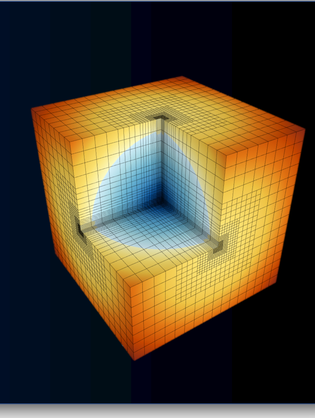
# CUDA Implementation of Matrix-Free Finite Element Operators

Steven Roberts[1] and Jean-Sylvain Camier[2]

1. Virginia Tech Department of Computer Science
2. Lawrence Livermore National Lab Center for Applied Scientific Computer



Abstract: libCEED is an open-source library developed by the Center for Efficient Exascale Discretizations (CEED) for the creation and evaluation of finite element operators. Instead of using a sparse matrix to represent these operators, it uses a technique called partial assembly to perform the evaluations on the fly, which is often significantly more efficient. In order for libCEED to support future exascale simulations, support for a wide range of computing platforms is needed. This summer, I developed a CUDA backend for libCEED which runs on NVIDIA GPUs. The project involved adapting the existing framework to better accommodate GPU computing and optimizing kernels for finite element operations. The new CUDA backend shows a significant speedup over other backends.

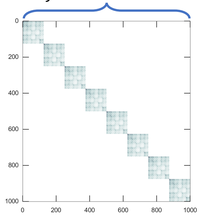
## Background

libCEED Goal: Create a lightweight C library for finite element operations that can be used at exascale

Motivation: Partial differential equations are the mathematical backbone to many physical simulations, and the finite element method is an effective and increasingly popular method for solving them

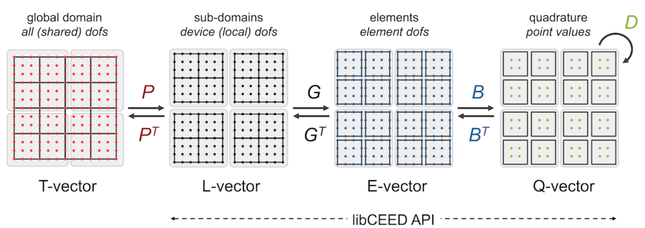
Techniques:
Using improved representations

$$-\nabla^2 u + \lambda u = f \Longrightarrow S^e u^e + \lambda M^e u^e = f^e$$




- Sparse matrices are a suboptimal method of representing finite element operators
- Use partial assembly to algebraically decompose operators
- Optimal memory and near-optimal floating point operations

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




| | libCEED Backends | Supported Hardware |
|---|---|---|
| ✓ | Reference | CPUs |
| ✓ | OCCA | CPUs and GPUs |
| ✓ | Magma | CPUs and GPUs |
| ✗ | CUDA | NVIDIA GPUs |

## Methodology

Project Goal: Develop an optimized CUDA backend for libCEED

Motivation:

- Operations are easy to parallelize since each element in a mesh can be processed independently
- Linear algebra operations (tensor contractions, inner products, etc.) are well-suited to GPUs
- More specialized optimizations than OCCA and Magma backends which also support GPUs

**Directly convert CPU backend into CUDA backend**

- Most interfaces force serial execution
- Many expensive memory transfers to and from GPU required
- Poor GPU utilization and performance

**Update interface to support "vectorization"**

- Better parallelism and fewer memory transfers
- Basis kernel bottlenecked by global memory access

**Improve memory accessing**

- Remove unnecessary memory transfers
- Utilizing shared memory to reduce global memory access

**Update tests and examples**

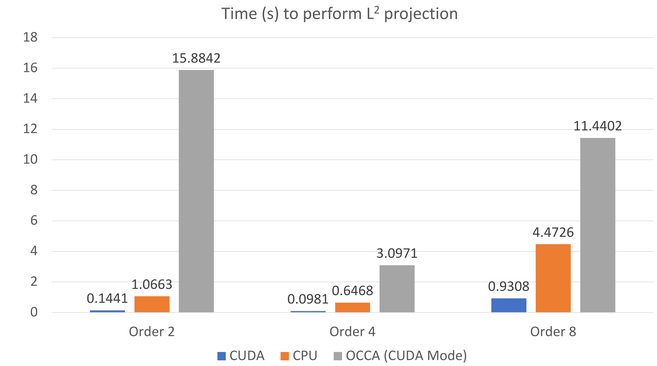
- Find and fix bugs
- Ensure accuracy
- Use examples for profiling

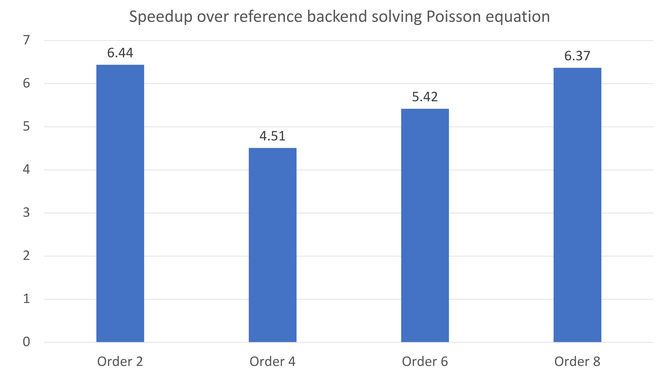
## Using libCEED

- The code is available at `github.com/CEED/libCEED`
- See `ceed.exascaleproject.org/libceed` for more information

## Results

- Performance tests were run on an NVIDIA Tesla P100-SXM2-16GB on Ray
- In all examples, the CUDA backend significantly outperforms all other backends and scales well at high orders
- Performance improvements would increase further if linear solves could be completed on GPU
- Profiling reveals high compute utilization but room for improvement in terms of memory bandwidth efficiency




## Acknowledgements

I would also like to thank Tzanio Kolev, Jeremy Thompson, Jed Brown, and the other developers of libCEED for their guidance in creating the CUDA backend and integrating it into the project.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-755562