# Empowering MFEM Using libCEED: Features and Performance Analysis

MFEM Community Workshop – Livermore, California, USA

**Yohann Dudouit** [1], Natalie Beams [2], Jed Brown [3], John Camier [1], Veselin Dobrev [1], Tzanio Kolev [1], Jeremy Thompson [3], Tim Warburton [4],& the CEED Team [1,2,3,4,5,6,7]

October 26, 2023

LLNL-PRES-856535

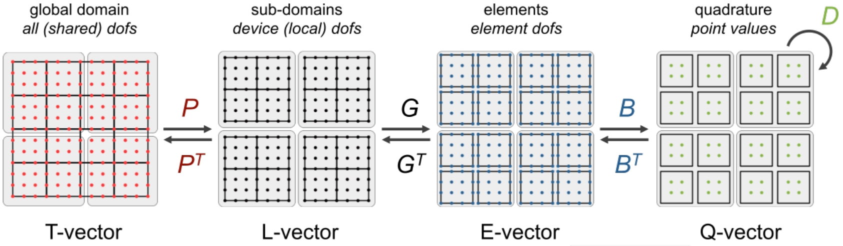
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

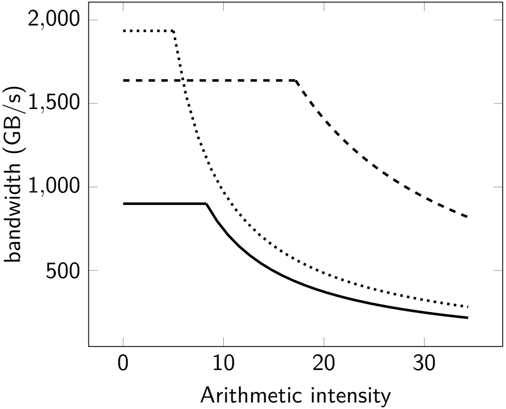
# The CEED project: The partial assembly decomposition

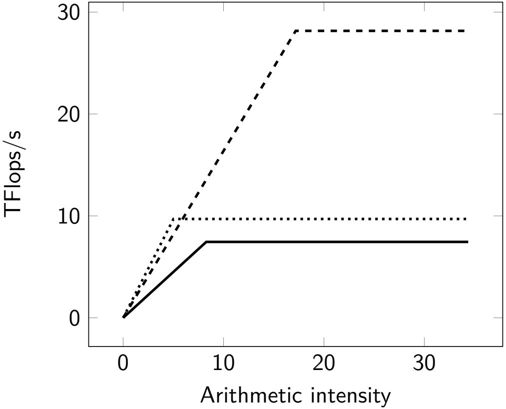
The assembly/evaluation of FEM operators can be decomposed into parallel, **mesh topology**, basis, and geometry/physics components:

$$A = P^T G^T B^T D B G P$$



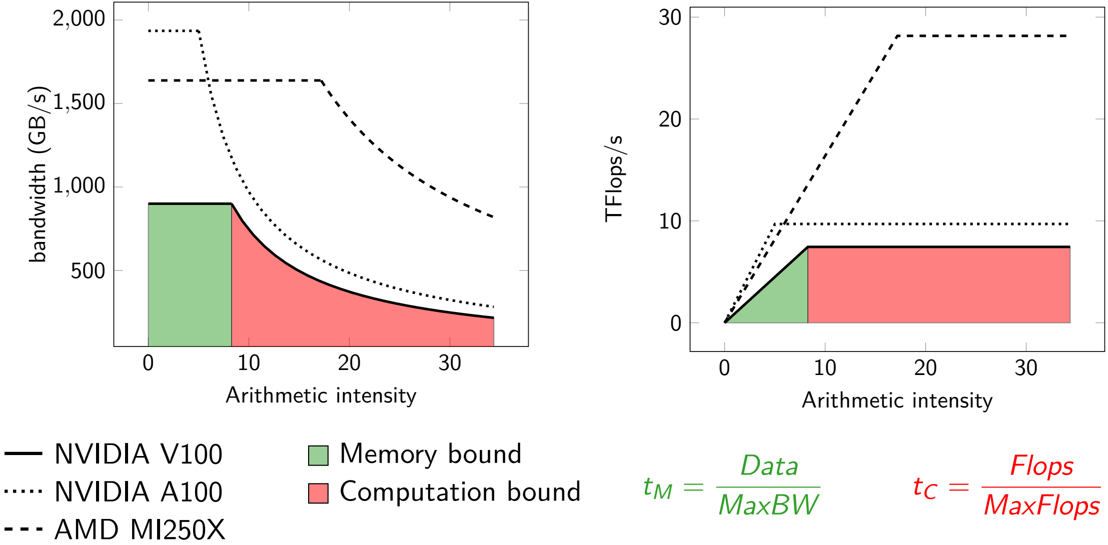
# Roofline model: The two sides of GPU performance



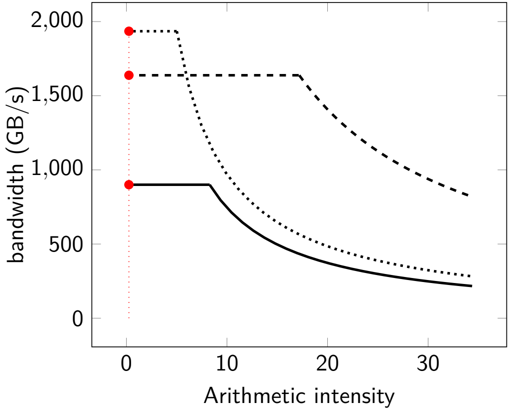

— NVIDIA V100
···· NVIDIA A100
- - - AMD MI250X

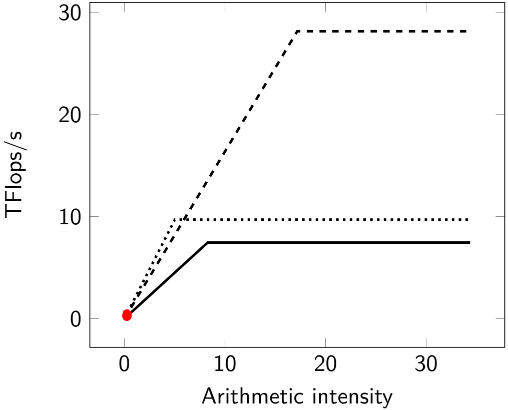
# Roofline model: The two sides of GPU performance




$$t_M = \frac{Data}{MaxBW} \qquad t_C = \frac{Flops}{MaxFlops}$$

# Roofline model: The sparse matrix case

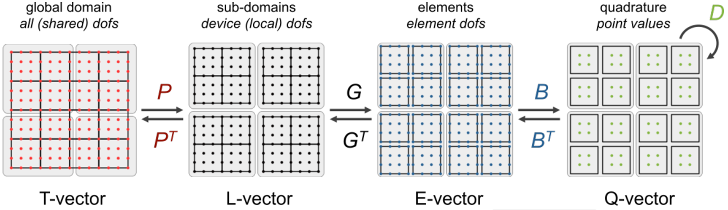



$\Longrightarrow$ The only way to run faster than a sparse matrix is to **move less data**.

# Partial assembly: The algorithmic costs

$$A = P^T G^T B^T D B G P$$




| | Sparse Matrix | Partial Assembly | $G$ | $B$ | $D$ |
|---|---|---|---|---|---|
| Amount of storage | $\mathcal{O}(np^{2d})$ | $\mathcal{O}(np^d)$ | $\mathcal{O}(np^d)$ | $\mathcal{O}(p^{2d})$ | $\mathcal{O}(np^d)$ |
| FLOPs to apply | $\mathcal{O}(np^{2d})$ | $\mathcal{O}(np^{2d})$ | $\mathcal{O}(np^d)$ | $\mathcal{O}(np^{2d})$ | $\mathcal{O}(np^d)$ |
| Arithmetic intensity | $\mathcal{O}(1)$ | $\mathcal{O}(p^d)$ | $\mathcal{O}(1)$ | $\mathcal{O}(p^d)$ | $\mathcal{O}(1)$ |

## Potential speedup:

Data movement and storage is reduced from $\mathcal{O}(np^{2d})$ to $\mathcal{O}(np^d)$ to apply the finite element operator, potential speedup for Partial Assembly of $\sim \mathcal{O}(p^d)$.

# Reducing the arithmetic intensity: The sum factorization trick

On tensor product finite elements, the $B$ operator can be computed as:

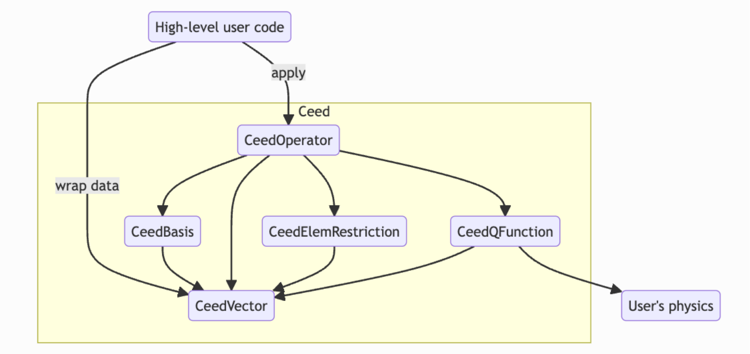
$$v_{k_1 k_2 k_3} = B_{IK} u_I = \underbrace{\sum_{i_1, i_2, i_3} u_{i_1 i_2 i_3} \varphi_{i_1}(x_{k_1}) \varphi_{i_2}(x_{k_2}) \varphi_{i_3}(x_{k_3})}_{\mathcal{O}(p^6) = \mathcal{O}(p^{2d})}$$

$$= \sum_{i_3} \varphi_{i_3}(x_{k_3}) \left( \sum_{i_2} \varphi_{i_2}(x_{k_2}) \left( \underbrace{\sum_{i_1} \varphi_{i_1}(x_{k_1}) u_{i_1 i_2 i_3}}_{\mathcal{O}(p^4) = \mathcal{O}(p^{d+1})} \right) \right)$$

$$= \tilde{B}_{i_3 k_3} \otimes \tilde{B}_{i_2 k_2} \otimes \tilde{B}_{i_1 k_1} u_{i_1 i_2 i_3}$$

| | No Sum Factorization | Sum Factorization |
|---|---|---|
| Amount of storage for $B$ | $\mathcal{O}(p^{2d})$ | $\mathcal{O}(p^2)$ |
| FLOPs to apply | $\mathcal{O}(p^{2d})$ | $\mathcal{O}(p^{d+1})$ |
| Arithmetic intensity | $\mathcal{O}(p^d)$ | $\mathcal{O}(p)$ |

# The libCEED core interface




| | |
|---|---|
| CeedOperator | $A = P^T G^T B^T D B G P$ |
| CeedBasis | $B$ |
| CeedElemRestriction | $G$ |
| CeedQFunction | $D$ |
| CeedVector | Wrapper for degrees-of-freedom/data at quadrature points |

# Features of the libCEED library

| Backend | Description |
| --- | --- |
| AVX | Optimized cpu backend taking advantage of AVX instructions. |
| CUDA | Pure CUDA backend using JIT compilation. |
| HIP | Pure HIP backend using JIT compilation. |
| SYCL | Pure SYCL backend using JIT compilation. |
| Magma | Backend leveraging the Magma library, high performance on non-tensor elements. |
| XSMM | Backend leveraging the libXSMM library, highest cpu performance. |
| OCCA | Backend based on the OCCA abstraction layer. |

Extra libCEED features:

- Provide an interface to compute the diagonal of any operator,
- Provide an interface to assemble a sparse-matrix for any operator,
- Provide an interface for p-multigrid (Jeremy Thompson).

# Features of the libCEED integration in MFEM

## Using the libCEED backend:

- **MFEM_USE_CEED=YES**.
- **-d ceed-cpu/ceed-cuda/ceed-hip**.
- Specific libCEED backends can be selected using **:**, e.g. **-d ceed-hip:/gpu/hip/magma**.

| Supported MFEM Integrators | Weak form |
|---|---|
| MassIntegrator | $\int u\, v$ |
| VectorMassIntegrator | $\int \mathbf{u} \cdot \mathbf{v}$ |
| ConvectionIntegrator | $\int (\mathbf{a} \cdot \nabla u)\, v$ |
| VectorConvectionNLFIntegrator | $\int c(\nabla \mathbf{u}\mathbf{u}) \cdot \mathbf{v}$ |
| DiffusionIntegrator | $\int c \nabla u \cdot \nabla v$ |
| VectorDiffusionIntegrator | $\int c \nabla \mathbf{u} \cdot \nabla \mathbf{v}$ |

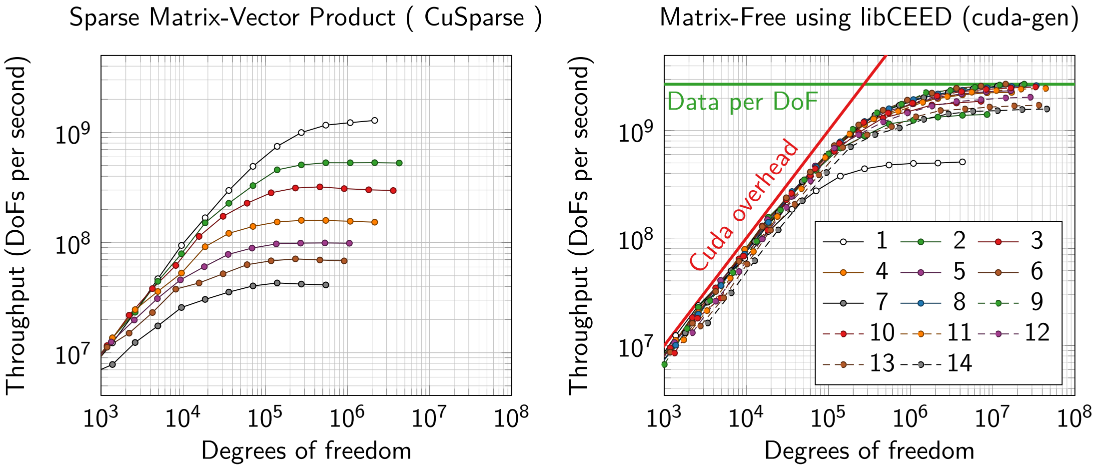
## Pros of the libCEED backend:

- Support for mixed-meshes (including simplicies) and p-adaptivity (limited to serial),
- Interface to construct partial assembly and fully matrix-free operators,
- Algebraic multigrid solver based on the libCEED interface (Andrew Barker).

## Cons of the libCEED backend:

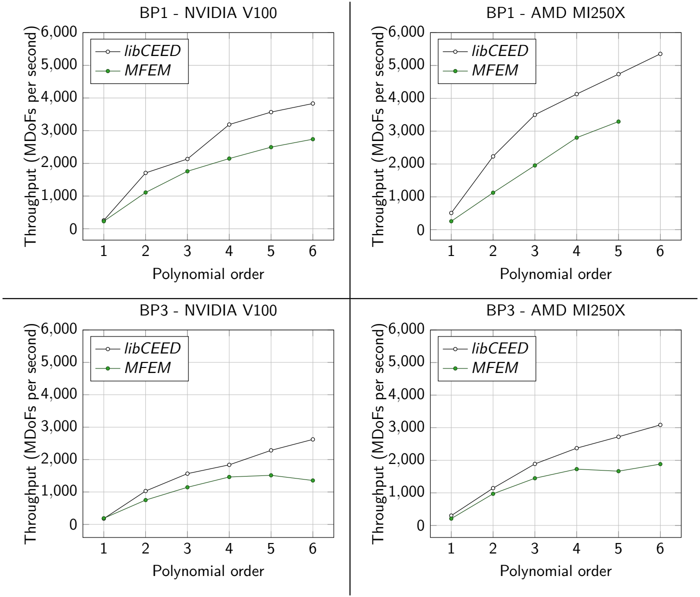
- Does not currently support as many integrators as native MFEM,
- libCEED GPU operators can be "non-deterministic" (use atomic operations).

# Comparing sparse-matrix and matrix-free

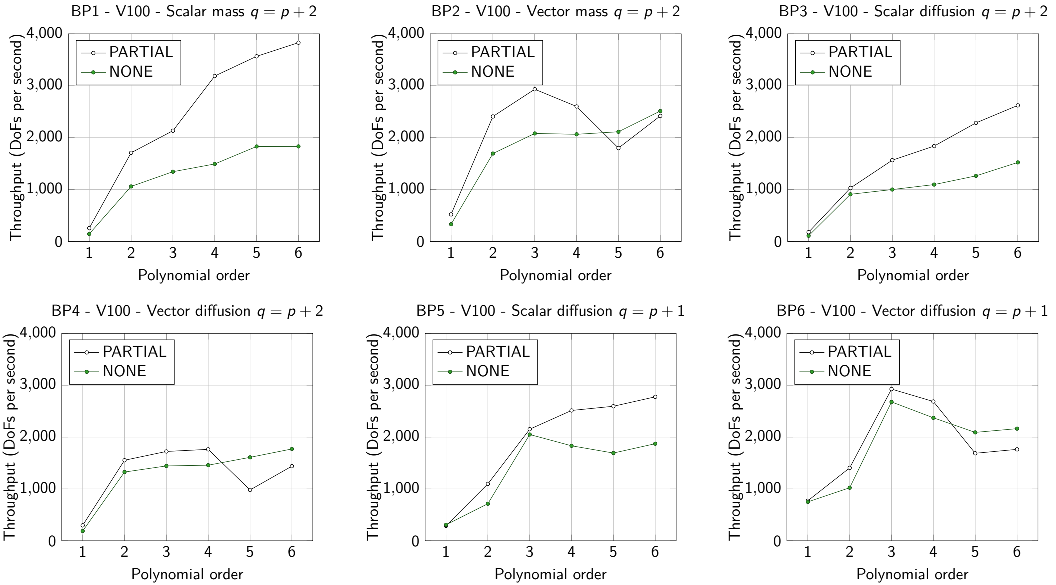



$$t_M = \frac{Data}{MaxBW} \quad \Rightarrow \quad \texttt{MaxThroughput} = \frac{MaxBW}{\texttt{Data per DoF}}$$

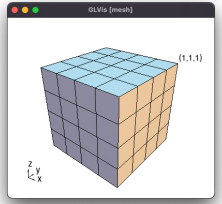
# Comparing native MFEM and the libCEED backend

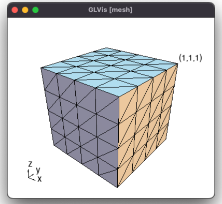


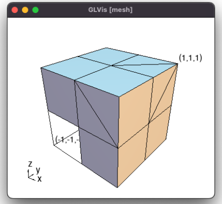
# Comparing the assembly levels: AssemblyLevel::PARTIAL vs AssemblyLevel::NONE

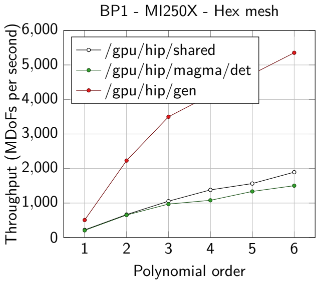


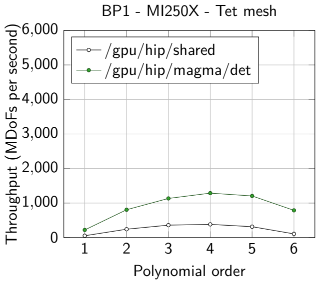
# Performance on simplicies and mixed-meshes

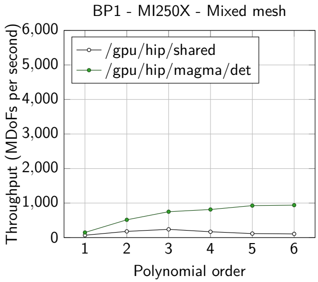


# Key features and future directions

## Key features:

- Competitive performance,
- Run on any hardware ( cpu, CUDA, HIP, SYCL ),
- Support for simplices and mixed-meshes,
- Support for p-adaptivity.

## Future directions:

- Add support for $H(div)$ and $H(curl)$ (non-tensor only),
- Add support for discontinuous Galerkin methods,
- Add support for sparse-matrix assembly through libCEED.

# Disclaimer



Lawrence Livermore National Laboratory