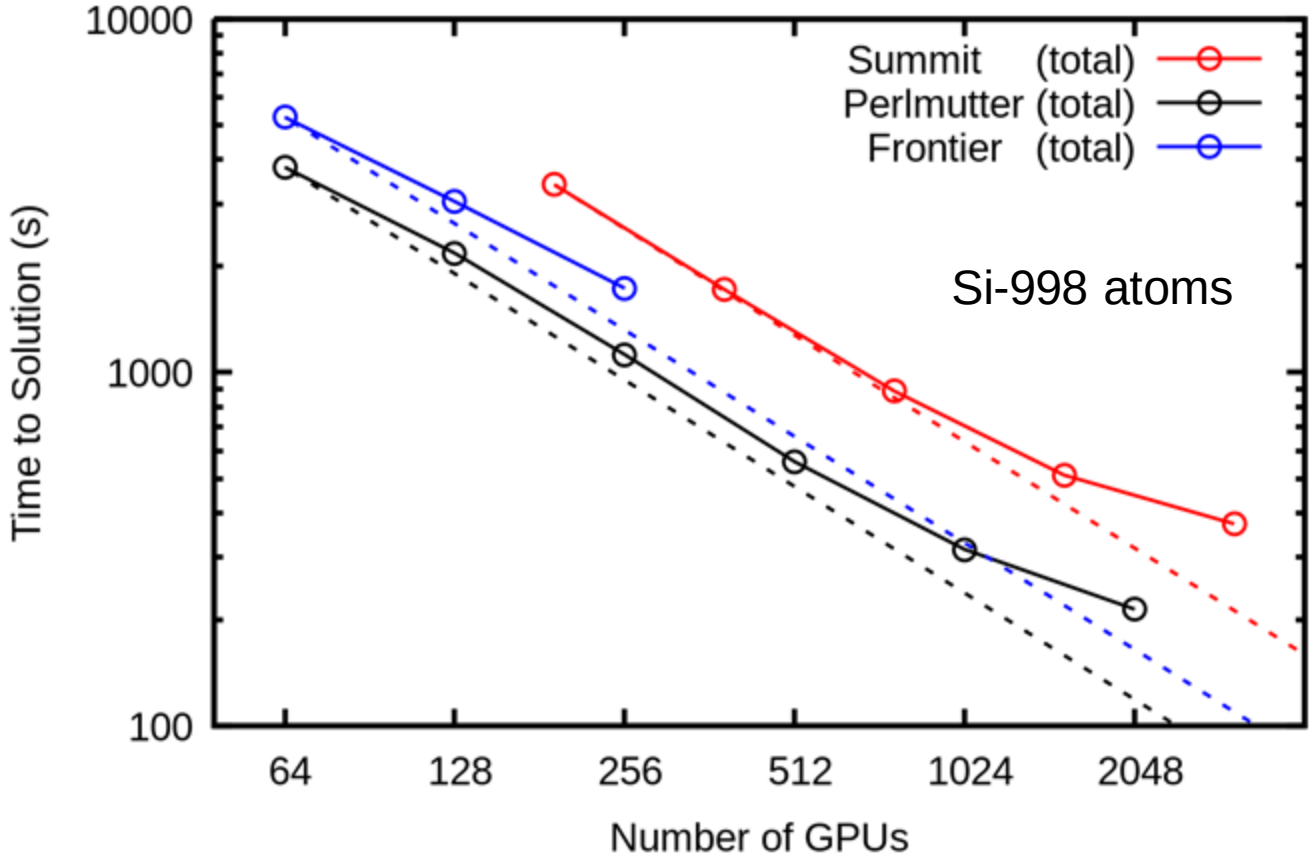
	OpenACC	OpenMP-target
`Epsilon`	☑	☑
`Sigma`	☑	☑
`Kernel`	☑	☑
`Absorption (PRIMME)`	☑	☑

BerkeleyGW on Frontier: Epsilon Strong Scaling

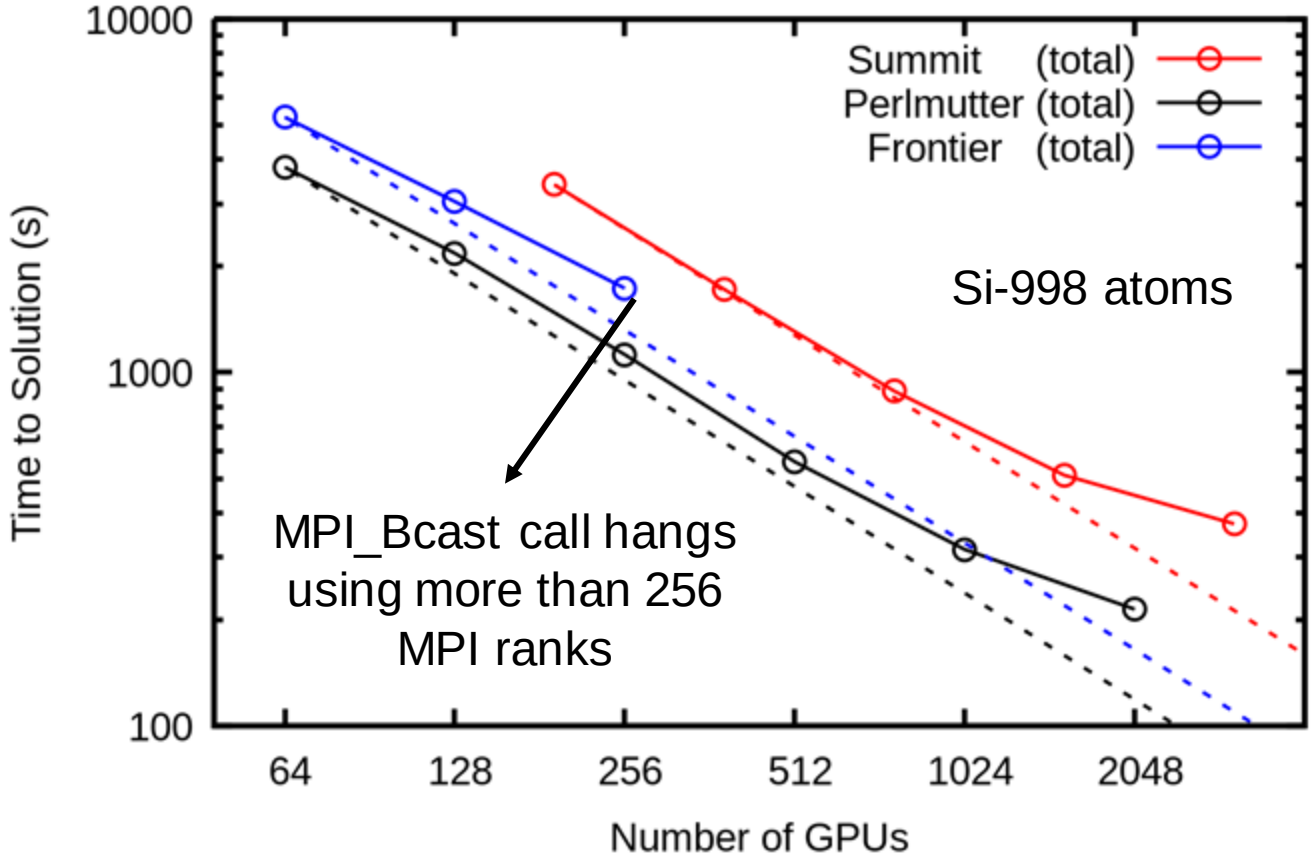


- Summit Node: 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs
- Perlmutter Node: 1 AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs
- Frontier Node: 1 64-core AMD “Optimized 3rd Gen EPYC” CPU + 4 AMD Instinct MI250X GPUs (each feature 2 Graphics Compute Dies for a total of 8 devices)

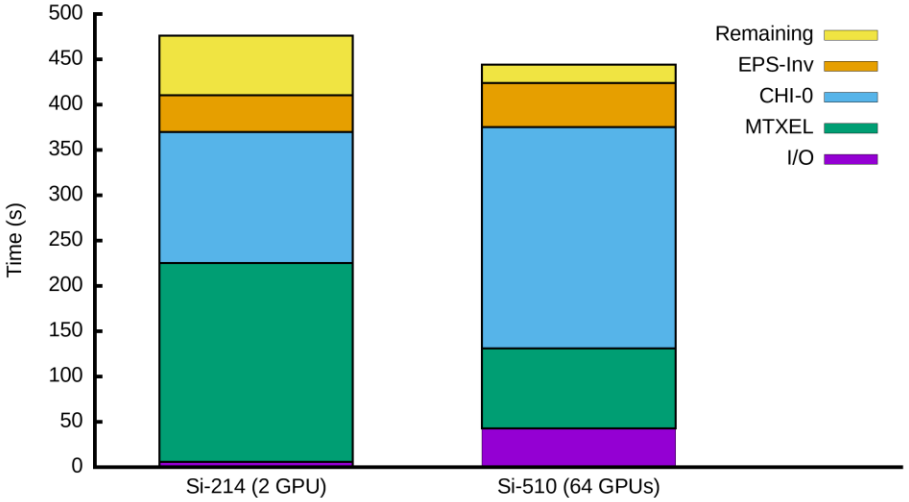
BerkeleyGW on Frontier: Epsilon Strong Scaling

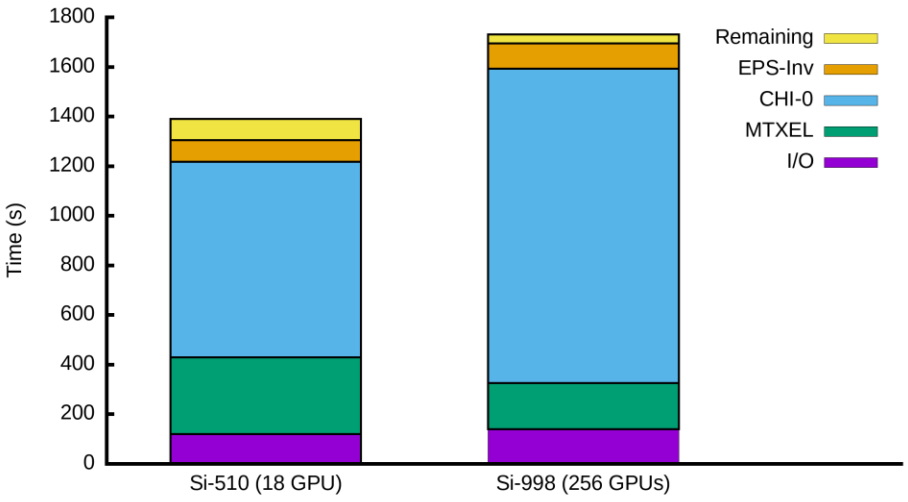


BerkeleyGW on Frontier: Epsilon Strong Scaling

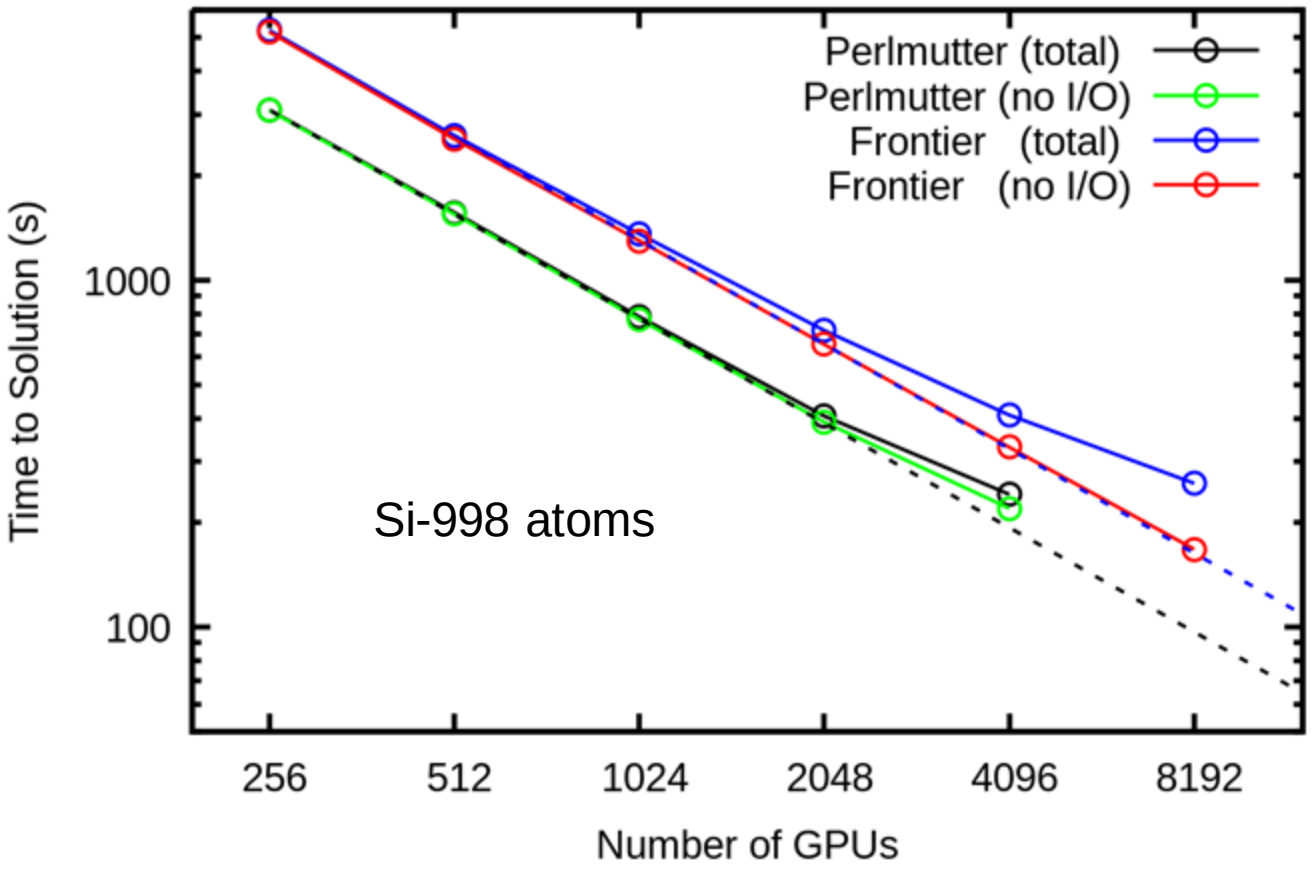


BerkeleyGW on Frontier: Epsilon Weak Scaling

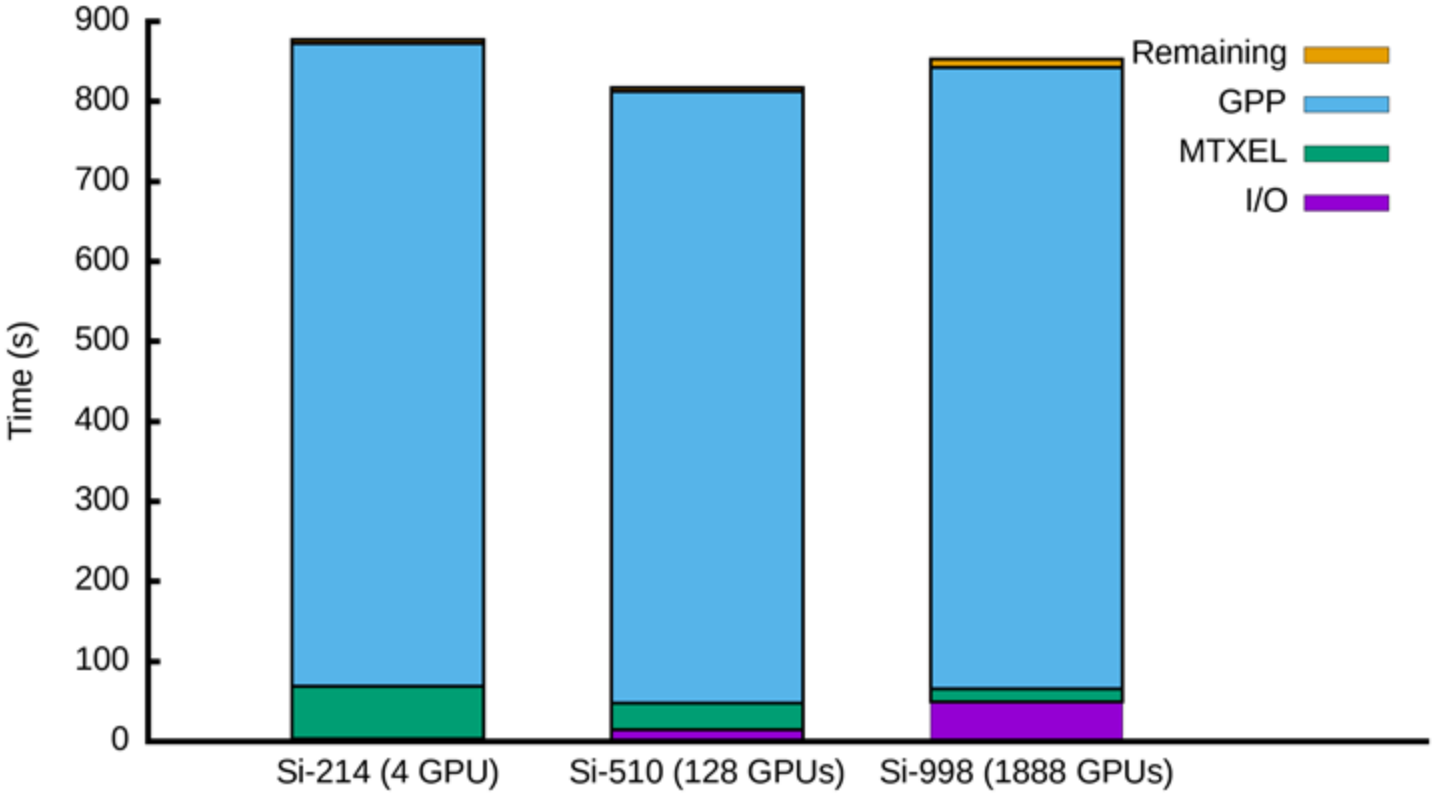




BerkeleyGW on Frontier: Sigma Strong Scaling



BerkeleyGW on Frontier: Sigma Weak Scaling



Portability Status for BerkeleyGW: Sunspot

- Sunspot is the testbed for Aurora
- Intel (ifx) compiler, OneAPI
- Modules:
  oneapi/release/2023.05.15.006; e4s/23.05/2023.05.15.006-2; hdf5/1.14.2-oneapi-mpich mpich/52.2
- Test: Silicon Carbide, 3x3x3 k-grid (coarse), 5x5x5 k-grid (fine)

	OpenMP-target
Epsilon	Functional
Sigma	☑
Kernel	☑
Absorption (PRIMME)	☑

Issues

- Dedicated branch, deep copy of derived types not supported yet
- Performance variability for MPI and I/O operations
- Full Offload kernal CRASH

BerkeleyGW on Sunspot

Aurora MPICH is still under development. The version used here does not represent its expected production state

- Test: Molecular crystal Antraceen (48 atoms), 4x4x4 k-grid (coarse).
- All calculation performed on a single node, total time in seconds.
- Polaris node: 1 64-cores AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs.
- Frontier node: 1 64-core AMD “Optimized 3rd Gen EPYC” CPU + 4 AMD Instinct MI250X GPUs (each feature 2 Graphics Compute Dies for a total of 8 devices).
- Sunspot node: 2 52-physical core Intel Xeon CPU Max Series + 6 Intel Data Center GPUs Max Series (Pontevecchio).

	Epsilon	Sigma
Polaris	105.2	123.8
Frontier	79.1 (49.8)	78
Sunspot	87.6	144.8
Sunspot (no GPUs)	565	1144

BerkeleyGW on Sunspot

Aurora MPICH is still under development. The version used here does not represent its expected production state

- Test: Molecular crystal Antraceen (48 atoms), 4x4x4 k-grid (coarse).
- All calculation performed on a single node, total time in seconds.
- Polaris node: 1 64-cores AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs.
- Frontier node: 1 64-core AMD “Optimized 3rd Gen EPYC” CPU + 4 AMD Instinct MI250X GPUs (each feature 2 Graphics Compute Dies for a total of 8 devices).
- Sunspot node: 2 52-physical core Intel Xeon CPU Max Series + 6 Intel Data Center GPUs Max Series (Pontevecchio).

	Epsilon	Sigma
Polaris	105.2	123.8
Frontier	79.1 (49.8)	78
Sunspot	87.6	144.8
Sunspot (no GPUs)	565	1144

BerkeleyGW on Sunspot

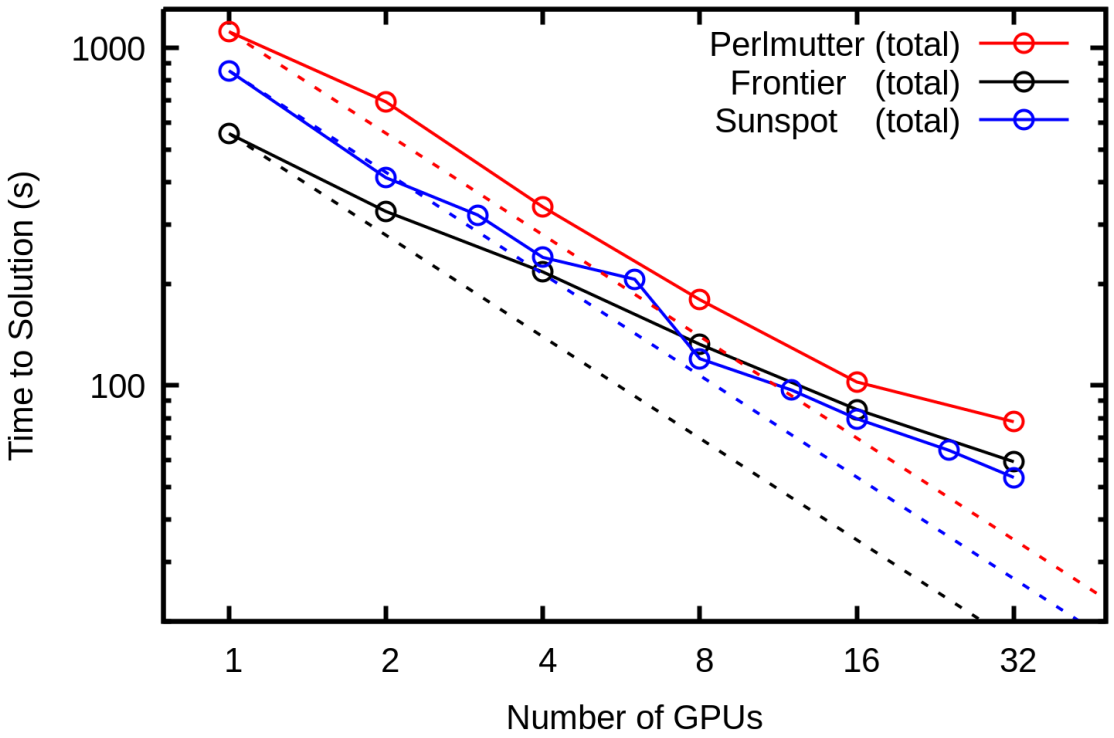
Aurora MPICH is still under development. The version used here does not represent its expected production state

- Test: Molecular crystal Antraceen (48 atoms), 4x4x4 k-grid (coarse).
- All calculation performed on a single node, total time in seconds.
- Polaris node: 1 64-cores AMD EPYC "Milan" CPU + 4 NVIDIA A100 GPUs.
- Frontier node: 1 64-core AMD “Optimized 3rd Gen EPYC” CPU + 4 AMD Instinct MI250X GPUs (each feature 2 Graphics Compute Dies for a total of 8 devices).
- Sunspot node: 2 52-physical core Intel Xeon CPU Max Series + 6 Intel Data Center GPUs Max Series (Pontevecchio).

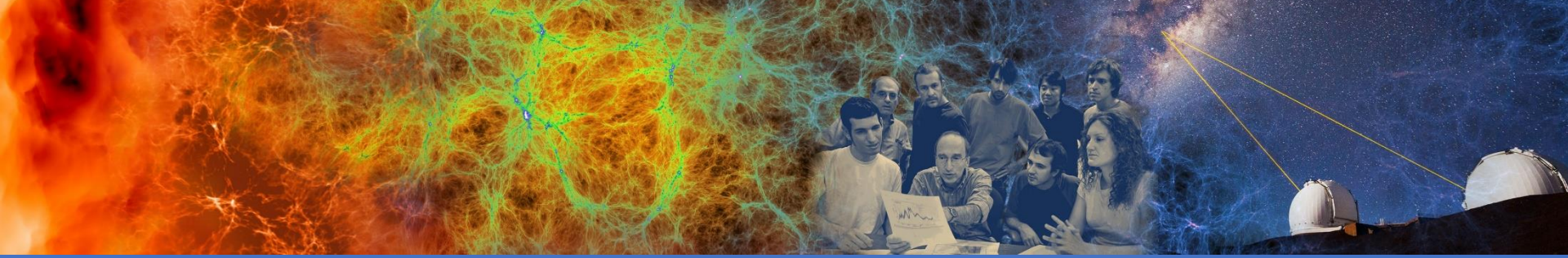
	Epsilon	Sigma
Polaris	105.2	123.8
Frontier	79.1 (49.8)	78
Sunspot	87.6	144.8
Sunspot (no GPUs)	565	1144

Antraceen Strong Scaling

Aurora MPICH is still under development. The version used here does not represent its expected production state



Submat offload algorithm (fullofload crash on PVC), each GPU is consider a A100, MI250x (2 devices) and PVC (2 tiles)

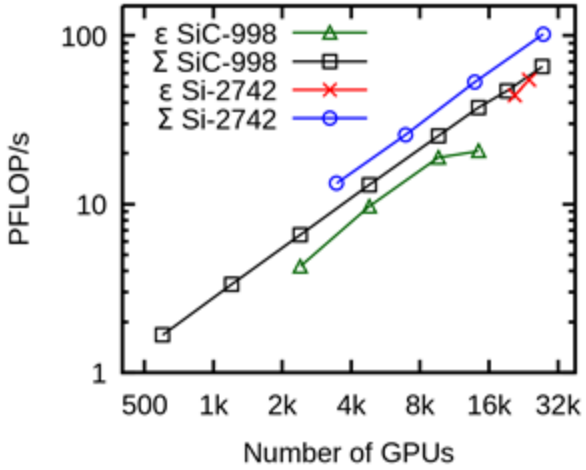


Summary

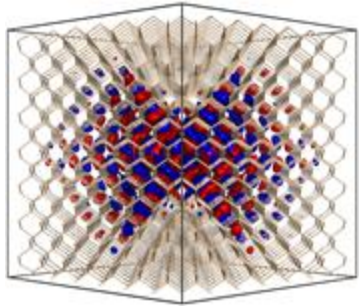
BerkeleyGW On the Path to Exascale

Optimized version of BerkeleyGW on GPU accelerated systems:

- Scale up to the full Summit machine at OLCF: **>27k GPUs**
- Reach nearly 53% of the peak performance at **106 PFLOP/s**
- Time to solution of **~10 mins for 11k electrons** system



M. Del Ben, C. Yang, Z. Li, F. H. da Jornada, S. G. Louie and J. Deslippe, "Accelerating Large-Scale Excited-State GW Calculations on Leadership Class HPC Systems" in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '20 No.4 pp.1 (2020), ACM Gordon-Bell Finalist

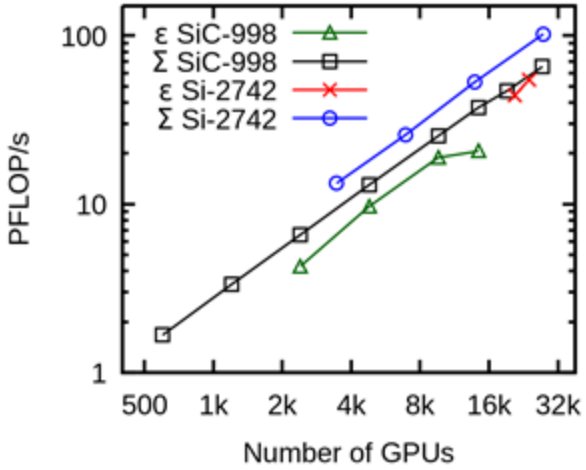


Divacancy defect in semiconductor (such as Si and SiC) are proxy for solid state Qubits. For silicon shown is the 2742-atoms Si supercell, 10,968 electrons.

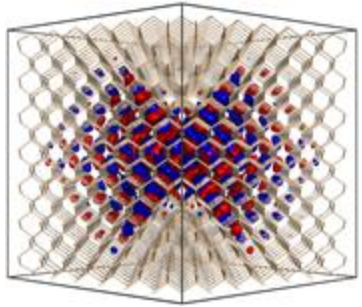
BerkeleyGW On the Path to Exascale

Optimized version of BerkeleyGW on GPU accelerated systems:

- Scale up to the full Summit machine at OLCF: **>27k GPUs**
- Reach nearly 53% of the peak performance at **106 PFLOP/s**
- Time to solution of **~10 mins for 11k electrons** system



M. Del Ben, C. Yang, Z. Li, F. H. da Jornada, S. G. Louie and J. Deslippe, "Accelerating Large-Scale Excited-State GW Calculations on Leadership Class HPC Systems" in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '20 No.4 pp.1 (2020), ACM Gordon-Bell Finalist



Divacancy defect in semiconductor (such as Si and SiC) are proxy for solid state Qubits. For silicon shown is the 2742-atoms Si supercell, 10,968 electrons.

OpenMP-target seems a valuable option to port Fortran codes to different vendor architectures!

Funding Sources

- This research used resources at the National Energy Research Scientific Computing Center (NERSC), which is supported by the U.S. Department of Energy Office of Science under contract DE-AC02-05CH11231.

- This research used resources at the Oak Ridge Leadership Computing Facility (OLCF) through the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program, which is supported by the U.S. Department of Energy Office of Science under Contract No. DE-AC05-00OR22725.

- This work was supported by the Center for Computational Study of Excited-State Phenomena in Energy Materials (C2SEPEM), funded by the U.S. Department of Energy Office of Science under Contract No. DEAC02-05CH11231.