Hardware aware matrix-free approach for accelerating FE discretized eigenvalue problems: Application to large-scale Kohn-Sham density functional theory

Gourab Panigrahi

Doctoral Advisor: Dr. Phani Motamarri

Department of Computational and Data Sciences (CDS)
Indian Institute of Science (IISc) Bangalore

Partial Differential Equations (PDE)

- PDEs commonly encountered in Navier-Stokes, Quantum modeling of materials (DFT, DFPT), Electrostatics, Modal analysis, etc

Partial Differential Equations :

$$\mathcal{F}u = f$$

$\mathcal{F}$: PDE operator
- $-\nabla^2 \rightarrow$ Poisson Equation
- $-\nabla^2 + \kappa \rightarrow$ Helmholtz Equation
- $-\nabla^2 + V(x) \rightarrow$ Quantum Mechanics

u : Scalar Field

f : Forcing function

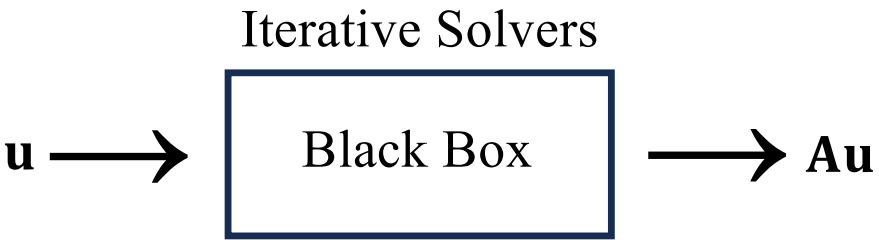
Finite-element (FE) Discretization:

$$\mathbf{Au = f} \qquad \mathbf{A}: N_{DoF} \times N_{DoF} \qquad \mathbf{f}: N_{DoF}$$

❖ $N_{DoF} \rightarrow$ No of grid points/degrees of freedom (~10^6 - 10^9)

❖ *Compact support of FE polynomial basis* → **A** is sparse (~99%)

❖ *Large-scale sparse matrix problems*: Iterative solvers are employed



Computationally dominant step

Sparse matrix multi-vector products (AU)

$$\mathcal{F}u^i(\mathbf{x}) = \begin{cases} f^i(\mathbf{x}) \\ \lambda^i u^i(\mathbf{x}) \end{cases} \quad \forall \mathbf{x} \in \Omega \text{ such that } \quad u^i(\mathbf{x}) = u_D(\mathbf{x}) \quad \forall \mathbf{x} \in \partial\Omega_D$$

$$u^i(\mathbf{x}) \approx u^{i,h}(\mathbf{x}) = \sum_J u_J^i N_J(\mathbf{x})$$

$$\mathbf{AU} = \begin{cases} \mathbf{F} \\ \mathbf{MU\Lambda} \end{cases} \text{ such that } \quad U_{ki} = u_D(\mathbf{x}_k) \quad \forall \mathbf{x}_k \in \partial\Omega_D$$

where multi-vectors $\mathbf{U} = \begin{bmatrix} \boldsymbol{u}^1 & \boldsymbol{u}^2 & \dots & \boldsymbol{u}^{n_v} \end{bmatrix}$ and $\mathbf{F} = \begin{bmatrix} \boldsymbol{f}^1 & \boldsymbol{f}^2 & \dots & \boldsymbol{f}^{n_v} \end{bmatrix}$ $\mathbf{\Lambda}$ = Diagonal Matrix of Eigenvalues

- Kohn-Sham Equation (Density Functional Theory) :

$$\left(-\frac{1}{2}\nabla^2 + V_{\text{eff}}[\rho, \boldsymbol{R}]\right)\psi_i = \epsilon_i \psi_i, \quad \rho = \sum_i |\psi_i|^2, \quad i = 1, 2, \dots N$$

- Challenging PDE : Non-linear Eigenvalue Problem and $N \sim 1 - 10^5$ eigenvalues/eigenvectors pairs depending on number of electrons

[1] J. Sun, A. Zhou, Finite Element Methods for Eigenvalue Problems, Chapman and Hall/CRC, 2016.
[2] K. Ghosh, H. Ma, V. Gavini, G. Galli, All-electron density functional calculations for electron and nuclear spin interactions in molecules and solids, Physical Review Materials 3 (2019)
[3] S. Markidis, The Old and the New : Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?, Frontiers in Big Data 4 (2021)

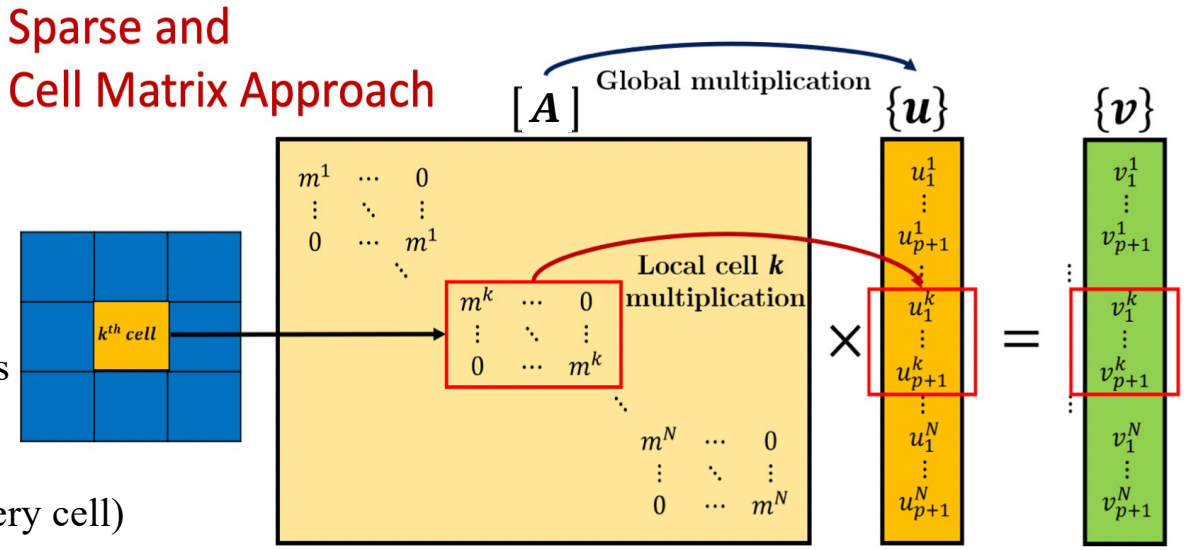
Sparse matrix-vector products (Au)

- **Global sparse-matrix Approach** [1] ⟶ Global Sparse Matrix is assembled and multiplied with global vector.
  (Uses Sparse Matrix modules in popular FEM libraries like PETSc) (Efficient for FE polynomial degrees 1 and 2)

- **Cell-matrix Approach** [2, 3] :
  - FE cell-level matrix construction
  - FE cell-level vector extraction
  - FE cell-level matrix vector product
  - Assembly of cell-level product vectors

- **Advantages of cell-matrix over sparse matrix**
  - Local dense matrix-vector product at cell-level
  - Amenable to thread parallelism on CPUs/GPUs

- **Demerits of cell-matrix approach**
  - Storage and data access cost of cell-matrix (every cell)
  - Storage of cell-level vectors (***u*** and ***v***)



[1] C.D. Cantwell, S.J. Sherwin, R.M. Kirby, P.H.J. Kelly, From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements, Computers & Fluids 43 (2011) 23–28

[2] G. F. Carey, E. Barragy, R. McLay, M. Sharma, Element-by-element vector and parallel computations, Communications in Applied Numerical Methods 4 (1988) 299–307.

[3] S. Das, P. Motamarri, V. Subramanian, D. M. Rogers, V. Gavini, DFT- FE 1.0: A massively parallel hybrid CPU-GPU density functional theory code using finite-element discretization, Computer Physics Communications 280 (2022) 108473.

Strategies to compute sparse matrix multi-vector products (AU)

- Blocked Iterative Solvers : CG, GMRES, MINRES (Linear System of Equations)
  : Arnoldi, Davidson, Chebyshev Filtered Subspace Iteration (ChFSI) (Eigenvalue Problems)

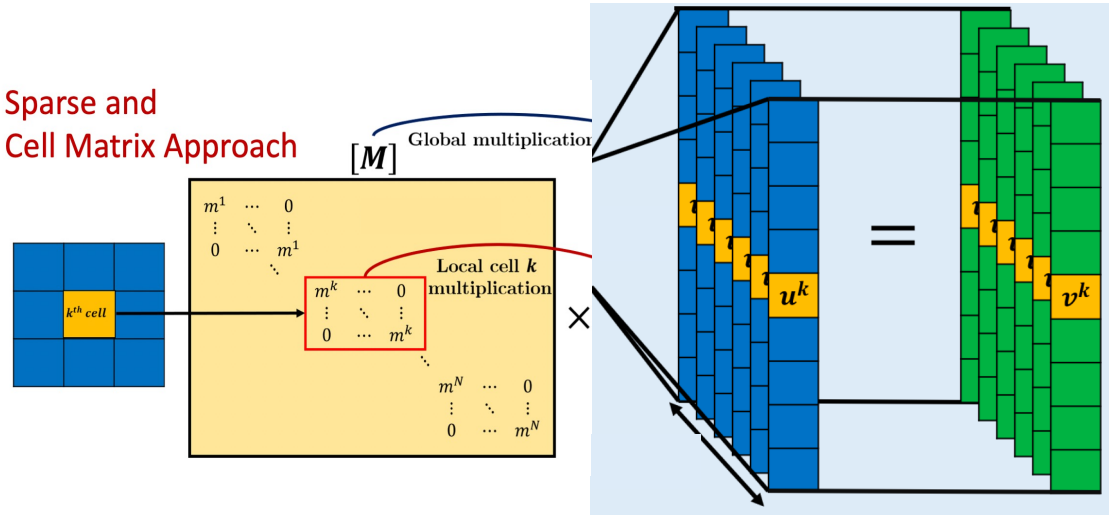
- Computationally dominant step → **AU**

- Cell-matrix approach is shown to be computationally efficient with good throughput performance than sparse matrix approach for evaluating **AU**

Cell-matrix approach in ChFSI for a DFT calculation **[10]**
(Non-linear eigenvalue problem for 15k eigenpairs)

Evaluation of AU

- $\sim 4 \times 10^{-3}$ node-hrs per **AU** on V100 GPUs of Summit
- Performed $\sim 10^5$ times!
- $\sim$ 58% of the total eigenvalue solve!
- $\sim$ 100 GB device memory on GPUs

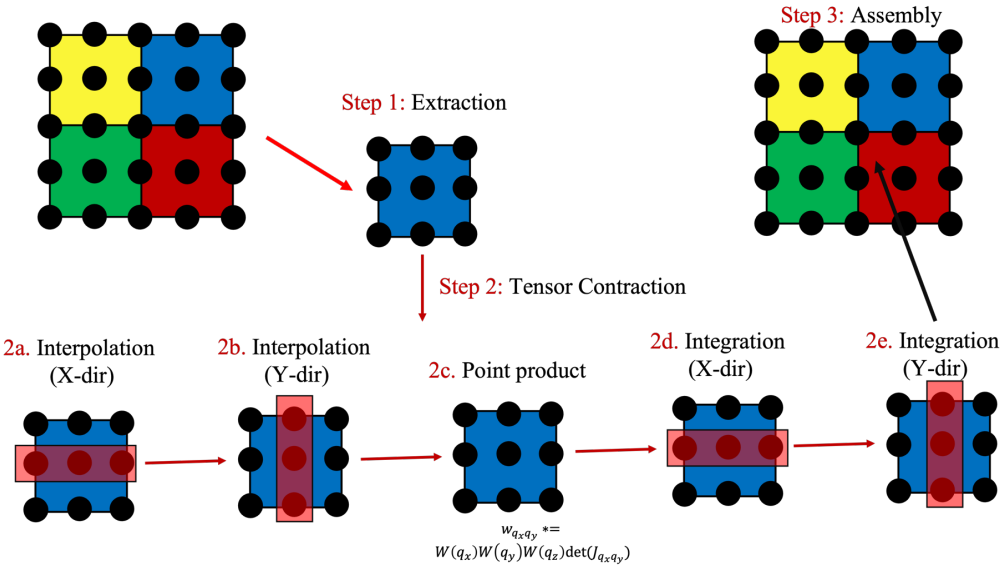


[10] S. Das, P. Motamarri, V. Subramanian, D. M. Rogers, V. Gavini, DFT- FE 1.0: A massively parallel hybrid CPU-GPU density functional theory code using finite-element discretization, Computer Physics Communications 280 (2022) 108473.

Matrix-free approach for matrix-vector products

- On-the-fly matrix-vector products without storing the cell-level dense matrices → reduce both arithmetic complexity and memory footprint [4 - 6].

- **Matrix-free** approaches, exploit the tensor-structured nature of the finite-element basis functions $N_I(\mathbf{x})$ and recast the 3D integrals involved in the matrix-vector products as a sequence of tensor contractions.



$$\boldsymbol{v} = [\mathbf{M}]\{\boldsymbol{u}\} \rightarrow v_I = \sum_J M_{IJ} u_J = \int_\Omega N_I(\mathbf{x}) N_J(\mathbf{x}) u_J d\mathbf{x}$$

$$= \int_{\hat{\Omega}} \hat{N}_I^{3D}(\boldsymbol{\xi}) \hat{N}_J^{3D}(\boldsymbol{\xi}) u_J det(\mathbf{J}) d\boldsymbol{\xi} = \int_{\hat{\Omega}} \hat{N}_{i_x}(\xi_1) \hat{N}_{i_y}(\xi_2) \hat{N}_{i_z}(\xi_3) \hat{N}_{j_x}(\xi_1) \hat{N}_{j_y}(\xi_2) \hat{N}_{j_z}(\xi_3) u_{j_x j_y j_z} det(\mathbf{J}) d\xi_1 d\xi_2 d\xi_3$$

Matrix free Algorithm

$O(n_p^4)$, n_p = No of grid points in each direction

$$v_I = v_{i_x i_y i_z} = \sum_{q_z} w_{q_z} . \hat{N}_{i_z}(\xi_{q_z}) \sum_{q_y} w_{q_y} . \hat{N}_{i_y}(\xi_{q_y}) \sum_{q_x} w_{q_x} . \hat{N}_{i_x}(\xi_{q_x}) \left| J_{q_x q_y q_z} \right| \sum_{j_z} \hat{N}_{j_z}(\xi_{q_z}) \sum_{j_y} \hat{N}_{j_y}(\xi_{q_y}) \sum_{j_x} \hat{N}_{j_x}(\xi_{q_x}) u_{j_x j_y j_z}$$

[4] K. Ljungkvist, Matrix-Free Finite-Element Computations on Graphics Processors with Adaptively Refined Unstructured Meshes, in: Proceedings of the 25th High Performance Computing Symposium, HPC '17, Society for Computer Simulation International, San Diego, CA, USA, 2017, pp. 1–12.
[5] M. Kronbichler, K. Kormann, A generic interface for parallel cell-based finite element operator application, Computers & Fluids 63 (2012) 135–147.
[6] D. Davydov, J. Pelteret, D. Arndt, M. Kronbichler, P. Steinmann, A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid, International Journal for Numerical Methods in Engineering 121 (2020) 2874–2895.

Strategies to compute sparse matrix multi-vector products (AU)

- Cell-matrix → Dense matrix-matrix multiplications, High arithmetic intensity, just a Strided Batched GEMM call by vendor optimized library, trivially extendable from single vector case

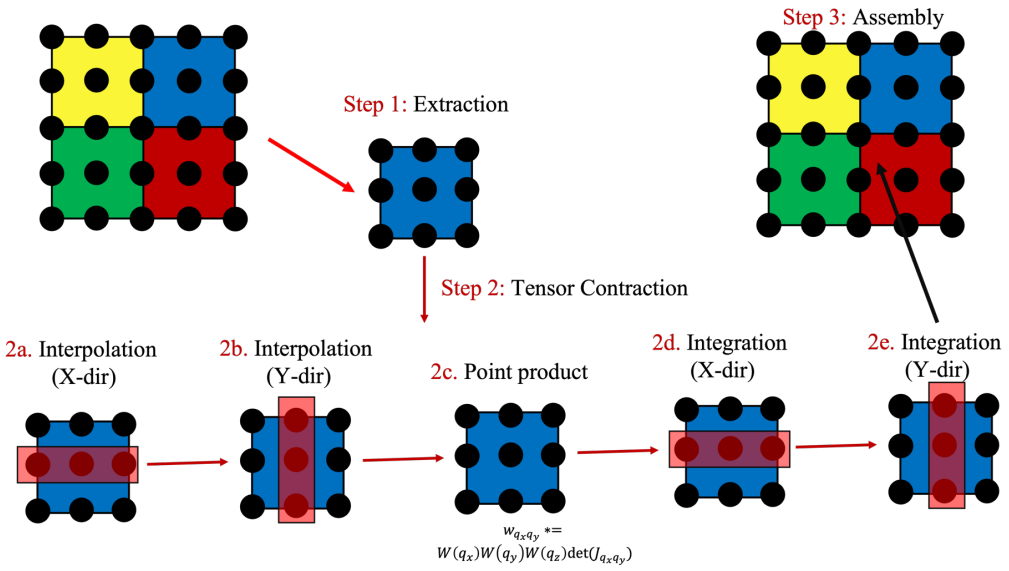
Can we use Matrix-Free methods for multi-vector products?

- Matrix-Free → Sequence of small matrix multiplications, Low arithmetic intensity, non-trivial for multi-vectors.

- Current open-source implementations of matrix-free methods

→ Not directly applicable for the action of a FE discretized operator (**A**) on a large number of FE discretized fields (**U**).

Goal : Develop efficient matrix-free implementation for evaluating matrix multi-vector products AU



Multi-vector products involving Helmholtz-like operator (AU)

- Consider a model problem with the Helmholtz-like operator $\mathcal{F}$

$$\mathcal{F}u^i(\mathbf{x}) = -\mu\nabla^2 u^i(\mathbf{x}) + \kappa(\mathbf{x})u^i(\mathbf{x}) = \begin{cases} f^i(\mathbf{x}) \\ \lambda^i u^i(\mathbf{x}) \end{cases} \quad \forall \mathbf{x} \in \Omega \quad \text{such that} \quad u^i(\mathbf{x}) = u_D(\mathbf{x}) \quad \forall \mathbf{x} \in \partial\Omega_D$$

$$u^i(\mathbf{x}) \approx u^{i,h}(\mathbf{x}) = \sum_J^m u_J^i N_J(\mathbf{x})$$

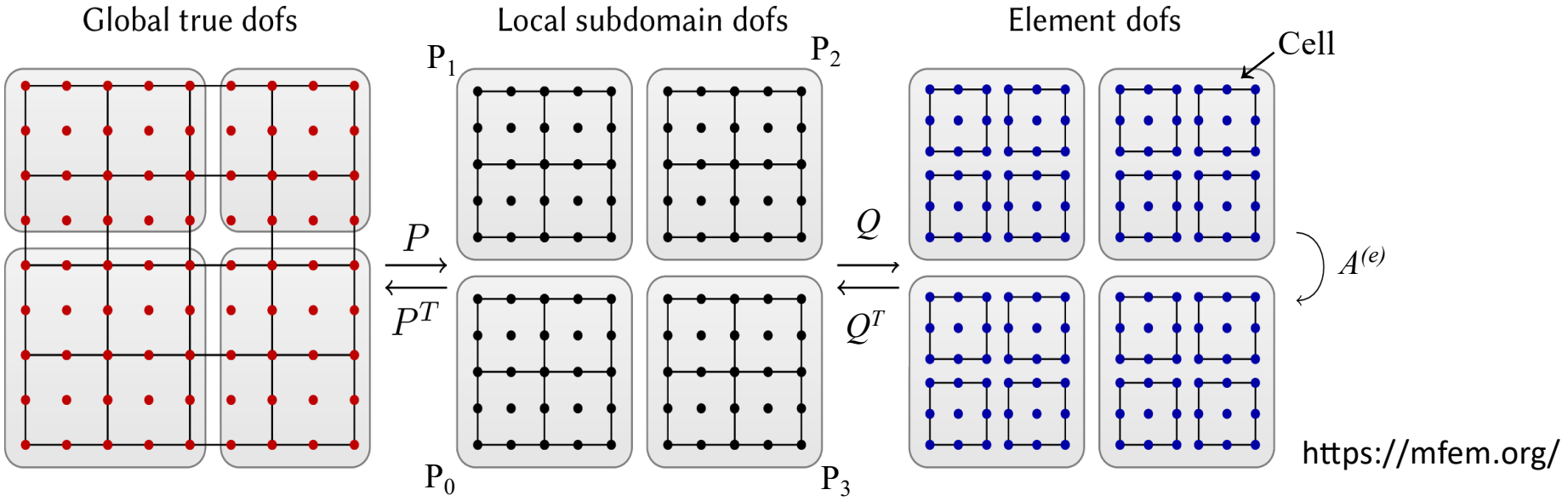
$$\mathbf{AU} = \mathbf{KU} + \mathbf{M}_\kappa\mathbf{U} = \begin{cases} \mathbf{F} \\ \mathbf{MU\Lambda} \end{cases} \quad \text{such that} \quad U_{ki} = u_D(\mathbf{x}_k) \quad \forall \mathbf{x}_k \in \partial\Omega_D$$

where multi-vectors $\mathbf{U} = \begin{bmatrix} \boldsymbol{u}^1 & \boldsymbol{u}^2 & \dots & \boldsymbol{u}^{n_v} \end{bmatrix}$ and $\mathbf{F} = \begin{bmatrix} \boldsymbol{f}^1 & \boldsymbol{f}^2 & \dots & \boldsymbol{f}^{n_v} \end{bmatrix}$ $\quad \boldsymbol{\Lambda}$ = Diagonal Matrix of Eigenvalues

$$K_{IJ} = \int_\Omega \mu\nabla N_I(\mathbf{x}) \cdot \nabla N_J(\mathbf{x})\, d\mathbf{x} \qquad M_{IJ} = \int_\Omega N_I(\mathbf{x})\, N_J(\mathbf{x})\, d\mathbf{x} \qquad M_{IJ}^\kappa = \int_\Omega \kappa(\mathbf{x})\, N_I(\mathbf{x})\, N_J(\mathbf{x})\, d\mathbf{x}$$

Proposed Matrix-Free Algorithm for AU

$$\mathbf{V} = \sum_{t}^{n_t} \mathbf{P}^{(t)^T} \mathbf{C}^{(t)^T} \left(\sum_{i_b}^{n_b} \sum_{e}^{E_t} \mathbf{B}^{(i_b,t)^T} \mathbf{Q}^{(i_b,e,t)^T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \right) \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$



$\mathbf{P}^{(t)}$: Subdomain level extraction
$\mathbf{C}^{(t)}$: Constraints
$\mathbf{Q}^{(i_b,e,t)}$: FE cell-level extraction
$\mathbf{B}^{(i_b,t)}$: Batch extraction

$\mathbf{Q}^{(i_b,e,t)^T}$: FE cell-level assembly
$\mathbf{B}^{(i_b,t)^T}$: Batch assembly
$\mathbf{C}^{(t)^T}$: Constraints
$\mathbf{P}^{(t)^T}$: Subdomain level assembly

$\mathbf{A}^{(e)}$: FE cell-level matrix

Matrix-free Algorithm for AU

$$\mathbf{V} = \sum_{t}^{n_t} \mathbf{P}^{(t)^T} \mathbf{C}^{(t)^T} \left(\sum_{i_b}^{n_b} \sum_{e}^{E_t} \mathbf{B}^{(i_b,t)^T} \mathbf{Q}^{(i_b,e,t)^T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \right) \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

$$\mathbf{A} = \mathbf{K} + \mathbf{M}_\kappa \rightarrow \mathbf{A}^{(e)} = \mathbf{K}^{(e)} + \mathbf{M}_\kappa^{(e)}$$

$$\begin{aligned} K_{IJ}^{(e)} &= \int_{\Omega^{(e)}} \mu \nabla N_I \cdot \nabla N_J d\mathbf{x} \\ &= \int_{\hat{\Omega}} \mu \left(\mathbf{J}^{(e)^{-T}} \nabla_{\xi} \hat{N}_I \right) \cdot \left(\mathbf{J}^{(e)^{-T}} \nabla_{\xi} \hat{N}_J \right) \det \mathbf{J}^{(e)} d\hat{\mathbf{x}} \\ &= \sum_{Q=1}^{n_q^3} \left(\nabla_{\xi} \hat{N}_I \right)^T \mathbf{J}^{(e)^{-1}} \mathbf{J}^{(e)^{-T}} \left(\nabla_{\xi} \hat{N}_J \right) \mu w_Q \det \mathbf{J}^{(e)} \Big|_{\hat{\xi}_Q} \end{aligned}$$

$$\begin{aligned} M_{\kappa IJ}^{(e)} &= \int_{\Omega^{(e)}} \kappa(\mathbf{x}) N_I N_J d\mathbf{x} \\ &= \int_{\hat{\Omega}} \kappa(\hat{\mathbf{x}}) \hat{N}_I \hat{N}_J det\mathbf{J}^{(e)} d\hat{\mathbf{x}} \\ &= \sum_{Q=1}^{n_q^3} \kappa \hat{N}_I \hat{N}_J w_Q det\mathbf{J}^{(e)} \Big|_{\hat{\xi}_Q} \end{aligned}$$

$\mathbf{J}^{(e)} \longrightarrow$ Jacobian matrix of the map from $\Omega^{(e)}$ to $\widehat{\Omega}$ (reference cell)

$\hat{\xi}_Q \longrightarrow$ Quadrature points in $\widehat{\Omega}$

$n_q \longrightarrow$ No of quadrature points in each direction

Matrix-free Algorithm for AU

$$\mathbf{V} = \sum_{t}^{n_t} \mathbf{P}^{(t)^T} \mathbf{C}^{(t)^T} \left(\sum_{i_b}^{n_b} \sum_{e}^{E_t} \mathbf{B}^{(i_b,t)^T} \mathbf{Q}^{(i_b,e,t)^T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \right) \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

$$D_{QI}^{(s)} = \nabla_{\xi} \widehat{N}_I \left(\widehat{\boldsymbol{\xi}}_Q \right) \cdot \widehat{\boldsymbol{n}}_s$$

$$N_{QI} = \widehat{N}_I(\widehat{\boldsymbol{\xi}}_Q)$$

$$\mathcal{G}_{QQ}^{(s,d)} = \left[\left(\mathbf{J}^{(e)} \right)^{-1} \left(\mathbf{J}^{(e)} \right)^{-T} \right]_{sd} \det \mathbf{J}^{(e)} \mu w_Q \Big|_{\widehat{\boldsymbol{\xi}}_Q}$$

$$\mathcal{G}_{QQ} = \kappa \det \mathbf{J}^{(e)} w_Q \Big|_{\widehat{\boldsymbol{\xi}}_Q}$$

$$\mathbf{K}^{(e)} = \begin{bmatrix} \mathbf{D}^{(0)} \\ \mathbf{D}^{(1)} \\ \mathbf{D}^{(2)} \end{bmatrix}^T \begin{bmatrix} \mathcal{G}^{(0,0)} & \mathcal{G}^{(0,1)} & \mathcal{G}^{(0,2)} \\ \mathcal{G}^{(1,0)} & \mathcal{G}^{(1,1)} & \mathcal{G}^{(1,2)} \\ \mathcal{G}^{(2,0)} & \mathcal{G}^{(2,1)} & \mathcal{G}^{(2,2)} \end{bmatrix} \begin{bmatrix} \mathbf{D}^{(0)} \\ \mathbf{D}^{(1)} \\ \mathbf{D}^{(2)} \end{bmatrix}$$

$$\mathbf{M}_{\kappa}^{(e)} = \mathbf{N}^T \mathcal{G} \mathbf{N}$$

Matrix-free Algorithm for AU

$$\mathbf{V} = \sum_{t}^{n_t} \mathbf{P}^{(t)^T} \mathbf{C}^{(t)^T} \left(\sum_{i_b}^{n_b} \sum_{e}^{E_t} \mathbf{B}^{(i_b,t)^T} \mathbf{Q}^{(i_b,e,t)^T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \right) \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

$$\mathbf{U}^{(i_b,e,t)} = \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

$$\mathbf{D}^{(s)} = \widetilde{\mathbf{D}}^{(s)} \mathbf{N} \qquad \widetilde{D}^{(s)}_{\hat{q}q} = \left. \frac{\partial \widetilde{N}_q(\boldsymbol{\xi})}{\partial \xi_s} \right|_{\xi_{\hat{q}}} \qquad \widehat{N}_i(\boldsymbol{\xi}) = \sum_{\hat{q}} \widehat{N}_i(\boldsymbol{\xi}_{\hat{q}}) \widetilde{N}_{\hat{q}}(\boldsymbol{\xi})$$

$$\mathbf{A}^{(e)} \mathbf{U}^{(i_b,e,t)} = \left(\mathbf{K}^{(e)} + \mathbf{M}_{\kappa}^{(e)} \right) \mathbf{U}^{(i_b,e,t)}$$

$$= \mathbf{N}^T \left(\begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix}^T \begin{bmatrix} \mathcal{G}^{(0,0)} & \mathcal{G}^{(0,1)} & \mathcal{G}^{(0,2)} \\ \mathcal{G}^{(1,0)} & \mathcal{G}^{(1,1)} & \mathcal{G}^{(1,2)} \\ \mathcal{G}^{(2,0)} & \mathcal{G}^{(2,1)} & \mathcal{G}^{(2,2)} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix} + \mathcal{G} \right) \mathbf{N} \mathbf{U}^{(i_b,e,t)}$$

Exploit Tensor Structure of FE Basis

$$\widehat{N}_J(\boldsymbol{\xi}) = \widehat{N}^{1D}_{j_1}(\xi_1) \widehat{N}^{1D}_{j_2}(\xi_2) \widehat{N}^{1D}_{j_3}(\xi_3)$$

$$w_Q = w^{1D}_{q_1} w^{1D}_{q_2} w^{1D}_{q_3}$$

Summary of key steps in matrix-free Algorithm for AU

Evaluation of AU in matrix-free

$$\mathbf{V} = \sum_{t}^{n_t} \mathbf{P}^{(t)T} \mathbf{C}^{(t)T} \left(\sum_{i_b}^{n_b} \sum_{e}^{E_t} \mathbf{B}^{(i_b,t)T} \mathbf{Q}^{(i_b,e,t)T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \right) \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

$$\mathbf{X}^{(t)} = \mathbf{C}^{(t)} \mathbf{P}^{(t)} \mathbf{U}$$

- ***Extraction*** – $\mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \mathbf{X}^{(t)}$
- ***FE-cell Evaluation*** – $\mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \mathbf{X}^{(t)}$
- ***Assembly*** – $\mathbf{B}^{(i_b,t)^T} \mathbf{Q}^{(i_b,e,t)^T} \mathbf{A}^{(e)} \mathbf{Q}^{(i_b,e,t)} \mathbf{B}^{(i_b,t)} \mathbf{X}^{(t)}$

n_q = Quadrature points in each direction

- $\mathbf{N}^{1D}$ and $\widetilde{\mathbf{D}}^{1D}$ are $n_q \times n_p$ and $n_q \times n_q$ respectively
- Computational complexity ⟶

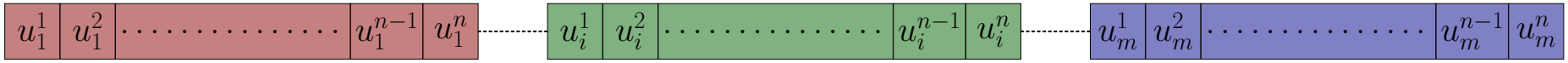
$$O((4(n_p^3 n_q + n_p^2 n_q^2 + n_p n_q^3) + 12n_q^4 + 3n_q^3) E n_v)$$

- For $n_p = n_q$, Computational complexity = $O(E n_v n_p^4)$
- Reduction of $O(n_p^2)$ computational complexity over cell-matrix approach $O(E n_v n_p^6)$
- Memory footprint ⟶ $O(n_p^2)$ for Matrix-Free while cell-matrix has $O(E n_p^6 + E n_v n_p^3)$

Numerical implementation aspects on GPUs

Data Layout: Storage of U

Contiguous Vector (CV) Layout (Existing Literature)



Batched Contiguous Vector (BCV) Layout (Proposed)

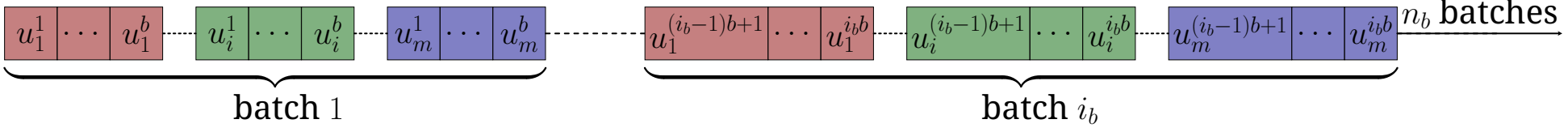


Figure: Pictorial depiction of the CV and BCV layouts. Here $\boldsymbol{u}_J^\beta = u^\beta(\mathbf{x}_J)$ with β representing the vector index and J representing the dof index.

Batched Contiguous Vector **(BCV) Layout :**

- Improved data locality
- Batches amenable to parallelism
- Batchsize tailored to hardware architectures

Evaluation of AU : First Tensor Contraction

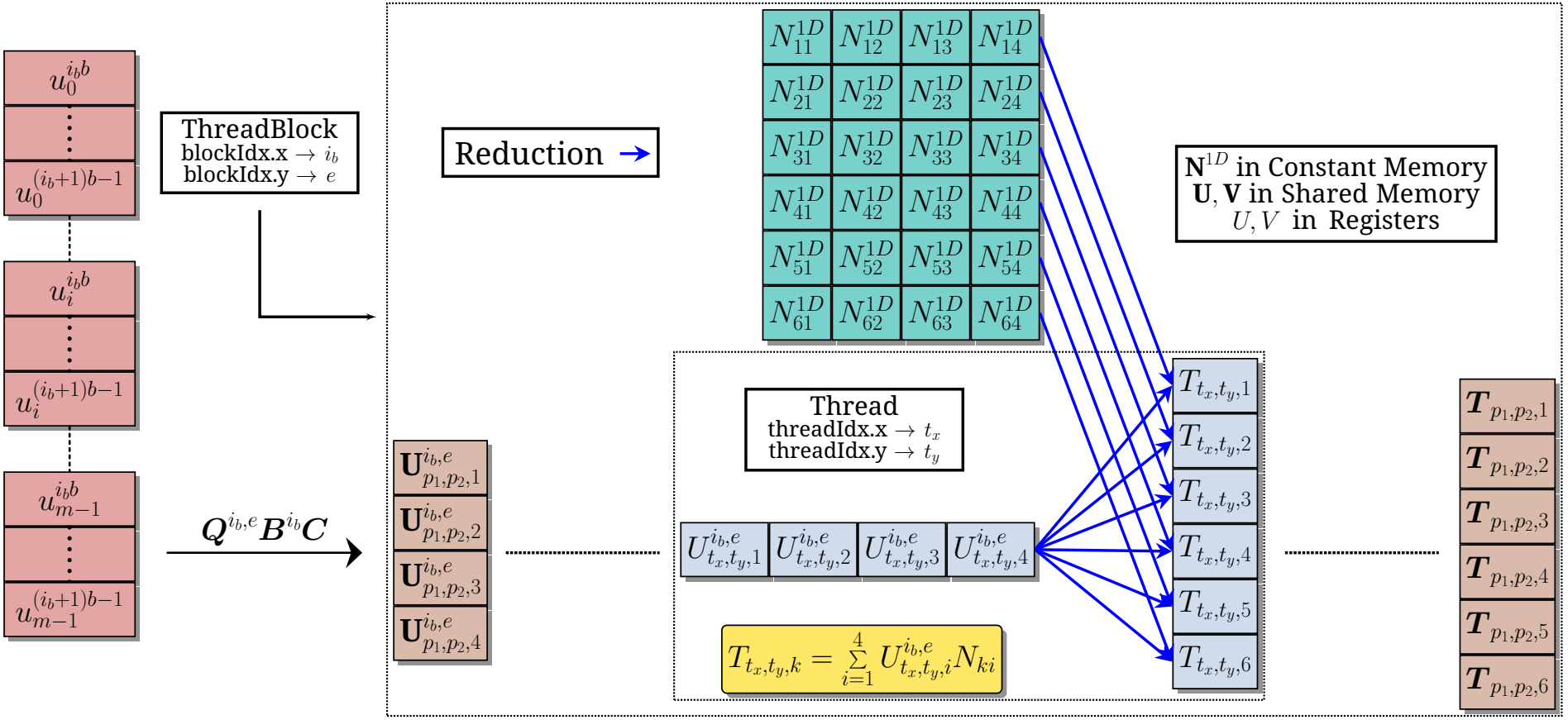


Figure: Pictorial depiction of tensor contractions done on GPUs. The extraction and first tensor contraction steps of evaluation of $\mathbf{A}^{(e)}\mathbf{U}^{(i_b,e,t)}$ are depicted for the case of $n_p = 4$ and $n_q = 8$. Each block in $\mathbf{U}$ represents n_p^2 sized array of b doubles.

Key Ideas in Proposed Matrix-Free Implementation

- Fuse all the tensor contractions in a single kernel and use shared memory to store the data.
- Fuse the extraction and assembly steps in a single kernel to minimize data movements.
- Store $\mathbf{N}$ and $\tilde{\mathbf{D}}$ in constant memory to take advantage of broadcast and better utilize the GPU pipelines.
- Combine the extraction step and first tensor contraction and perform tensor contractions as linear combinations of columns of $\mathbf{N}$ and $\tilde{\mathbf{D}}$. Thus, the floating-point operations can be started as soon as a portion of $\mathbf{U}$ is read from the device memory, without needing to wait for its complete bn_p^3 data inside shared memory.
- Use registers to store the local sum for each thread, which reduces bank conflicts and better utilizes the hardware.
- Like extraction, assembly and last tensor contraction are also combined to directly add to output $\mathbf{V}$ on device memory.

System Configuration

System Config	Summit Supercomputer
Processor	IBM® POWER9
GPU	NVIDIA® Tesla® V100 SXM2 16GB
Nodes	4608
CPU cores/Node	32
GPUs/Node	6
Node Performance	42 TFLOP/s (V100 FP64)
Memory/Node	512 GB DDR4 + 96 GB HBM2
Interconnect	Mellanox® EDR 100G InfiniBand
OS	RHEL 8.2

Table: System configurations for the benchmark architectures.

Library	GPU Benchmarks
Compiler	gcc 9.1.0 nvcc 11.0
Compiler Flags	`-O3 -arch=sm_70 -lcublas`
MPI	IBM Spectrum MPI 10.4
BLAS	cuBLAS 11.0

Table: External libraries and compiler flags used for compilation.

```
1 metrics+="
2 sm__sass_thread_inst_executed_op_dadd_pred_on.
      sum,\
3 sm__sass_thread_inst_executed_op_dfma_pred_on.
      sum,\
4 sm__sass_thread_inst_executed_op_dmul_pred_on.
      sum"
5
6 ncu --metrics $metrics --profile-from-start
      off --target-processes all $EXECUTABLE
```

Wrapper script for profiling with NVIDIA Nsight Compute 2021.2 for multi-node GPUs

GPU Benchmarks – deal.ii, MFEM and Cell-Matrix

- MFEM-libCEED → DOE Exascale Computing Project for Matrix-Free
- deal.ii → Popular FEM library with Matrix-free support
- Cell-Matrix Method → DFT-FE library (Extended defects in metallic alloys calculation) (**ACM Gordon Bell Prize 2023**) (660 PFLOPS on Frontier)

GPU Benchmarks – deal.ii, MFEM and Cell-Matrix (V100 and A100 GPUs)

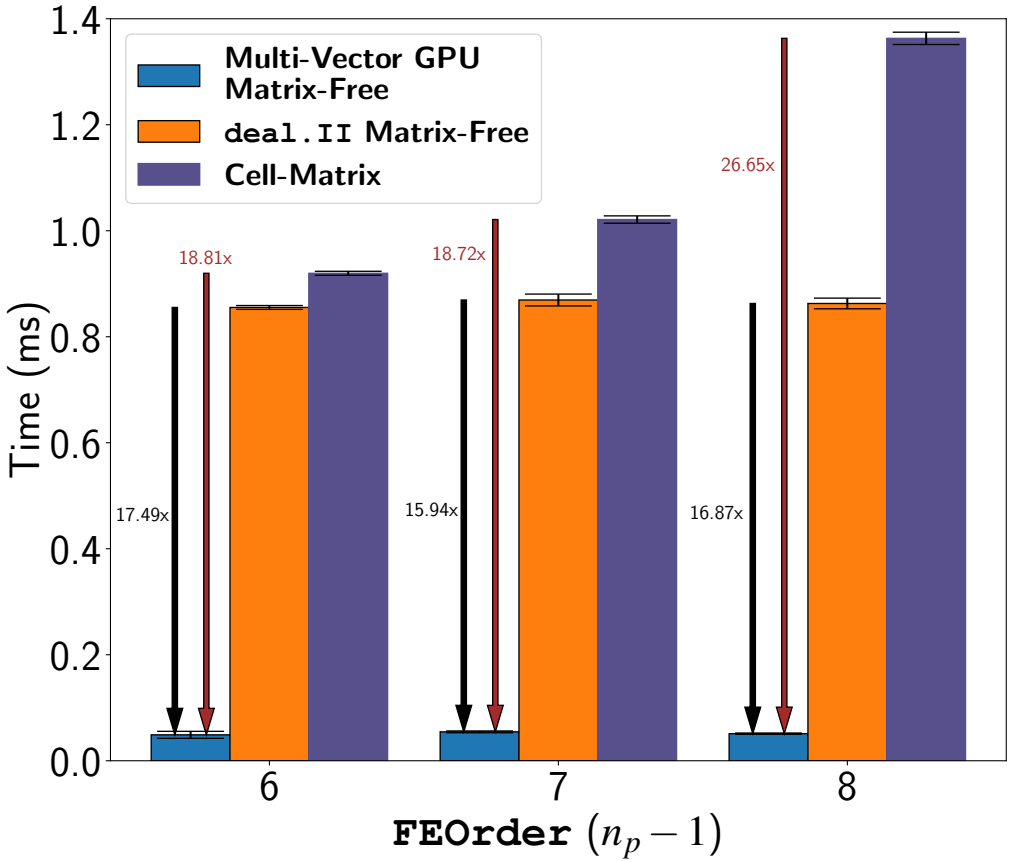
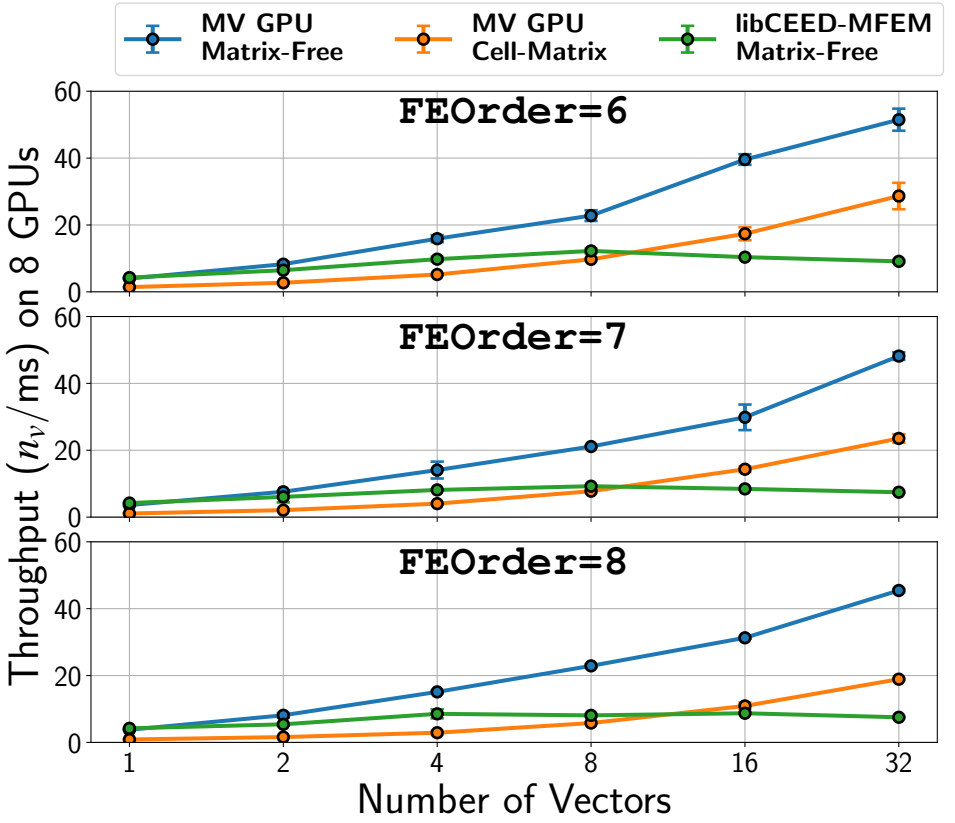


Figure: Comparison of our single-vector matrix-free implementation against `deal.II`'s matrix-free method and the cell-matrix method on a NVIDIA® Tesla® V100 SXM2 16GB (Summit Supercomputer).
Case studies: 117649 DoFs (`FEOrder`=6 and 8); 125000 DoFs (`FEOrder`=7).



6.5x Speedup

Figure: Comparison of our multivector matrix-free implementation against libCEED matrix-free method and the cell-matrix method on 8 NVIDIA® A100-SXM4-80GB GPUs (Selene Supercomputer).
Case studies: 1092727 DoFs (`FEOrder`=6); 1191016 DoFs (`FEOrder`=7); 1157625 DoFs (`FEOrder`=8)

Benchmarks – Peak Performance and Scaling Efficiency (AU)

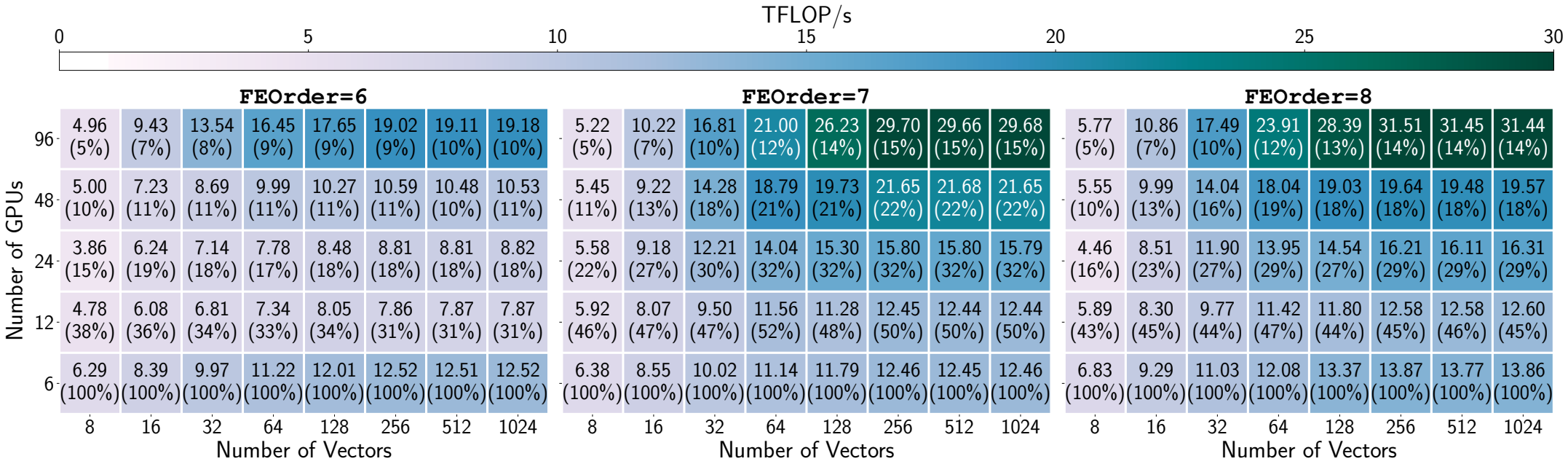


Figure: Scaling study of our matrix-free implementation on 6 to 96 V100 GPUs employing the number of vectors n_v = 8, 16, 32, 64, 128, 256, 512, 1024. For a large number of vectors (512-1024), our implementation results in parallel scaling efficiencies of ~30-50% for 12 GPUs (~90k DoFs/GPU) and ~10-15% for 96 GPUs (~12k DoFs/GPU). Case studies: 1092727 DoFs (`FEOrder=6`); 1191016 DoFs (`FEOrder=7`); 1157625 DoFs (`FEOrder=8`).

- ~30% of peak performance of 1 Summit node

GPU Benchmarks – Roofline

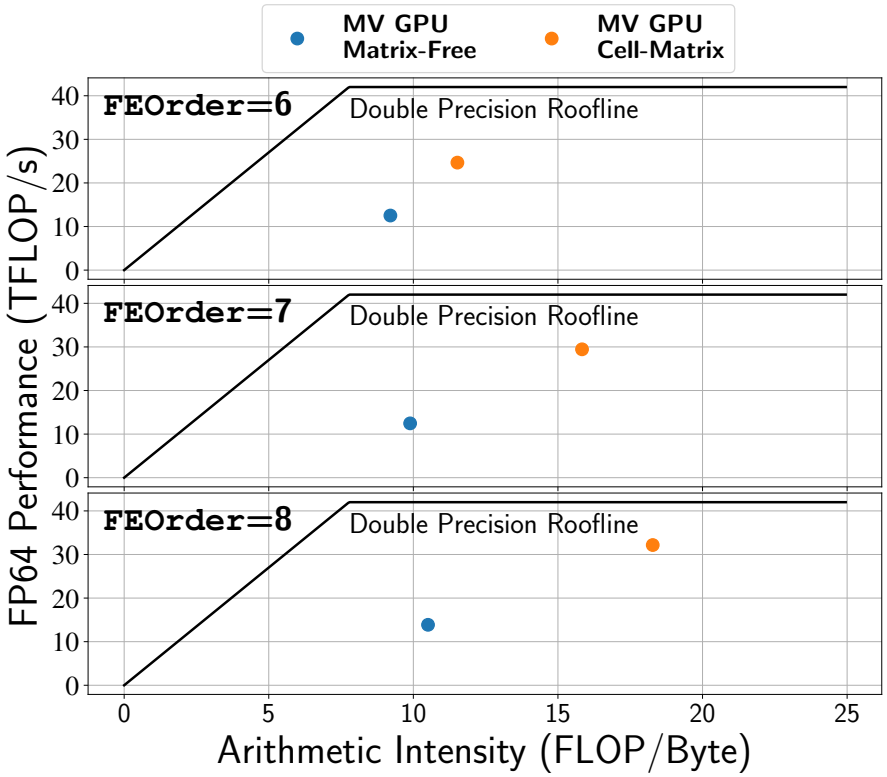
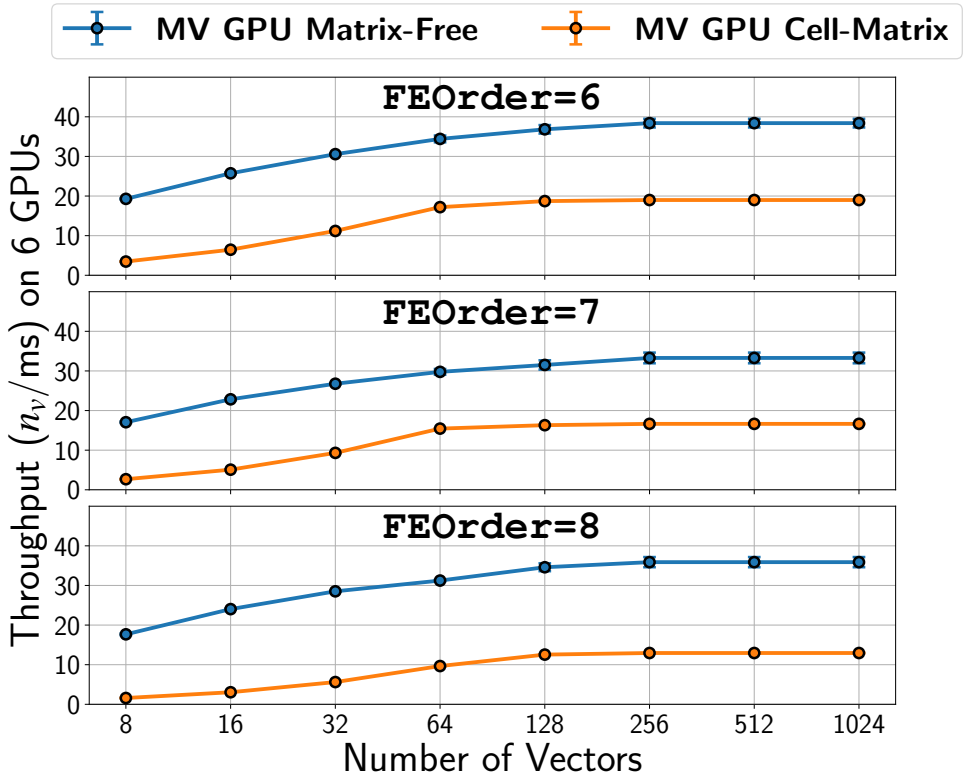


Figure: Roofline analysis of our matrix-free implementation and the cell-matrix implementation for $n_v = 1024$ with $n_q = n_p$ for uniform meshes on 1 node of Summit Supercomputer. Case studies: 1092727 DoFs (`FEOrder`=6); 1191016 DoFs (`FEOrder`=7); 1157625 DoFs (`FEOrder`=8) for the Helmholtz problem on V100 GPUs.



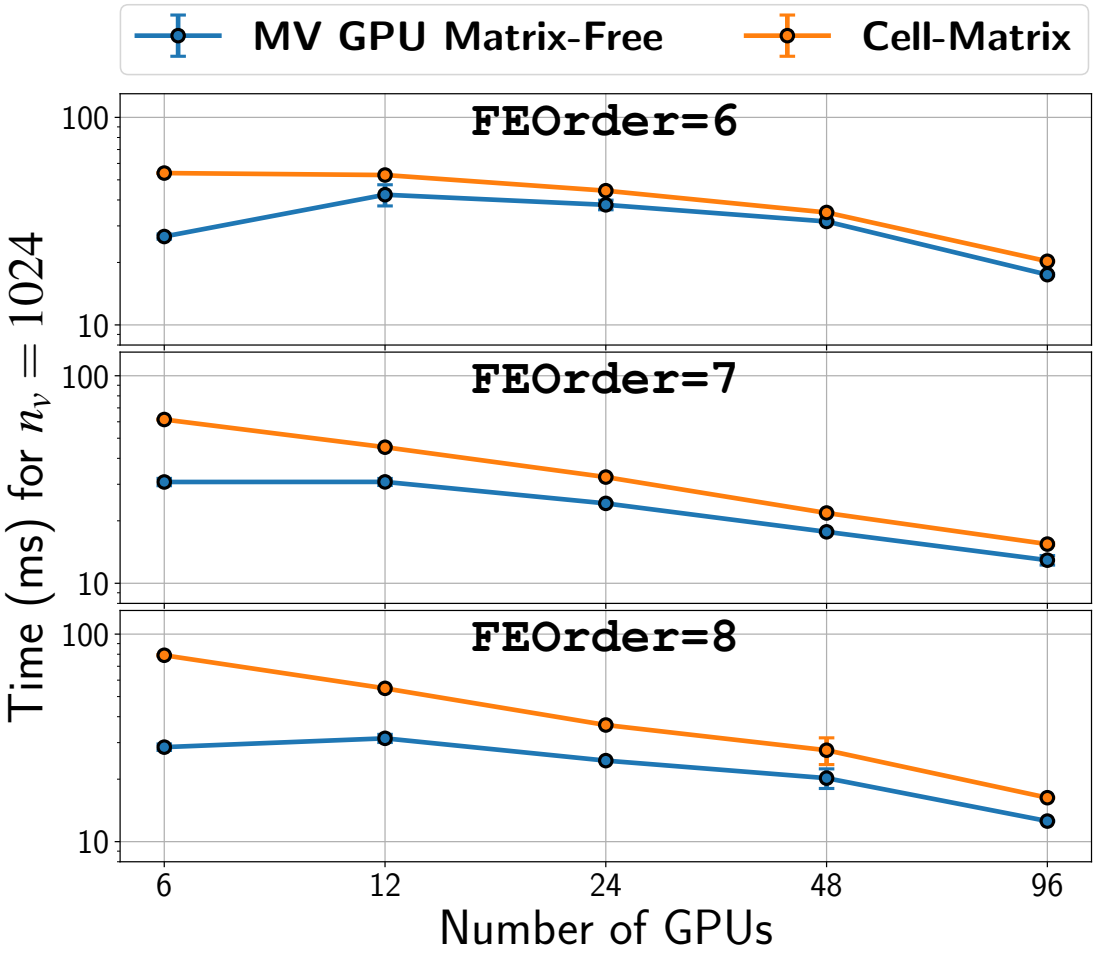
1024 vectors, 1 node (6 GPUs, ~200k DoFs/GPU)

FEOrder 6 : 2.0x

FEOrder 7 : 2.0x

FEOrder 8 : 2.8x

V100 GPU Benchmarks – Scaling Study (AU)



1024 vectors

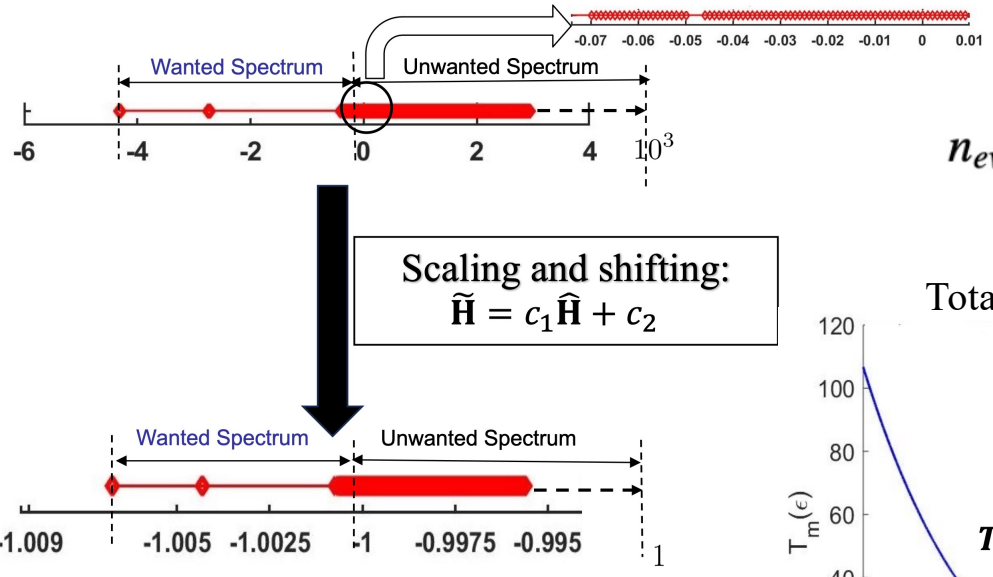
1 node (6 GPUs, ~200k DoFs/GPU) : 2.8x
4 nodes (24 GPUs , ~45k DoFs/GPU) : 1.5x
16 nodes (96 GPUs, ~12k DoFs/GPU) : 1.3x

Figure: Scaling study comparisons of the proposed matrix-free multivector implementation with cell-matrix baseline for 1024 vectors. Case studies: 1092727 DoFs (`FEOrder`=6); 1191016 DoFs (`FEOrder`=7); 1157625 DoFs (`FEOrder`=8) for the Helmholtz problem on V100 GPUs.

Eigenvalue Problem (Chebyshev Filtered Subspace Iteration)

$$\mathbf{HU} = \mathbf{MU\Lambda} \qquad \mathbf{H} = \mathbf{K} + \mathbf{M}^{\kappa}$$

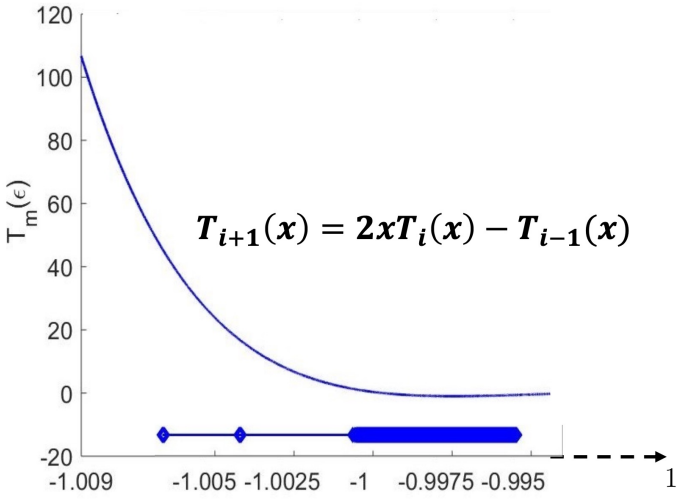
$$\widetilde{\mathbf{H}} = \mathbf{M}^{-1/2}\mathbf{H}\mathbf{M}^{-1/2} \qquad \widetilde{\mathbf{U}} = \mathbf{M}^{1/2}\mathbf{U} \qquad \widetilde{\mathbf{H}}\widetilde{\mathbf{U}} = \widetilde{\mathbf{U}}\mathbf{\Lambda}$$



$n_{ev} =$ 1024 smallest vectors

$\tau = 5 \times 10^{-5}$ residual

Total with buffer = 1280 vectors



Algorithm : Chebyshev Filtered Subspace Iteration

Input: Initial Guess of $\mathbf{U}$

Data: Chebyshev polynomial order m, estimates of the bounds of the eigenspectrum $\lambda_{max}, \lambda_{min}$, estimate of the upper bound of the wanted spectrum λ_u and the tolerance for the residual τ

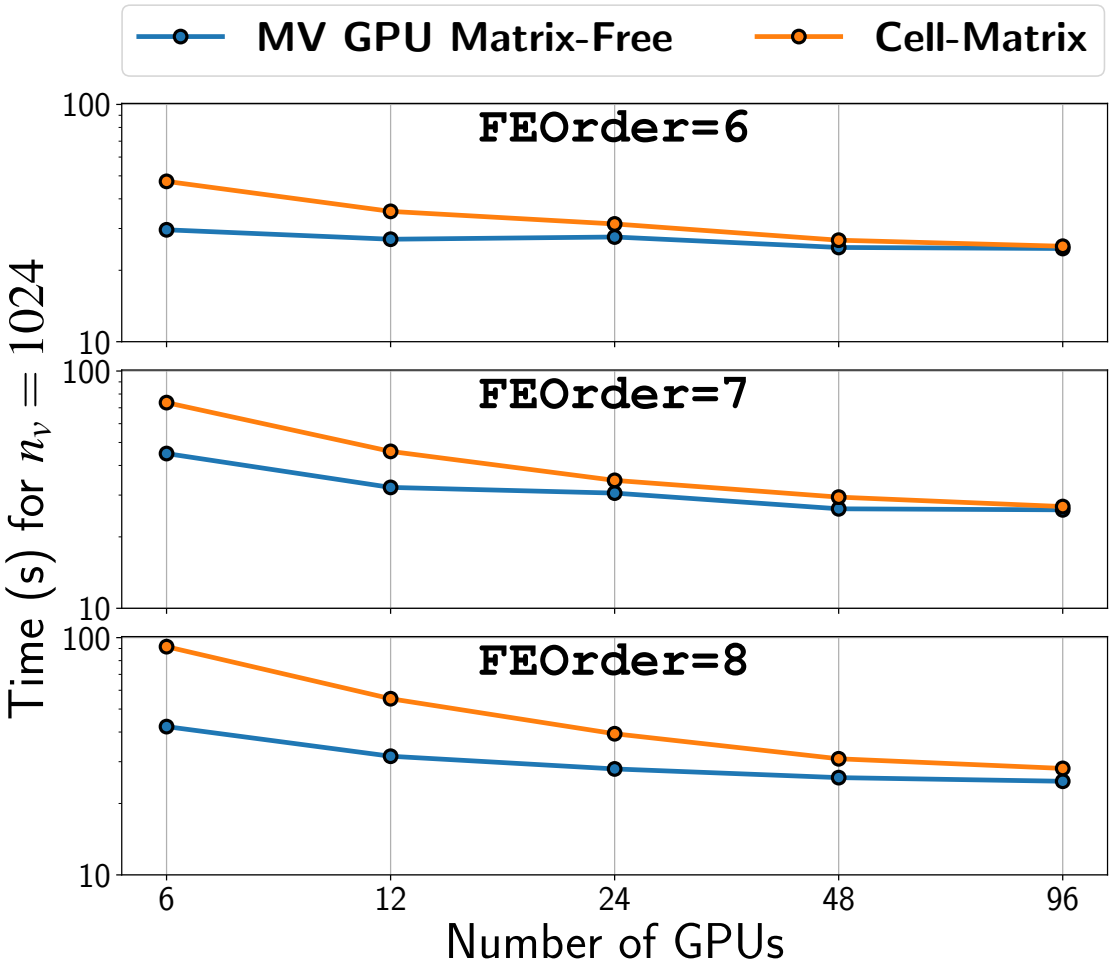
Temporary Variables: $e, c, \sigma, \sigma_1, \gamma, \alpha_1, \alpha_2, \mathbf{X}$ and $\mathbf{Y}$

Result: $\mathbf{U}$ and $\mathbf{\Lambda}$

while $\|\widetilde{H}\mathbf{U} - \mathbf{U\Lambda}\| > \tau$ **do**
- $e \leftarrow \frac{\lambda_{max}-\lambda_u}{2}$;
- $c \leftarrow \frac{\lambda_{max}+\lambda_u}{2}$;
- $\sigma \leftarrow \frac{e}{\lambda_{min}-c}$;
- $\sigma_1 \leftarrow \sigma$;
- $\gamma \leftarrow \frac{2}{\sigma_1}$;
- $\alpha_1 \leftarrow \frac{\sigma_1}{e}$;
- $\alpha_2 \leftarrow -c$;
- $\mathbf{Y} \leftarrow 0$;
- $\mathbf{X} \leftarrow \mathbf{U}$;
- $\mathbf{Y} \leftarrow \alpha_1\widetilde{H}\mathbf{X} + \alpha_1\alpha_2\mathbf{X}$;
- **for** $d \leftarrow 2$ **to** m **do**
  - $\sigma_2 \leftarrow \frac{1}{\gamma-\sigma}$;
  - $\alpha_1 \leftarrow \frac{2\sigma_2}{e}$;
  - $\alpha_2 \leftarrow -\sigma\sigma_2$;
  - $\mathbf{X} \leftarrow \alpha_1\widetilde{\mathbf{H}}\mathbf{Y} + \alpha_2\mathbf{X} - c\alpha_1\mathbf{Y}$;
  - swap($\mathbf{X}, \mathbf{Y}$);
- $\mathbf{X} \leftarrow \mathbf{Y}$;
- $\mathbf{X}_o \leftarrow$ orthogonalize($\mathbf{X}$);
- solve $\mathbf{X}_o^T\widetilde{\mathbf{H}}\mathbf{X}_o\mathbf{Q} = \mathbf{Q\Lambda}$;
- $\mathbf{U} \leftarrow \mathbf{X}_o\mathbf{Q}$;

return $\mathbf{U}$ *and* $\mathbf{\Lambda}$

Summit Benchmarks – Scaling Study (Eigenproblem, Uniform Mesh)



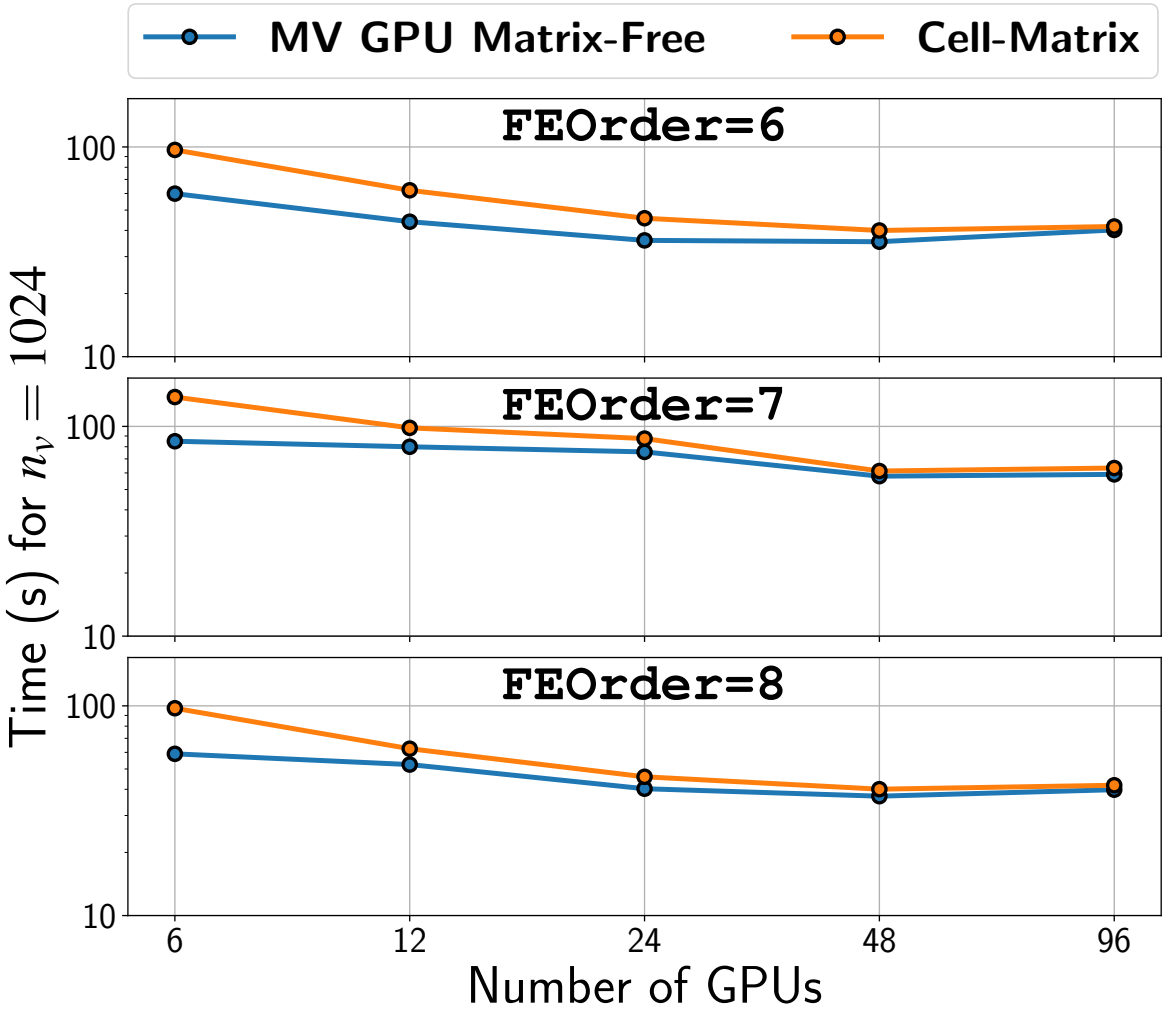
1024 vectors, 1 node (6 GPUs, ~200k DoFs/GPU)
FEOrder 6 : 1.6x
FEOrder 7 : 1.64x
FEOrder 8 : 2.2x

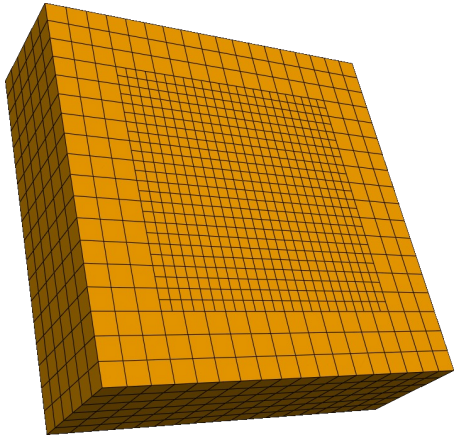
1024 vectors, 4 nodes (24 GPUs, ~45k DoFs/GPU)
FEOrder 6 : 14%
FEOrder 7 : 13%
FEOrder 8 : 41%

1024 vectors, 16 nodes (96 GPUs, ~12k DoFs/GPU)
FEOrder 6, 7, 8 : ~10%

Figure: Performance benchmarks of our matrix-free implementation compared to the cell-matrix implementation for the eigenvalue problem on uniform meshes. Case studies: 1092727 DoFs (`FEOrder`=6); 1191016 DoFs (`FEOrder`=7); 1157625 DoFs (`FEOrder`=8). Chebyshev polynomial orders 67, 76 and 83 were chosen for `FEOrder`=6, 7 and 8 respectively.

Benchmarks – Scaling Study (Eigenproblem, Adaptively Refined Mesh)





1024 vectors, 1 node (6 GPUs, ~200k DoFs/GPU)

FEOrder 6 : 1.62x
FEOrder 7 : 1.62x
FEOrder 8 : 1.65x

1024 vectors, 16 nodes (96 GPUs, ~12k DoFs/GPU)

FEOrder 6, 7, 8 : ~10%

Figure: Performance benchmarks of our matrix-free implementation compared to the cell-matrix implementation for the eigenvalue problem on adaptively refined meshes (1 level of refinement). Case studies: 1185321 DoFs (`FEOrder`=6); 1177963 DoFs (`FEOrder`=7); 1226673 DoFs (`FEOrder`=8). Chebyshev polynomial orders 67, 76, and 83 were chosen for `FEOrder`=6, 7 and 8 respectively.

Beyond Helmholtz Operator

- More realistic problems in various applications
  - Example – Large-scale eigenvalue problems arising in Density Functional Theory (DFT) (applications in quantum modelling of materials)

$$\mathbf{A}^{(e)}\mathbf{U}^{(i_b,e,t)} = \left(\mathbf{K}^{(e)} + \mathbf{M}_{\kappa}^{(e)}\right)\mathbf{U}^{(i_b,e,t)}$$

$$= \mathbf{N}^T \left(\begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix}^T \begin{bmatrix} \mathcal{G}^{(0,0)} & \mathcal{G}^{(0,1)} & \mathcal{G}^{(0,2)} \\ \mathcal{G}^{(1,0)} & \mathcal{G}^{(1,1)} & \mathcal{G}^{(1,2)} \\ \mathcal{G}^{(2,0)} & \mathcal{G}^{(2,1)} & \mathcal{G}^{(2,2)} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix} + \mathcal{G} \right) \mathbf{N}\mathbf{U}^{(i_b,e,t)}$$

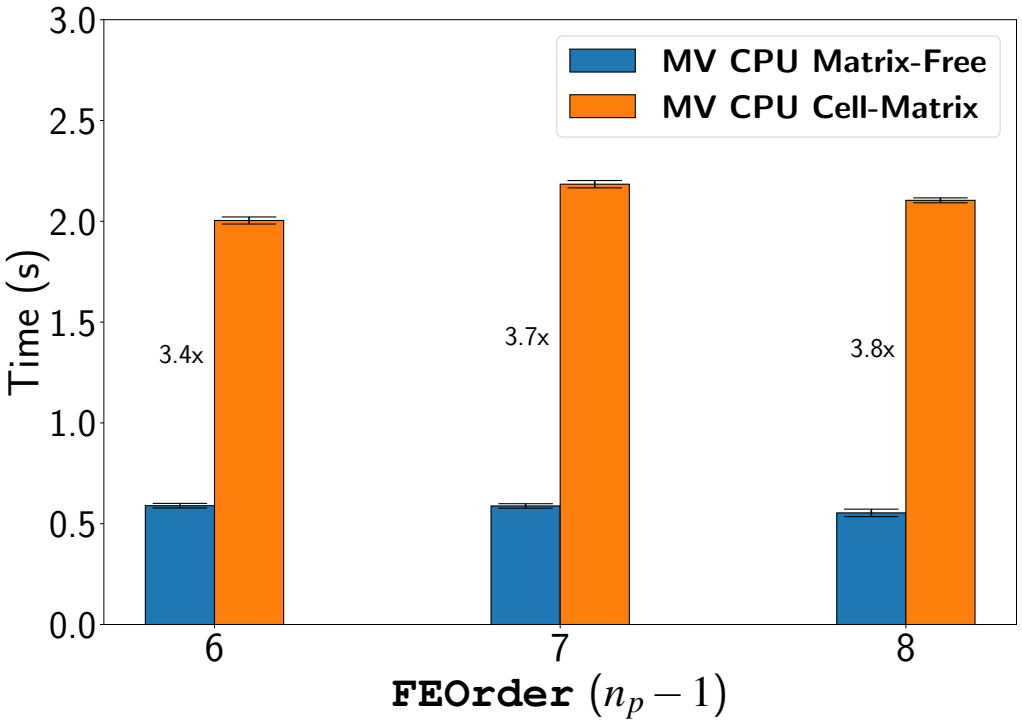
- Adding another term to the Helmholtz operator

$$S_{IJ} = \int_{\Omega} \nabla V \cdot \left(\nabla N_I^h(\mathbf{x})\, N_J^h(\mathbf{x}) + \nabla N_J^h(\mathbf{x})\, N_I^h(\mathbf{x})\right) d\mathbf{x}$$

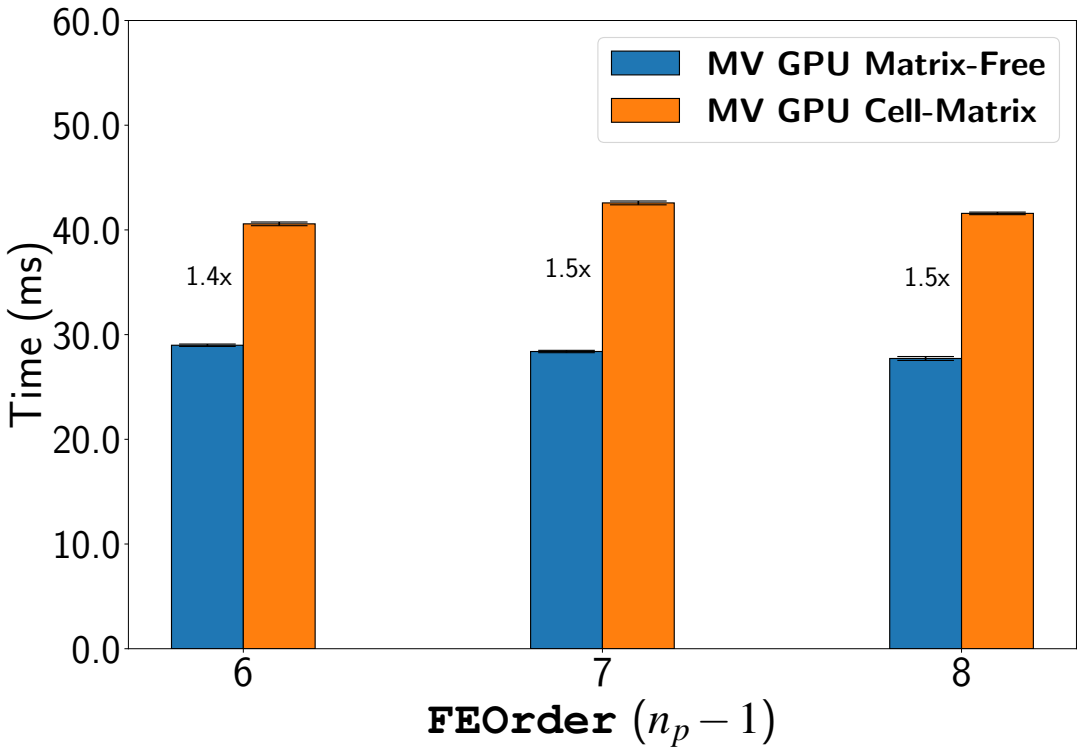
[This operator is motivated from FE discretized operator arising in DFT (GGA Functionals)]

$$\mathbf{S}^{(e)}\mathbf{U}^{(e,t)} = \mathbf{N}^T \left(\begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix}^T \begin{bmatrix} \mathbf{V}_G^{(0)} \\ \mathbf{V}_G^{(1)} \\ \mathbf{V}_G^{(2)} \end{bmatrix} + \begin{bmatrix} \mathbf{V}_G^{(0)} \\ \mathbf{V}_G^{(1)} \\ \mathbf{V}_G^{(2)} \end{bmatrix}^T \begin{bmatrix} \widetilde{\mathbf{D}}^{(0)} \\ \widetilde{\mathbf{D}}^{(1)} \\ \widetilde{\mathbf{D}}^{(2)} \end{bmatrix} \right) \mathbf{N}\mathbf{U}^{(e,t)}$$

Beyond Helmholtz Operator



1024 vectors, 1 node (Intel Xeon Gold 6248R 48 cores, 2 million DoFs)



1024 vectors, 1 GPU (NVIDIA V100, 1.2 million DoFs)

Publications

- ***G. Panigrahi***, *N. Kodali, D. Panda, P. Motamarri* : Fast hardware-aware matrix-free algorithms for higher-order finite-element discretized matrix multivector products on distributed systems (***Journal of Parallel and Distributed Computing, 2024***) https://doi.org/10.1016/j.jpdc.2024.104925

DFT-FE Library:
https://github.com/dftfeDevelopers/dftfe

- *S. Das, B. Kanungo, V. Subramanian,* ***G. Panigrahi****, P. Motamarri, D. Rogers, P. Zimmerman, V. Gavini* : Large-scale materials modeling at quantum accuracy: Ab initio simulations of quasicrystals and interacting extended defects in metallic alloys (***SC Proceedings '23***) https://doi.org/10.1145/3581784.3627037 (**ACM Gordon Bell Prize 2023**)

Acknowledgement

Collaborators :

- Dr. Phani Motamarri, Assistant Professor, CDS Department, IISc Bangalore (PhD Advisor)

Funding Sources :

Thank You