Empowering MFEM Using libCEED: Features and Performance Analysis

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
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:

\[ A = P^T G^T B^T D B G P \]

- **T-vector**: Global domain with all (shared) dofs
- **L-vector**: Sub-domains with device (local) dofs
- **E-vector**: Elements with element dofs
- **Q-vector**: Quadrature point values

\[ P \quad P^T \quad G \quad G^T \quad B \quad B^T \quad D \]
Roofline model: The two sides of GPU performance

![Diagram showing the Roofline model for NVIDIA V100, NVIDIA A100, and AMD MI250X GPUs.](image)

- **NVIDIA V100**
- **NVIDIA A100**
- **AMD MI250X**
Roofline model: The two sides of GPU performance

\[ t_M = \frac{\text{Data}}{\text{MaxBW}} \]

\[ t_C = \frac{\text{Flops}}{\text{MaxFlops}} \]
Roofline model: The sparse matrix case

The only way to run faster than a sparse matrix is to **move less data**.
Partial assembly: The algorithmic costs

\[ A = P^T G^T B^T DBGP \]

<table>
<thead>
<tr>
<th></th>
<th>Sparse Matrix</th>
<th>Partial Assembly</th>
<th>G</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of storage</td>
<td>( O(np^{2d}) )</td>
<td>( O(np^d) )</td>
<td>( O(np^d) )</td>
<td>( O(p^{2d}) )</td>
<td>( O(np^d) )</td>
</tr>
<tr>
<td>FLOPs to apply</td>
<td>( O(np^{2d}) )</td>
<td>( O(np^{2d}) )</td>
<td>( O(np^d) )</td>
<td>( O(np^{2d}) )</td>
<td>( O(np^d) )</td>
</tr>
<tr>
<td>Arithmetic intensity</td>
<td>( O(1) )</td>
<td>( O(p^d) )</td>
<td>( O(1) )</td>
<td>( O(p^d) )</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

Potential speedup:

Data movement and storage is reduced from \( O(np^{2d}) \) to \( O(np^d) \) to apply the finite element operator, potential speedup for Partial Assembly of \( \sim O(p^d) \).
Reducing the arithmetic intensity: The sum factorization trick

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

$$v_{k_1k_2k_3} = B_{IK} u_I = \sum_{i_1, i_2, i_3} u_{i_1i_2i_3} \varphi_{i_1}(x_{k_1}) \varphi_{i_2}(x_{k_2}) \varphi_{i_3}(x_{k_3})$$

$$= \sum_{i_3} \varphi_{i_3}(x_{k_3}) \left( \sum_{i_2} \varphi_{i_2}(x_{k_2}) \left( \sum_{i_1} \varphi_{i_1}(x_{k_1}) u_{i_1i_2i_3} \right) \right)$$

$$= \tilde{B}_{i_3k_3} \otimes \tilde{B}_{i_2k_2} \otimes \tilde{B}_{i_1k_1} u_{i_1i_2i_3}$$

<table>
<thead>
<tr>
<th></th>
<th>No Sum Factorization</th>
<th>Sum Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of storage for $B$</td>
<td>$O(p^{2d})$</td>
<td>$O(p^2)$</td>
</tr>
<tr>
<td>FLOPs to apply</td>
<td>$O(p^{2d})$</td>
<td>$O(p^{d+1})$</td>
</tr>
<tr>
<td>Arithmetic intensity</td>
<td>$O(p^d)$</td>
<td>$O(p)$</td>
</tr>
</tbody>
</table>
The libCEED core interface

\[ A = P^T G^T B^T D B G P \]

<table>
<thead>
<tr>
<th>CeedOperator</th>
<th>[ A = P^T G^T B^T D B G P ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CeedBasis</td>
<td>( B )</td>
</tr>
<tr>
<td>CeedElemRestriction</td>
<td>( G )</td>
</tr>
<tr>
<td>CeedQFunction</td>
<td>( D )</td>
</tr>
<tr>
<td>CeedVector</td>
<td>Wrapper for degrees-of-freedom/data at quadrature points</td>
</tr>
</tbody>
</table>
## Features of the libCEED library

<table>
<thead>
<tr>
<th>Backend</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX</td>
<td>Optimized cpu backend taking advantage of AVX instructions.</td>
</tr>
<tr>
<td>CUDA</td>
<td>Pure CUDA backend using JIT compilation.</td>
</tr>
<tr>
<td>HIP</td>
<td>Pure HIP backend using JIT compilation.</td>
</tr>
<tr>
<td>SYCL</td>
<td>Pure SYCL backend using JIT compilation.</td>
</tr>
<tr>
<td>Magma</td>
<td>Backend leveraging the Magma library, high performance on non-tensor elements.</td>
</tr>
<tr>
<td>XSMM</td>
<td>Backend leveraging the libXSMM library, highest cpu performance.</td>
</tr>
<tr>
<td>OCCA</td>
<td>Backend based on the OCCA abstraction layer.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Supported MFEM Integrators</th>
<th>Weak form</th>
</tr>
</thead>
<tbody>
<tr>
<td>MassIntegrator</td>
<td>$\int u , v$</td>
</tr>
<tr>
<td>VectorMassIntegrator</td>
<td>$\int u \cdot v$</td>
</tr>
<tr>
<td>ConvectionIntegrator</td>
<td>$\int (a \cdot \nabla u) , v$</td>
</tr>
<tr>
<td>VectorConvectionNLFIntegrator</td>
<td>$\int c(\nabla uu) \cdot v$</td>
</tr>
<tr>
<td>DiffusionIntegrator</td>
<td>$\int c \nabla u \cdot \nabla v$</td>
</tr>
<tr>
<td>VectorDiffusionIntegrator</td>
<td>$\int c \nabla u \cdot \nabla v$</td>
</tr>
</tbody>
</table>

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

Sparse Matrix-Vector Product (CuSparse)

\[ t_M = \frac{Data}{MaxBW} \quad \Rightarrow \quad \text{MaxThroughput} = \frac{MaxBW}{Data \text{ per DoF}} \]
Comparing native MFEM and the libCEED backend

### Throughput (MDoFs per second) vs Polynomial order

#### BP1 - NVIDIA V100

- **libCEED**
- **MFEM**

#### BP1 - AMD MI250X

- **libCEED**
- **MFEM**

#### BP3 - NVIDIA V100

- **libCEED**
- **MFEM**

#### BP3 - AMD MI250X

- **libCEED**
- **MFEM**
Comparing the assembly levels: AssemblyLevel::PARTIAL vs AssemblyLevel::NONE

BP1 - V100 - Scalar mass $q = p + 2$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE

BP2 - V100 - Vector mass $q = p + 2$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE

BP3 - V100 - Scalar diffusion $q = p + 2$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE

BP4 - V100 - Vector diffusion $q = p + 2$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE

BP5 - V100 - Scalar diffusion $q = p + 1$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE

BP6 - V100 - Vector diffusion $q = p + 1$

Throughput (DoFs per second)

Polynomial order

0 1 2 3 4 5 6

PARTIAL
NONE
Performance on simplicies and mixed-meshes

- **BP1 - MI250X - Hex mesh**
- **BP1 - MI250X - Tet mesh**
- **BP1 - MI250X - Mixed mesh**
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 $H(\text{div})$ and $H(\text{curl})$ (non-tensor only),
- Add support for discontinuous Galerkin methods,
- Add support for sparse-matrix assembly through libCEED.
Disclaimer: This document was prepared as an account of work sponsor by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees make any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and options of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.