Summary of MFEM: A Modular Finite Element Methods Library

Prepared by: Emir Fatih Demir (2377885)

March 16, 2024

Abstract

The library providing a comprehensive suite for facilitating high-order finite element method simulations is known as the Modular Finite Element Method (MFEM). The design prioritizes flexibility, scalability, and modularity, allowing for the incorporation of arbitrary high-order elements, meshes, and an extensive range of discretization techniques. The principal objective of this software is to empower application scientists to utilize state-of-the-art algorithms for discretizations, linear solvers, and high-order finite element meshing. Furthermore, MFEM facilitates the rapid development of prototypes and investigation of novel algorithms in unstructured mesh contexts that are highly generalized, including parallel and GPU-accelerated environments. This overview underscores the primary characteristics, foundational algorithms, and practical implementations of the library, thereby demonstrating its importance in propelling numerical methods and computational science inquiry forward. [1]

1 Introduction to MFEM

The document provides a comprehensive examination of the Modular Finite Element Methods (MFEM) library, emphasizing its functionalities, architecture, and scientific computing implementations. MFEM is a robust, open-source C++ library that has been specifically developed to facilitate high-order finite element analysis in a diverse array of applications. It facilitates the construction of intricate simulations in the fields of computational science and engineering through the provision of discretization, high-order meshing, and linear solver tools. Support for a variety of geometry types, discretizations based on finite elements, parallel processing, and GPU acceleration are all essential features. The library's modular design, scalability, and efficacy distinguish it as an exceptional choice for research endeavors as well as practical implementations. MFEM facilitates the solution of partial differential equations (PDEs) by operating efficiently on unstructured grids and supporting a wide range of discretization approaches. Its compatibility with a wide variety of external numerical libraries expands the functionality and applicability of the system. Since its inception at

Lawrence Livermore National Laboratory for physics and engineering applications, MFEM has expanded to include contributions from a worldwide developer and user community, as evidenced by its active development on GitHub. The design of the library prioritizes portability and usability in diverse computing environments, facilitating a smooth transition from serial to parallel computing contexts.

The library possesses extensive capabilities in managing a wide range of mathematical expressions. Consider, for example, the Poisson problem that was resolved using MFEM:

$$-\Delta u = f \quad \text{in } \Omega \tag{1}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{2}$$

By discretizing and solving these equations over arbitrary high-order meshes, MFEM demonstrates its prowess in addressing intricate mathematical challenges.

The visualization capabilities of MFEM are exemplified via a range of instances, including solution visualizations and mesh representations.

The following table summarizes the key features of MFEM:

Feature	Description
High-order elements	Supports arbitrary high-order elements
Scalability	Efficiently scales on HPC architectures
Modularity	Easily extendable for new algorithms
GPU acceleration	Supports CUDA, OCCA, RAJA, and OpenMP

Table 1: Key features of MFEM.

2 Finite Element Abstractions

MFEM offers an extensive collection of finite element abstractions, including:

The software provides support for conforming, non-conforming, NURBS, and parallel meshes, which greatly simplifies a variety of finite element analyses. Finite Element Discretization: Provides comprehensive assistance for finite element spaces, encompassing 2D and 3D versions of the complete high-order de Rham complex, as well as a multitude of bilinear, linear, and nonlinear forms. The finite element operators course covers linear systems, discretization methods, operator decomposition, and high-order partial assembly in order to facilitate computations in an efficient manner. The Finite Element Method (FEM) is a robust discretization technique that approximates the solutions of numerous partial differential equations (PDEs) using general unstructured grids. MFEM is a software library designed for finite elements that is open-source, lightweight, modular, and scalable. It supports a wide range of discretization approaches,

allows for the use of arbitrary high-order finite element meshes and spaces, and prioritizes usability, portability, and high-performance computing (HPC) efficiency.

To illustrate some of the functionality of MFEM, we consider the model Poisson problem with homogeneous boundary conditions: Find a function $u : \Omega \to R$ such that

$$-\Delta u = f \quad \text{in} \quad \Omega, \tag{3}$$

with

$$u = 0 \quad \text{on} \quad \Gamma, \tag{4}$$

where $\Omega \subset R^d$ is the domain of interest, Γ is its boundary, and $f : \Omega \to R$ is the given source.

2.1 Finite Element Discretization

The solution to this problem lies in the infinite-dimensional space of admissible solutions $V = \{v \in H^1(\Omega), v = 0 \text{ on } \Gamma\}$. To discretize, we begin by defining a mesh of the physical domain Ω, represented in MFEM using a *Mesh* object. Once the mesh is given, we define a finite dimensional subspace $V_h \subset V$, represented in MFEM by *FiniteElementSpace*. The approximate solution $u_h \in V_h$ is found by solving the corresponding finite element problem:

$$\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f v_h \quad \forall v_h \in V_h, \tag{5}$$

which can be equivalently written as $a(u_h, v_h) = l(v_h)$ for all $v_h \in V_h$, where the bilinear form $a(\cdot, \cdot)$ and linear form $l(\cdot)$ are defined by the integrals.

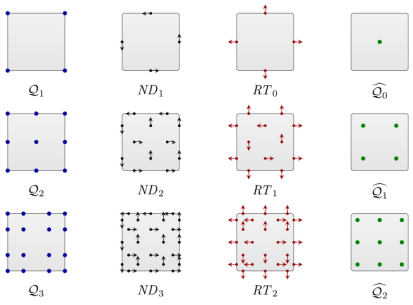

Figure 1: Linear, quadratic and cubic H1 finite elements and their respective H(curl), H(div) and L2 counterparts in 2D. Note that the MFEM degrees of freedom for the Nedelec (ND) and Raviart–Thomas (RT) spaces are not integral moments, but dot products with specific vectors in specific points as shown above.

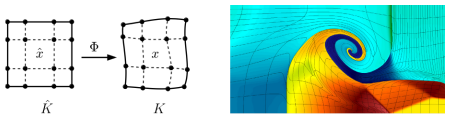

Figure 2: Left: The mapping from the reference element ˆK to a bi-cubic element K in physical space with high-order nodes shown as black dots. Right: Example of a highly deformed high-order mesh from a Lagrangian hydrodynamics simulation

2.2 Solution Process

The finite element problem is rewritten in terms of coefficients c_j as a discrete system of linear equations $Ax = b$. The *BilinearForm* and *LinearForm* objects in MFEM represent the matrix A and vector b, respectively. After solving this linear system, the resulting vector is used to define a *GridFunction* representing the discrete solution $u_h \in V_h$.

This illustration describes the fundamental MFEM classes and concepts required to solve a Poisson problem. Subsequent sections elaborate on additional functionalities of MFEM, such as parallelization, mesh adaptability, additional discretization techniques, and GPU acceleration.

3 Meshes

3.1 Conforming meshes

The topological (connectivity) and geometric (coordinates) information of a conforming, unstructured mesh in MFEM is used to describe its structure.

Vertices, elements, and boundary element listings dominate the topological component. Elements are specified by their form (triangles or tetrahedrons), a subdomain or boundary property, and vertex indices to the mesh's vertices. Boundary elements are defined similarly but assume one less dimension than normal elements. From primary data, edges and faces are inferred.

The mesh's arrangement may be shown geometrically using the direct coordinates of all vertices or an MFEM GridFunction, or nodes. This second technique, which works beyond linear meshes, uses the GridFunction class to define finite element functions/solutions. The finite element formulation uses (a) basis functions to link each reference element to its physical counterpart and (b) coefficients (nodal coordinates or degrees of freedom) to change these basis functions. Nodes are linked to vertices, edges, faces, and element interiors in this geometric nodal description.

Each element's shape is established by mapping (ϕ) from a standard reference element to the mesh element using basis functions (usually polynomials) and local nodal coordinates. The ElementTransformation class in MFEM computes physical coordinates, the Jacobian matrix, and integration weights needed to switch from the reference to the actual elements during computations. These

reference points are usually specified using the IntegrationPoint class for quadrature integrations.

After distinguishing between the mesh's reference and real spaces, MFEM can smoothly handle surface meshes with reference and spatial dimensions of 2 and 3.

Given the mapping $\Phi : \hat{x} \to x$, the transformation is defined as:

$$x(\hat{x}) = \Phi(\hat{x}) = \sum_{i=1}^{N} x_{K,i} w_i(\hat{x}). \tag{6}$$

Here, $x_{K,i}$ represents the nodal coordinates, and $w_i(\hat{x})$ are the basis functions evaluated at $\hat{x}$.

3.2 Non-conforming meshes

Hanging nodes indicate non-conforming meshes, which are conforming meshes with extra vertice limitations. Each constrained vertex in a linear mesh is a convex combination of parent vertices with possible constraints. The unique resolution of these relationships makes every constrained vertex a linear combination of unrestricted vertices.

These non-conforming meshes usually occur while locally refining quadrilateral and hexahedral meshes, when refined components share edges or faces with unrefined elements. Constrained vertices on the shared object align with its vertices to handle refinement propagation. Refinement should not create gaps or overlaps, keeping the mesh "watertight."

In high-order curved meshes or high-order finite element spaces applied to linear non-conforming meshes, restricted vertices become constrained degrees of freedom. The major goal is to avoid mesh gaps and overlaps and keep the limited, non-conforming finite element space a valid subspace of the continuous space it represents, such as H1 or H(div).

Non-conforming meshes are represented as conforming meshes with linear restrictions on specific nodes in MFEM. The Mesh class defines the mesh's conforming topology, while the nodal GridFunction imposes restrictions with unconstrained and limited degrees of freedom. Mesh uses a reference to an NCMesh object to control mesh non-conformity. This class records the refinement hierarchy, including parent-child connections for non-conforming edges and faces, recording the mesh's present state as leaves.

NCMesh can improve quadrilateral and hexahedral meshes isotropically and anisotropically. It supports a full level of hanging nodes and limitless edge and face improvements.

3.3 NURBS meshes

NURBS are crucial in geometric modeling for their exact conic section representation. Their use in IsoGeometric Analysis (IGA) to discretize PDEs has grown

in recent years. NURBS meshes and discrete spaces, like high-order polynomial situations, are quadrilateral (2D) or hexahedral (3D) meshes with basis functions as tensor products of 1D NURBS basis functions. Control points, or nodal degrees of freedom, are not confined to edges, faces, or vertices, unlike NURBS. Instead, control points may affect many layers owing to NURBS basis functions supporting blocks of (k+2)×(k+2) (2D) and (k+2)×(k+2)× (3D) elements, where k represents NURBS space continuity.

The NURBSExtension class in the MFEM library manages NURBS meshes, mapping components to their degrees of freedom. Users interact with NURBS meshes using the Mesh class, which represents quadrilateral or hexahedral components and links to a NURBSExtension object and a nodal GridFunction describing basis functions and control points. Many of MFEM's examples support NURBS meshes and IGA discretizations. From version 3.4, MFEM supports variable-order NURBS, with demos in miniapps/nurbs.

3.4 Parallel meshes

The ParMesh class, which extends Mesh, distributes the parallel mesh over MPI jobs in MFEM. Though faces, edges, and vertices may be shared across processors, each global mesh element has a unique MPI rank.

Beginning a ParMesh requires a serial Mesh object and a partitioning array that assigns MPI ranks to mesh elements. Meshing elements are treated as graph vertices and their connected entities as graph edges using the METIS toolkit, making partitioning fast.

Processor groups occur when shared mesh entities are partitioned and connected to processors. These groups have a master processor, usually the lowest tier, to sustain mesh connectivity. The shared entities are explicitly stored with local vertex indices on each processor to guarantee a consistent description across MPI activities. This rigorous structure keeps the mesh's description constant across jobs, notably mesh refining.

Parallel non-conforming meshes use the ParMesh class and a ParNCMesh class, which extends NCMesh. Unlike the conforming mesh scenario, partitioning may employ space-filling curves instead of METIS. ParMesh is linked to a ParNURBSExtension class, developed from NURBSExtension, to handle parallel NURBS meshes, however MFEM does not enable parallel refining. MFEM can easily handle diverse parallel mesh architectures while maintaining processor consistency and communication efficiency using this architecture.

4 High-Performance Computing

MFEM is intended for environments with high-performance computation, supporting:

Operators and parallel mesh spaces for distributed computing. GPU acceleration and scalable linear solvers for high-performance applications. Adaptivity of finite elements, including adaptive mesh refinement for conforming and

non-conforming meshes, and mesh optimization to ensure dynamic simulation fidelity. MFEM manages large-scale parallelism via the Message Passing Interface (MPI) by incorporating a layer that optimizes the reutilization of serial code. This is accomplished by subclassing serial classes that implement parallel operations and, on occasion, by overriding code segments with virtual functions. MFEM partitions the problem domain into K segments for K MPI tasks, with the objective of performing local processing on each segment. In each MPI task, the parallel mesh object, ParMesh, functions as a conventional Mesh object while incorporating supplementary information regarding geometric entities that are shared with other processors, such as faces, edges, and vertices. In a similar fashion, ParFiniteElementSpace operates as a standard FiniteElementSpace supplemented with communication group-organized descriptions of shared degrees of freedom. One of the fundamental roles of the parallel finite element space is to facilitate adaptive mesh refinement via the GetProlongationMatrix() method and to supply the prolongation matrix P for parallel assembly. Parallel grid functions, referred to as ParGridFunction, operate similarly to standard GridFunction objects with the ability to toggle between L-vector and T-vector levels. Functionality is improved via methods such as ParallelAverage and Distribute.

In order to enhance performance and minimize MPI communication, the design guarantees that only essential data is transferred between neighboring nodes in the mesh, with this exchange ideally occurring concurrently with computational tasks. In the concluding stages of assembly, a parallel PtALP triple matrix product is implemented, which is carried out via PETSc routines or the hypre library, depending on the particular operator type. By simplifying the process of converting serial MFEM-based codes to scalable parallel versions, this subclassing strategy enables the prefixing of finite element variables with 'Par'.

By explicitly computing and storing the parallel matrices in hypre's ParCSR format, MFEM provides access to efficient parallel linear algebra routines. By integrating with hypre, MFEM is able to utilize the algebraic multigrid (AMG) preconditioner of hypre, which has the capability to handle millions of parallel tasks. In addition, discretization-enhanced AMG methods, such as the auxiliary-space Maxwell solver (AMS), which is specifically engineered to handle Maxwell problems discretized with Nedelec H(curl)-conforming elements, are supported by MFEM.

The compatibility of MFEM with a range of additional solvers and preconditioners, including hypre and PETSc, enables the implementation of techniques such as Balancing Domain Decomposition by Constraints (BDDC) to address distinct finite element discretizations. As high-order methods can increase the computational complexity of matrix assembly, MFEM investigates low-order refined preconditioning and matrix-free preconditioners for high-order operators, including h- and p-multigrid methods.

Support for GPU acceleration and numerous programming models was added in Version 4.0, which embraced performance portability by incorporating support for multiple backends. This methodology enables MFEM to dynamically respond to changes in the computational environment without being constrained

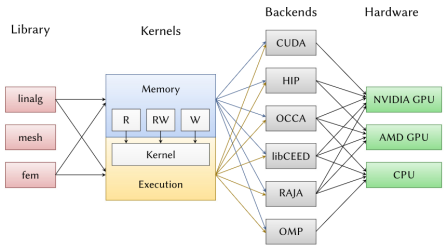

Figure 3: Diagram of MFEM's modular design for accelerator support, combining flexible memory management with runtime-selectable backends for executing key finite element and linear algebra kernels.

by a particular technology, providing versatility in the implementation of hardware accelerators and programming libraries such as CUDA, OCCA, libCEED, RAJA, and OpenMP.

5 Finite Element Adaptivity

MFEM offers extensive assistance in managing the adaptivity of both parallel and serial finite elements in the presence of high-order unstructured structures. This includes mesh optimization via node movement (r-adaptivity), nonconforming adaptive mesh refinement for quadrilateral and hexahedral meshes (non-conforming h-adaptivity), and local conforming mesh refinement for triangular and tetrahedral meshes (h-adaptivity). An attribute that sets MFEM apart is its unified methodology for local refinement, which applies to both simplex and tensor-product elements. These functionalities are elaborated upon and improved in the following sections via integration with the parallel unstructured mesh infrastructure (PUMI) of RPI.

MFEM automates for the user the process by which locally enforced refinements may need to be extended to adjacent processors in parallel scenarios. For example, when refining tetrahedral meshes, the vertices are rearranged in a manner that gives priority to the longest edge, which is subsequently divided in half. The process ensures consistency, even when neighboring tetrahedra are involved, by performing a global classification of edges according to length. The refinement process comprises several phases, one of which is "green refinement," which guarantees that elements are bisected only when required and do not require recurrent refinement. This system ensures that face refinements are consistent across all processors, enabling parallel refinement to occur without requiring interprocessor communication.

Particularly applicable to high-order applications that demand adaptive refinement on unstructured meshes, MFEM addresses the difficulties associated with irregular meshes and dangling nodes in non-conforming mesh refinement. This is accomplished by utilizing software abstractions and algorithms that oversee parallel non-conforming meshes. These methods enable the incorporation of

complex, anisotropic 3D meshes with different degrees of refinement and support the entire spectrum of finite element spaces at high orders. To maintain these updates, MFEM employs a tree-based data structure that has been optimized for memory efficiency.

The adaptive mesh refinement (AMR) methodology of MFEM is based on a variational restriction strategy. This involves developing a geometry that satisfies the continuity criteria for non-conforming interfaces in the particular finite element space in question, such as H1 or H(curl). In order to guarantee the conformity of a mesh, function values are interpolated from master faces or edges to subordinate ones; this limits the degrees of freedom (DOFs) of specific finite elements. By defining the relationship between all DOFs, the global interpolation matrix facilitates the formulation of a conforming AMR linear system. By handling these limitations automatically, MFEM simplifies the procedure of determining the actual degrees of freedom and retrieving the entire set of DOFs, including those that are constrained.

In general, the dependable adaptivity support of MFEM, in conjunction with its effective management of non-conforming meshes and seamless integration with external mesh infrastructure such as PUMI, establishes a robust framework for conducting precise finite element analysis in a diverse array of applications.

Table 2: Performance results with MFEM-4.0: Poisson problem (Example 1), 200 conjugate gradient iterations using partial assembly, 2D, 1.3M dofs, GV100, sm_70, CUDA 10.1, Intel Xeon Gold 6130@2.1 GHz. The best performing backends in each category (GPU, multicore, and CPU) are shown in bold.

	p = 1	**p = 2**	**p = 4**	**p = 8**
GPU				
OCCA-CUDA	0.52	0.31	0.20	0.19
RAJA-CUDA	0.38	0.30	0.28	0.45
CUDA	0.36	0.26	0.17	0.15
CEED-CUDA	**0.19**	**0.15**	**0.12**	**0.12**
Multicore				
OCCA-OMP	3.34	2.41	2.13	1.95
RAJA-OMP	3.32	2.45	2.10	1.87
OMP	3.30	2.46	2.10	1.86
MPI	**2.72**	**1.66**	**1.45**	**1.44**
CPU				
OCCA-CPU	21.05	15.77	14.23	14.53
RAJA-CPU	45.42	16.53	14.22	14.88
CPU	25.18	16.11	13.73	14.45
CEED-AVX	43.04	18.16	**11.20**	**8.53**
CEED-XSMM	**53.80**	**20.13**	10.73	7.72

6 Applications and Development

The MFEM library comprises an extensive collection of illustrative applications that effectively illustrate a wide range of its capabilities and illustrate the discretization of diverse PDEs using finite elements. These instances function as a preliminary manual to the functionalities of MFEM in straightforward situations. Every instance is comprehensively documented and presented in both serial and parallel formats, demonstrating the straightforwardness of migrating to parallel computation and utilizing hypre solvers and preconditioners. There are also specialized illustrations of integration with additional software programs, including PETSc, SUNDIALS, and PUMI. Through command-line arguments, users are able to investigate various mesh types and computational options; the '–help' option provides information on available features. GLVis can be utilized to visualize the results obtained from these examples; additional resources and tutorials are available on the MFEM website.

The numbered example codes—ex1 through ex21—are designed to progressively acquaint users with the interface of MFEM as their complexity increases. Beginning with simplified examples is recommended for novice users. The initial example in the library, denoted as ex1, employs H1 elements to solve the Laplace equation. Subsequent examples, including ex6, ex8, and ex14, investigate various formulations of the Poisson problem, such as discontinuous Galerkin (DG), adaptive mesh refinement (AMR), and discontinuous Petrov-Galerkin (DPG) techniques. Illustrative instances ex2 and ex17 utilize the Galerkin and DG methods, respectively, to tackle linear elasticity issues. In contrast, example ex10 explores nonlinear elasticity by employing a Newton solver. In petsc/ex10p, integration with the nonlinear solvers of PETSc is illustrated through the utilization of a Newton Krylov approach devoid of Jacobian terms. Ex3 and ex13 discuss solutions to Maxwell's equations utilizing H(curl) elements, while ex7 illustrates the application of surface meshes in three-dimensional spaces. Time-dependent simulations are examined in instances ex9, ex10, ex16, and ex17. For individuals interested in the SUNDIALS and PETSc ODE solvers, references to sundials/ex9p and petsc/ex9p are provided. The series culminates in ex11, ex12, and ex13, which are devoted to eigenvalue computations and frequency domain issues, respectively.

MFEM also comprises electromagnetic miniapps that serve as rudimentary prototypes for practical implementations. The Volta miniapp utilizes source terms and modifiable boundary conditions to tackle electrostatic issues. The Tesla miniapp effectively simulates magnetostatic phenomena by incorporating distinct boundary conditions obtained from surface currents and accommodating volumetric inputs such as permanent magnets or current densities. A "divergence cleaning" procedure is also incorporated to guarantee solenoidal vector fields. The Maxwell miniapp is designed to simulate the propagation of electromagnetic waves while accommodating a variety of boundary conditions and sources and conserving energy via symplectic time-integration. Furthermore, the Joule miniapp serves as an illustration of a multi-physics application through the coupling of electromagnetic and thermal conduction equations, which are

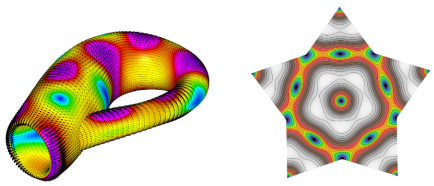

Figure 4: Left: Solution of a Maxwell problem on a Klein bottle surface with ex3; mesh generated with the klein-bottle miniapp in miniapps/meshing. Right: An electromagnetic eigenmode of a star-shaped domain computed with 3rd order finite elements computed with ex13p.

utilized to simulate Joule heating in conductors.

The Laghos miniapp, which is an abbreviation for Lagrangian high-order solver, utilizes a Lagrangian framework to simulate the dynamics of compressible, inviscid gases. It emphasizes the advantages of employing high-order curved meshes to model significant deformations and offers the flexibility to optimize memory and computational resources through full or partial assembly. Laghos facilitates the development of more intricate multi-physics simulations, as demonstrated by the BLAST project. This endeavor expands Laghos' functionalities to encompass solid mechanics, multi-material flows, mesh remapping, and more, thereby serving an extensive range of hydrodynamic systems and fusion dynamics research.

7 Conclusion

The architecture of MFEM incorporates an extensive range of finite element abstractions, such as robust mesh and operator support, finite element discretization, and comprehensive mesh support. Serial and parallel finite element adaptivity on high-order unstructured meshes are supported, along with mesh optimization and local conforming and non-conforming adaptive mesh refinement. Usability and portability across a variety of computing environments—from desktops to high-performance computing systems—are emphasized by the library.

Support for the entire high-order de Rham complex, which is crucial for numerous scientific and engineering applications, is one of the distinguishing characteristics of MFEM. This incorporates 2D and 3D support for NURBS, H1-conforming, discontinuous (L2), H(div)-conforming, and H(curl)-conforming finite element spaces. In addition, rapid prototyping of finite element discretizations is facilitated by MFEM, which provides a finite element toolbox comparable to that of MATLAB in linear algebra.

Modern paradigms of high-performance computation, such as GPU acceleration and parallel meshes, are incorporated into the MFEM design. The software incorporates sophisticated methods such as partial assembly and sum factorization in order to decrease memory consumption and computational expenses.

This is especially advantageous for high-order finite element methods. Moreover, by integrating with external numerical libraries and visualization tools, MFEM expands its functionality and solves a more extensive variety of problems.

Lawrence Livermore National Laboratory (LLNL) is the originator of the library, which is presently under the active maintenance of a worldwide community of developers and consumers. The library's extensive documentation and open-source nature foster active participation, promoting contributions and soliciting feedback.

In summary, MFEM signifies a substantial progression within the domain of finite element analysis by providing an adaptable and effective framework for the construction and evaluation of novel algorithms in parallel, high-order, and intricate computational environments. Its emphasis on high-performance computing, modular construction, and extensive capability set render it an invaluable instrument for engineers and scientists engaged in a vast array of computational science and engineering challenges. [1]

References

[1] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. Mfem: A modular finite element methods library. *Computers Mathematics with Applications*, 81:42–74, 2021. Development and Application of Open-source Software for Problems with Numerical PDEs.