ECP Milestone Report

Identify initial kernels, bake-off problems (benchmarks) and miniapps

WBS 1.2.5.3.04, Milestone CEED-MS6

Veselin Dobrev
Jack Dongarra
Jed Brown
Paul Fischer
Azzam Haidar
Ian Karlin
Tzanio Kolev
Misun Min
Tim Moon
Thilina Ratnayaka
Stanimire Tomov
Vladimir Tomov

July 6, 2017

ECP Milestone Report
Identify initial kernels, bake-off problems (benchmarks) and miniapps
WBS 1.2.5.3.04, Milestone CEED-MS6

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

July 6, 2017

ECP Milestone Report
Identify initial kernels, bake-off problems (benchmarks) and miniapps
WBS 1.2.5.3.04, Milestone CEED-MS6

Approvals

Submitted by:

Tzanio Kolev, LLNL CEED PI	Date

Approval:

Douglas B. Kothe, Oak Ridge National Laboratory Director, Applications Development Exascale Computing Project	Date

Revision Log

Version	Creation Date	Description	Approval Date
1.0	July 6, 2017	Original	

EXECUTIVE SUMMARY

The CEED co-design center is developing a number of kernels, bake-off / benchmark problems (BPs) and mini-applications (miniapps) that capture the unique requirements of high-order finite element algorithms and use them to influence standards, best practices and vendors to generate more portable, better performing code across diverse hardware designs. This is a main activity in CEED's Hardware, Applications and Finite Element thrusts.

CEED's miniapps, a.k.a. CEEDlings, are small, standalone applications - surrogates for sub-components of interest from the selected first wave of applications (the MARBL/LLNLApp and ExaSMR projects). The BPs are even smaller standalone programs that solve a simple physical problem, including discretization, linear and/or nonlinear solve, and potentially a relatively small number of time steps. A BP may be a proxy/surrogate for part of a miniapp or another problem of general interest.

In CEED, both miniapps and BPs will be used extensively in the project to evaluate and compare the performance of algorithms in specific contexts. The parts of the miniapps and BPs that play an important role in their overall performance will be identified and separated as standalone kernels. These kernels and their interaction inside BPs, miniapps and the actual applications, will be the main focus of our optimization efforts. All three standalone pieces of software (miniapps, BPs, and kernels) will serve as a basis for engaging and collaborating with hardware vendors and software technologies (ST) projects. These partners can start at the bottom of the hierarchy, e.g. with the kernels, and easily transition to BPs and miniapps as all three families will be produced in the same CEED environment.

In this milestone we identified the initial kernels, bake-off problems and miniapps and delivered software and documentation for them through the new CEED website, `http://ceed.exascaleproject.org`.

We specifically identified the Nekbone miniapp for the ExaSMR application and developed a new miniapp, Laghos, for the MARBL/LLNLApp application. We also formulated four initial benchmark problems (BP1-BP4) with clearly defined performance metrics and performed an initial bake-off comparison between the Nek5000 (spectral element) and MFEM (high-order finite element) technologies. As part of the milestone, we also engaged vendors (AMD) and ST projects (MPICH, STRUMPACK).

In this document we are reporting some details and results from these R&D activities, as well as additional project-wide activities performed in Q3 of FY17, including: making CEED software available on GitHub, developing the project website, and the preparation of CEED's first annual meeting.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The CEED co-design center is developing a number of kernels, bake-off / benchmark problems (BPs) and mini-applications (miniapps) that capture the unique requirements of high-order finite element algorithms and use them to influence standards, best practices and vendors to generate more portable, better performing code across diverse hardware designs. This is a main activity in CEED's Hardware, Applications and Finite Element thrusts.

CEED's miniapps, a.k.a. CEEDlings, are small, standalone applications - surrogates for sub-components of interest from the selected 1st wave of applications (the MARBL/LLNLApp and ExaSMR projects). The bake-off (or benchmark) problems are even smaller standalone programs that solve a simple physical problem, including discretization, linear and/or nonlinear solve, and potentially a relatively small number of time steps. A BP may be a proxy/surrogate for part of a miniapp or another problem of general interest.

In CEED, both miniapps and BPs will be used extensively in the project to evaluate and compare the performance of algorithms in specific contexts. The parts of the miniapps and BPs that play an important role in their overall performance will be identified and separated as standalone kernels. These kernels and their interaction inside BPs, miniapps and the actual applications, will be the main focus of our optimization efforts. All three standalone pieces of software (miniapps, BPs, and kernels) will serve as a basis for engaging and collaborating with hardware vendors and software technologies (STs). These partners can start at the bottom of the hierarchy, e.g. with the kernels, and easily transition to BPs and miniapps as all three families will be produced in the same CEED environment.

2. CEED MINIAPPS

CEED is developing a variety of miniapps encapsulating key physics and numerical kernels of high-order applications. The miniapps are designed to be used in a variety of co-design activities with ECP vendors, software technologies projects, as well as external partners. More information about the CEED miniapps can be found on CEED's website at `http://ceed.exascaleproject.org/miniapps`.

2.1 Nekbone

Nekbone is a lightweight subset of Nek5000 that is intended to mimic the essential computational complexity of Nek5000, relevant to large eddy simulation (LES) and direct numerical simulation (DNS) of turbulence in complex domains, in relatively few lines of code. It allows software and hardware developers to understand the basic structure and computational costs of Nek5000 over a broad spectrum of architectures ranging from software-based simulators running at one ten-thousandth the speed of current processors to exascale platforms running millions of times faster than single-core platforms. Nekbone has weak-scaled to 6 million MPI ranks on the Blue Gene/Q Sequoia at LLNL while Nek5000 has strong-scaled to over a million ranks on the Blue Gene/Q Mira at ANL. Nekbone provides flexibility to adapt new programming approaches for scalability and performance studies on a variety of platforms without having to understand all the features of Nek5000, currently supporting OpenACC/CUDA-based GPU variants.

It solves a standard Poisson equation using a conjugate gradient iteration with a simple or spectral element multigrid preconditioner on a block or linear geometry. In CEED, Nekbone serves as a proxy for a sub-component of the ExaSMR application. The Nekbone miniapp is available at `https://github.com/Nek5000/Nekbone` as well as at the CEED mirror `https://github.com/ceed/Nekbone`.

2.2 Laghos

Laghos (LAGrangian High-Order Solver) is a new miniapp developed in CEED that solves the time-dependent Euler equations of compressible gas dynamics in a moving Lagrangian frame using unstructured high-order finite element spatial discretization and explicit high-order time-stepping. In CEED, Laghos serves as a proxy for a sub-component of the MARBL/LLNLApp application.

Laghos captures the basic structure and user interface of many other compressible shock hydrocodes, including the BLAST code at LLNL. The miniapp is build on top of a general discretization library (MFEM),

thus separating the pointwise physics from finite element and meshing concerns. It exposes the principal computational kernels of explicit time-dependent shock-capturing compressible flow, including the FLOP-intensive definition of artificial viscosity at quadrature points. The implementation is based on the numerical algorithm from [1].

The computational effort of Laghos is focused on constructing and solving the following semi-discrete system of linear equations:

$$M_v \frac{dv}{dt} = -F \cdot 1\,, \quad M_e \frac{de}{dt} = F^T \cdot v\,. \tag{1}$$

The mass matrices (M_v and M_e) and the force matrix (F) have the form:

$$(M_v)_{ij} = \int \rho w_j w_i\,, \quad (M_e)_{ij} = \int \rho \phi_j \phi_i\,, \quad F_{ij} = \int (\sigma : \nabla w_i)\phi_j\,,$$

where w and ϕ are H^1 and L_2 basis functions, respectively, of arbitrary order. The stress tensor, σ, and density, ρ, are determined by quadrature point-based computations, which include definitions of artificial viscosity, length scales and time-step estimates. Currently, Laghos provides two methods for constructing and solving the linear system (1):

- *Full assembly*, where M_v, M_e and F are fully assembled and stored in compressed sparse row (CSR) format. The CSR mass matrices are assembled only once at the beginning of the computation, and they are applied numerous times for every linear system solve. The force matrix is reassembled at each time step, and the result is applied only twice for constructing the right-hand side of the system.
- *Partial assembly*, see Section 3.2, where instead of assembling the matrices, the miniapp only defines their zone-based actions. These actions are used to define the right-hand side of the system and find its solution by an iterative method. The application of the matrices depends on the quadrature point data, ρ and σ, which is constant-in-time for the mass matrices M_v and M_e, but changes in each time step for the force matrix F. As the local action is defined by utilizing the tensor structure of the finite element spaces, the amount of data storage, memory transfers, and FLOPs are lower, especially for higher orders.

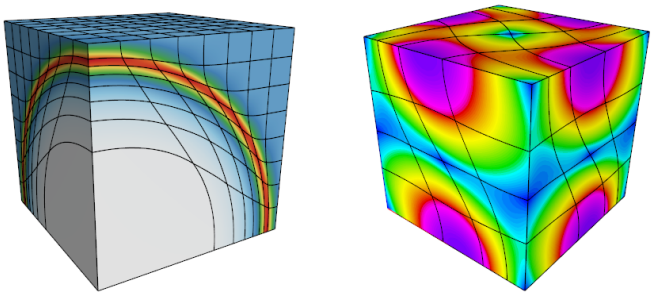


Figure 1: The Sedov blast (left) and the Taylor-Green vortex (right) problems in the Laghos miniapp.

The Laghos miniapp is available at `https://github.com/ceed/Laghos`. Coarse results from two of its test problems, demonstrating shock capturing and high-order deformed meshes can be seen in Figure 1.

2.3 Additional Miniapps

The CEED team is working on several additional miniapps, including the **NekCEM CEEDling** available at `https://github.com/NekCEM/NekCEM_CEEDling`, which is a NekCEM miniapp, solving the time-domain Maxwell equation for electromagnetic systems, and **Holmes**, which is an experimental testbed for multi-level parallel implementations of high-order finite element computations.

CEED team members are also actively involved in the **HPGMG-FE** benchmark, which is a Full Multigrid solver for a third order accurate finite element discretization of an elliptic operator on mapped grids designed to give a picture of the "performance spectrum" across a range of problem sizes and thus solution time.

Finally, the MFEM distribution includes 17 **MFEM Example Codes** that can be viewed as simple miniapps for model high-order physics.

See `http://ceed.exascaleproject.org/miniapps` for more details.

3. CEED BAKE-OFF PROBLEMS

A major goal of CEED is to develop a library that will facilitate efficient numerical simulation for broad range of applications on future high-performance computing (HPC) platforms. An important first step in this process is to identify optimal implementations of key kernels for high-order discretizations that will deliver high accuracy at low cost. Specifically, we are seeking the best implementations for a given architecture, the components that make them perform well, and software expressions that will allow these implementations to be leveraged on future platforms without change to the application-level software.

To identify candidate formulations, as described in the introduction, CEED team is conducting a series of performance tests (bake-off/benchmark problems), referred to as BPs here onwards, for several solver kernels. The initial tests are conducted using existing codes within the CEED project (e.g., Nek5000, MFEM) but the test specifications and results are established with the goal of allowing the technology comparison to be a continuing process for future codes and architectures. This section presents a description for the set of bake-off tests along with preliminary results for Nek5000 and MFEM.

3.1 Bake-off/Benchmark Problems Description

We note that the spirit of the bake-off is to solve problems that reflect actual use cases, rather than to solve the bake-off problem as fast as possible. Thus, optimizations that exploit specifics of the bake-off problem are deemed uninteresting and results from such optimizations must be discounted. Unlike an application (which is a particular instantiation of a given physics problem, using a given code, at a fixed resolution or error tolerance), a bake-off specification must span many decades of scale and thus be readily describable over such a large dynamic range. To simplify the specification, we prescribe the mesh topology as a tensor-product array of $E = E_x \times E_y \times E_z$ elements, but make no attempt to exploit this structure in either the solution strategy or the communication patterns. Instead, we use code suited to the fully unstructured cases that are encountered in practice.

The choice of a tensor-product array of elements is motivated by the need for scalability of the bake-off problem over a wide range of problem sizes. As our target problems are in fact unstructured domains, one is not allowed to exploit the global tensor-product structure in the domain. Thus, one must use general communication kernels and preconditioners as would be applied in the unstructured case. We further assume (and may later insist) that each element is deformed, which means that one must account for variable Jacobians, grid-transformation metrics, etc. Consequently, we include all cross terms in the derivative evaluations as if the geometry were fully deformed. (The cost can be lowered by a factor of two or more if the elements are undeformed bricks.)

Bake-off Problems. We define our first four bake-off problems, denoted as BPk, $k = 1, 2, 3, 4$.

- **BP1** Solve $B\underline{u} = \underline{f}$, where B is the mass matrix.
- **BP2** Solve the (block-diagonal) vector system, $B\underline{u}_i = \underline{f}_i$ $(i = 1, 2, 3)$ where B is as prescribed in BP1.
- **BP3** Solve $A\underline{u} = \underline{f}$, where A is the Poisson operator.
- **BP4** Solve the (block-diagonal) vector system, $A\underline{u}_i = \underline{f}_i$ $(i = 1, 2, 3)$ where A is as prescribed in BP3.

Bake-off Problem Details. The following items describe the details common to all BPs:

- Mesh: use a 3D box mesh with hexahedral elements.

- Boundary conditions (BCs): either no essential BCs, or essential BCs on the whole boundary.
- Solution space orders: $p = 1, 2, 3, \ldots, 8$, and optionally higher p.
- Quadrature: tensor product Gauss-Legendre (GL) with $q = p + 2$ points in each spatial dimension; the quadrature order is $2q - 1 = 2p + 3$.
- The cases of $q = 2$, for $p = 1$, and $q = 3$, for $p = 2$, are of interest as they provide a more favorable ratio of the total number of quadrature points to the total number of unknowns. Note that this ratio is smaller (and therefore, advantageous in terms of work per unknown) for larger p.
- Use nodal basis with $p + 1$ Gauss-Legendre-Lobatto (GLL) points in each spatial dimension.
- Consider mesh orders of $p_{\text{mesh}} = 1$, and/or $p_{\text{mesh}} = p$.
- Elements are assumed to be deformed, meaning that the local element-by-element evaluation cannot exploit simplifications arising from the absence of cross-terms in the Laplacian, etc.
- Use the QA/PA operator representation, see Sections 3.2 and 3.6.
- Meshes: consider meshes with $E = 2^s$ elements with $s \in \mathbb{N}$; for a given s, use a 3D Cartesian mesh with $2^{s_1} \times 2^{s_2} \times 2^{s_3}$ elements ($s_i \in \mathbb{N}$), where $\{s_i\}$ are uniquely determined by the conditions: $s_1 + s_2 + s_3 = s$ and $\lfloor s/3 \rfloor + 1 \geq s_1 \geq s_2 \geq s_3 \geq \lfloor s/3 \rfloor$.
- For example:
  - if $s = 15$, then $s_1 = s_2 = s_3 = 5$,
  - if $s = 16$, then $s_1 = 6$ and $s_2 = s_3 = 5$,
  - if $s = 17$, then $s_1 = s_2 = 6$ and $s_3 = 5$.
- Consider tests with 2^t processors, $0 \leq t \leq s$, and partition the mesh into $2^{t_1} \times 2^{t_2} \times 2^{t_3}$ uniform parts, where $\{t_i\}$ are derived from t the same way $\{s_i\}$ are derived from s. Using a partitioning of this type allows us to consider cases with a small number of elements per MPI rank - down to one element/rank.
- Alternative mesh partitioning algorithms are also acceptable.
- Consider runs with "large" number of processors and vary the number of mesh elements per MPI rank starting from 1 and gradually increasing to a number where performance saturation is observed. This suite of benchmarks thus captures both the strong-scale and weak-scale performance limits under the assumption that the underlying code is scalable.

3.2 Mathematical Formulations

The first set of test problems calls for parallel solution of a sequence of linear systems arising from spectral/finite element discretization of the (positive definite) Helmholtz equation,

$$-\nabla \cdot \mu \nabla u + \gamma u = f \text{ in } \Omega, \quad \nabla u \cdot \hat{\mathbf{n}} = 0 \text{ on } \partial\Omega, \tag{2}$$

where γ and μ may be non-negative functions of $\mathbf{x} \in \Omega$. Before proceeding with the details, we remark that the discretization of (2) leads to the symmetric positive definite (SPD) system

$$H\underline{u} := A\underline{u} + B\underline{u} = \underline{b}, \tag{3}$$

with the stiffness matrix A corresponding to the variable-coefficient Poisson operator in (2) and mass matrix B corresponding to the term involving γ. For the first bake-off problem (BP1), we take $\mu = 0$, $\gamma = 1$, so the problem reduces to solving $B\underline{u} = \underline{b}$. For the third problem (BP3), we take $\mu = 1$, $\gamma = 0$, corresponding to the solution of the discrete Poisson equation, $A\underline{u} = \underline{b}$. We assume homogeneous Neumann conditions for all cases. BP3 is singular in this case and the singularity is removed by constraining the solution to have zero mean. We further remark that one has several options when implementing quadrature for the variational forms

introduced below. The standard spectral element approach is to use equal-order Gauss-Lobatto-Legendre (GLL) bases and quadrature, in which case B is diagonal and trivially inverted. In some applications, however, it is advantageous to use higher-order quadrature and it was thus agreed among the bake-off participants to use Gauss-Legendre (GL) quadrature on $q = p + 2$ points.

The variational formulation for (2) is, *Find* $u \in X^p \subset \mathcal{H}^1$ *such that*

$$a(v,u) \; + \; (v,u)_\gamma \quad = \quad (v,f) \;\; \forall \; v \; \in \; X^p, \tag{4}$$

where, for sufficiently regular v, u, and f, we have:

$$(v,u)_\gamma \; := \; \int_\Omega v \, \gamma \, u \, d\mathbf{x}, \;\; a(v,u) \; := \; \int_\Omega \mu \nabla v \cdot \nabla u \, d\mathbf{x}, \;\; (v,f) \; := \; \int_\Omega v \, f \, d\mathbf{x}. \tag{5}$$

Following the standard finite/spectral element approach, we formally expand all functions in terms of basis functions, such as

$$u(\mathbf{x}) \; = \; \sum_{j=1}^{n} u_j \, \phi_j(\mathbf{x}), \qquad v(\mathbf{x}) \; = \; \sum_{i=1}^{n} v_i \, \phi_i(\mathbf{x}). \tag{6}$$

The coefficients $\{u_j\}$ and $\{v_i\}$ are the finite element degrees of freedom (DOFs). Insertion of (6) into (5) leads to the inner-products

$$a(v,u) \; = \; \underline{v}^T A \underline{u}, \;\; (v,u)_\gamma \; = \; \underline{v}^T B \underline{u}, \;\; (v,f) \; = \; \underline{v}^T \underline{b}. \tag{7}$$

Here, we have introduced the stiffness matrix, A, the (weighted) mass matrix, B, and the right-hand side, $\underline{b}$,

$$A_{ij} \; := \; a(\phi_i, \phi_j), \quad B_{ij} \; := \; (\phi_i, \phi_j)_\gamma, \qquad b_i \; := \; (\phi_i, f), \tag{8}$$

each defined for index sets $i, j \; \in \; \{1, \ldots, n\}$. The system to be solved is thus

$$H \underline{u} = \underline{b}, \quad \text{where} \quad H \; := \; A \; + \; B. \tag{9}$$

The system (9) denotes a generic Galerkin discretization of (4). The point of departure for the finite element (FE)/spectral element (SE) formulation is in the choice of basis functions, ϕ_i, and choice of quadrature rules for the inner products (5). We give a brief description here; more details are provided in [2].

We begin by noting that (9) will be solved iteratively, typically with some type of preconditioned Krylov-subspace-projection method such as conjugate gradients or GMRES. In this case, one requires only matrix-vector products of the form $\underline{w} = H\underline{p}$, and not explicit formation of H itself. This crucial observation allows one to avoid not only global assembly of H but also *local* formation of H, which, for an pth-order method on tensor-product (curvilinear brick) elements reduces the work from $O(p^6)$ to $O(p^4)$ and the storage from $O(p^6)$ to $O(p^3)$, as noted in the seminal paper of Orszag [3]. For reasons of efficiency, the local stiffness matrices are never formed. As with the mass matrix, one has the option of performing the quadrature on the $O(p)$ GLL points or interpolating to a finer set of q GL points.

Matrix-Free Formulation. With the preceding definitions, we now consider the fast tensor-product evaluations of the inner products introduced in (7). To begin, we assume $\Omega = \bigcup_{e=1}^{E} \Omega^e$, where the non-overlapping subdomains (elements) Ω^e are images of the reference domain, $\hat{\Omega} := [-1,1]^3$, given by

$$\mathbf{x}|_{\Omega^e} \; = \; \mathbf{x}^e(r,s,t) \quad = \quad \sum_{k=0}^{p} \sum_{j=0}^{p} \sum_{i=0}^{p} \mathbf{x}^e_{ijk} \, h_i(r) \, h_j(s) \, h_k(t), \quad [r,s,t] \; \in \; \hat{\Omega}, \tag{10}$$

where the h_is are assumed to be cardinal Lagrange polynomials based on the Gauss-Lobatto-Legendre (GLL) quadrature points, $\xi_j \in [-1,1]$, $j = 0, \ldots, p$. This choice of points yields relatively well-conditioned operators and affords the option of direct (GLL) quadrature, if desired. We expand u and v in the same basis as the geometry. Starting with the mass matrix, we have

$$\underline{v}^T B \underline{u} \quad := \quad \int_\Omega \gamma \, v \, u \, dV \; = \; \sum_{e=1}^{E} \int_{\Omega^e} \gamma \, v \, u \, dV \; = \; \sum_{e=1}^{E} (\underline{v}^e)^T B^e \underline{u}^e \; = \; \underline{v}_L^T B_L \underline{u}_L, \tag{11}$$

where B_L is the block-diagonal matrix comprising unassembled local mass matrices, B^e, $e = 1, \ldots, E$. Because u and v are both continuous, the global-to-local map implies existence of vectors $\underline{u}$ and $\underline{v}$ such that

$$\underline{v}_L^T B_L \underline{u}_L \;=\; \underline{v}^T Q^T B_L Q \underline{u} \;=\; \underline{v}^T B \underline{u}, \tag{12}$$

from which we deduce that the *assembled* mass matrix is $B = Q^T B_L Q$. Technically, there is usually one additional step in defining B, namely, restriction of u and v to functions that satisfy appropriate Dirichlet conditions [2]. The local mass matrices are derived by integrating expansions of the form $u^e_{ijk} = u|_{\mathbf{x}^e_{ijk}}$ for u and v over Ω^e, subject to the mapping (10). In the tensor-product notation, B^e has entries

$$B^e_{(ijk),(\hat{\imath}\hat{\jmath}\hat{k})} \;=\; \int_{\Omega^e} h_i(r)h_j(s)h_k(t)\; h_{\hat{\imath}}(r)h_{\hat{\jmath}}(s)h_{\hat{k}}(t)\; \gamma\, d\mathbf{x} \tag{13}$$

$$\;=\; \int_{\hat{\Omega}} h_i(r)h_j(s)h_k(t)\; h_{\hat{\imath}}(r)h_{\hat{\jmath}}(s)h_{\hat{k}}(t)\; \gamma^e(\mathbf{r})\mathcal{J}^e(\mathbf{r})\, d\mathbf{r}, \tag{14}$$

with $\mathcal{J}^e$ the Jacobian associated with the mapping (10). Unless γ^e and $\mathcal{J}^e$ are separable, no further simplification of (14) is possible without resorting to quadrature. The standard approach for the SE method [4, 5] is to use tensor-product quadrature on the order p GLL nodes, in which case B^e (and B) are diagonal. For any $(q+1)$-point one-dimensional quadrature rule with nodes $\hat{r}_l$ and weights ρ_l, define

$$B^e_{(ijk),(\hat{\imath}\hat{\jmath}\hat{k})} = \sum_{n=0}^{q}\sum_{m=0}^{q}\sum_{l=0}^{q} \rho_l\rho_m\rho_n\, h_i(\hat{r}_l)h_j(\hat{s}_m)h_k(\hat{t}_n)\; h_{\hat{\imath}}(\hat{r}_l)h_{\hat{\jmath}}(\hat{s}_m)h_{\hat{k}}(\hat{t}_n)\; \tilde{\gamma}^e_{lmn}\, \tilde{\mathcal{J}}^e_{lmn},$$

where $\tilde{\gamma}^e_{lmn}$ and $\tilde{\mathcal{J}}^e_{lmn}$ represent evaluations of γ^e and $\mathcal{J}^e$ on the quadrature points.

The derivation for the stiffness matrix parallels that of the mass matrix. One starts with the bilinear form

$$a(v,u) \;:=\; \int_\Omega \nabla v \cdot \nabla u\, dV \;=\; \sum_e^E \int_{\Omega^e e} \nabla v \cdot \nabla u\, dV,$$

and invokes function continuity to yield

$$a(v,u) \;:=\; \underline{v}_L^T A_L \underline{u}_L \;=\; \underline{v}^T Q^T A_L Q \underline{u} \;=\; \underline{v}^T \bar{A} \underline{u},$$

with A_L =block-diagonal(A^e) the *unassembled* matrix comprising the local stiffness matrices, A^e, $e = 1, \ldots, E$. Here $\bar{A}$ is the *assembled* stiffness matrix. We typically refer to $\bar{A}$ as the Neumann operator as it corresponds to the singular matrix associated with pure Neumann boundary conditions. The singularity can be addressed by projecting constant-mode out of each search direction in the Krylov-subspace projection method (e.g., $\underline{p} = \underline{p} - \underline{e}\,\underline{e}^T\underline{p}/(\underline{e}^T\underline{e})$, with $\underline{e}$ the vector of all 1s) or, in the case of Dirichlet conditions, defining the restricted matrix $A = RAR^T$, where R^T is the prolongation matrix that extends by zero any vector of values in the domain interior to include the boundary values. The system $A\underline{u} = \underline{b}$ corresponds to solving a problem with homogeneous Dirichlet conditions and the corresponding solution (including the zeros on the boundary) is given by $\bar{\underline{u}} = R^T\underline{u} = R^T A^{-1}\underline{b}$. In the case of Neumann conditions, R is the identity matrix and we assume that the singular mode is controlled by some means such as projection.

For reasons of efficiency, the local stiffness matrices are never formed. As with the mass matrix, one has the option of performing the quadrature on the $O(p)$ GLL points or interpolating to a finer set of M GL points. (For the constant-coefficient 1D case, the result is the same [5].) While there is clear potential for a gain in accuracy with over-integration for the mass matrix [6], it is not clear that similar accuracy gains are realized when solving the Poisson equation. Similarly, while stability considerations favor over-integration of the advection operator [7], improved quadrature does not qualitatively alter the spectrum of the diffusion operator, which remains positive definite whether one over-integrates or not.

3.3 Benchmark Repository

In additions to the bake-off problem specifications, we have also developed a user-friendly software repository that can be used to run the BPs, as well as lower-granularity high-order kernels and experimental tests on a

variety of HPC architectures. The CEED *benchmarks* repository is available at `https://github.com/CEED/benchmarks` as part of the CEED software suite, a collection of software benchmarks, miniapps, libraries and APIs for efficient exascale discretizations based on high-order finite element and spectral element methods. Some of its features are:

- Simple build system for the CEED software stack.
- Variety of machine configurations, with machine-specific compilers, job scheduler commands, etc.
- Number of kernels and benchmark tests, including implementations of the 4 CEED BPs for both MFEM and Nek5000.
- Unified post-processing and data visualization scripts.

The benchmarks repository is an easy way to evaluate different CEED technologies in a consistent manner across different hardware. The bake-off results reported in the following sections have been generated using these benchmarks tools. For more details on the CEED benchmarks see `http://ceed.exascaleproject.org/bps/`.

3.4 Bake-off Between Nek5000 and MFEM

In this section, we demonstrate bake-off results with preliminary test data for Nek5000 and MFEM.

The results are plotted as the rate-of-work per unit-resource, given by [DOFs × CG Iterations] / [compute nodes × seconds], versus the "scalar" size of the problem on each compute node, denoted as [Points per compute node].

The definition of *points* is "DOFs/vector dimension", so points and DOFs are the same for BP1 and BP3, but points = DOFs/3 for BP2 and BP4. (This way it is easier to compare BP1 and BP2 results on the same mesh.)

We measure the rate of work in DOFs-per-second (dps) or more frequently in millions-of-DOFs-per-second (MDOFs, or *megadofs*). Two of the principal metrics of interest are the peak rate of work per unit resource, $r_{\max}$, and the $n_{\frac{1}{2}}$—the local problem size on the node required to realize one-half of the peak rate of work per unit resource. Note that $n_{\frac{1}{2}}$ is defined in terms of DOFs, not points.

The importance of $n_{\frac{1}{2}}$, $r_{\max}$ and their scaled ratio $t_{\frac{1}{2}} = 2n_{\frac{1}{2}}/r_{\max}$ is discussed further in Section 3.5.

3.4.1 Results on Intel Xeon Phi (2nd Generation)

Bake-Off problems BP1 and BP2 were run on a 2nd generation Intel® Xeon Phi processor (Knights Landing, referred to as KNL here onwards). The KNL processor we used has 64 cores spread in 36 tiles arranged in a 2D mesh. A tile contains 2 cores and 2 vector processing units. 2 cores in a given tile share 16-way L2 cache of size 1 MB. Apart from a DDR4 memory of size 96 GB, the KNL also have an additional high bandwidth memory of size 16 GB known as MCDRAM (Multi-Channel DRAM). In the benchmark runs, MCDRAM used as a last level cache. All the benchmarks were performed under CentOS version 7.3.

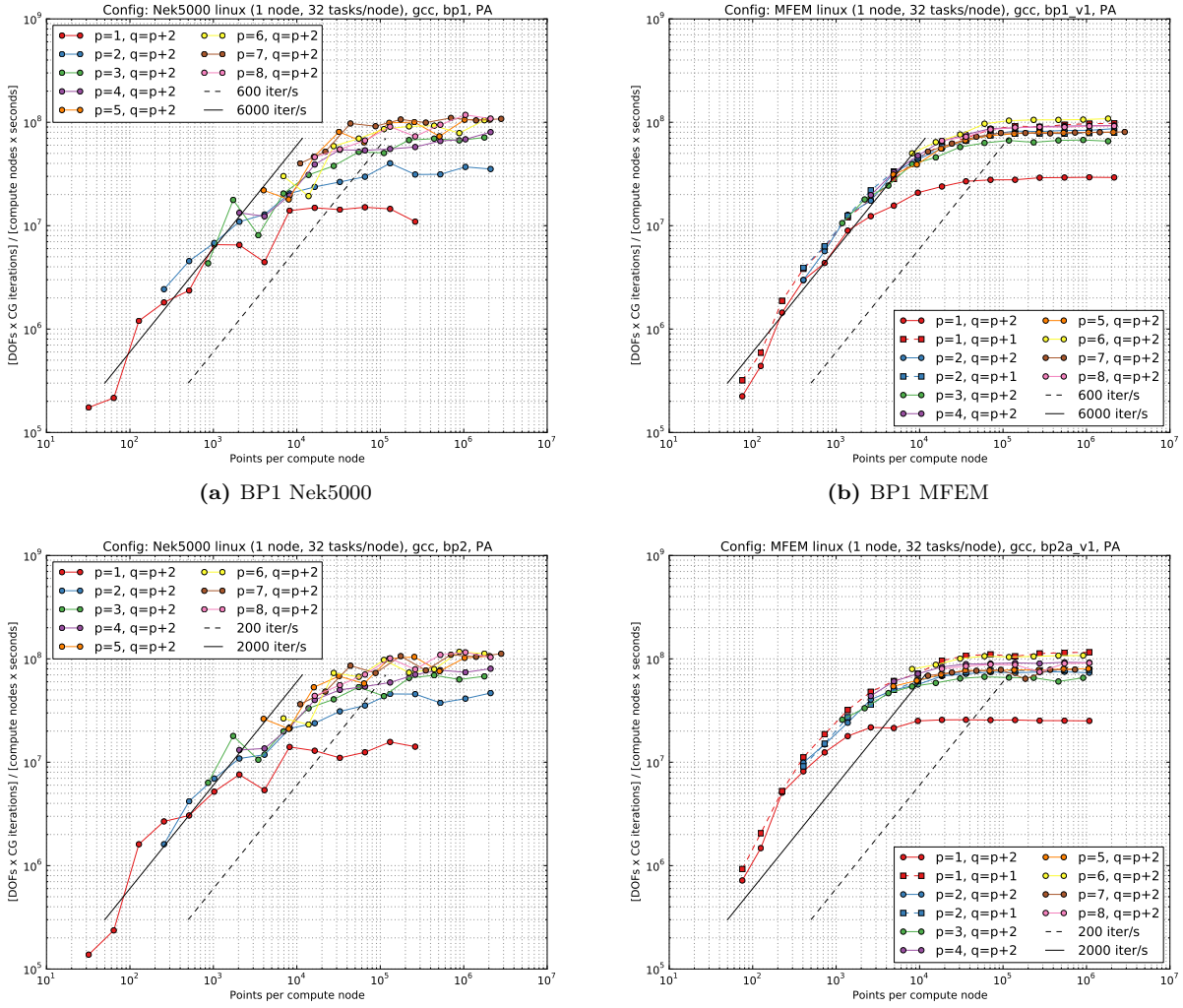
32 MPI ranks were used in this benchmarks instead of 64 to avoid disturbances to timing data due to OS processes. Each bake-off problem was compiled using the GCC (version 4.8.5) and the Intel (version 17.0) compilers. Figure 2 shows the performance of Nek5000 and MFEM for BP1 and BP2 in KNL using the GCC compiler. Figure 3 has the respective results for the Intel compiler.

Using the GCC compiler in BP1, for Nek5000 we observe an r_{max} of 100 MDOFs with $n_{\frac{1}{2}}$ of 20K for corresponding $t_{\frac{1}{2}}$ of 0.400ms. Using the GCC compiler in BP2, for Nek5000 we observe an r_{max} of 100 MDOFs with $n_{\frac{1}{2}}$ of 50K for corresponding $t_{\frac{1}{2}}$ of 1.000ms. Using the GCC compiler in BP1, for MFEM we observe an r_{max} of 100 MDOFs with $n_{\frac{1}{2}}$ of 8K for corresponding $t_{\frac{1}{2}}$ of 0.160ms. Using the GCC compiler in BP2, for MFEM we observe an r_{max} of 100 MDOFs with $n_{\frac{1}{2}}$ of 15K for corresponding $t_{\frac{1}{2}}$ of 0.300ms.

Using the Intel compiler in BP1, for Nek5000 we observe an r_{max} of 200 MDOFs with $n_{\frac{1}{2}}$ of 15K for corresponding $t_{\frac{1}{2}}$ of 0.150ms. Using the Intel compiler in BP2, for Nek5000 we observe an r_{max} of 200 MDOFs with $n_{\frac{1}{2}}$ of 30K for corresponding $t_{\frac{1}{2}}$ of 0.300ms. Using the Intel compiler in BP1, for MFEM we observe an

r_{max} of 150 MDOFs with $n_{\frac{1}{2}}$ of 30K for corresponding $t_{\frac{1}{2}}$ of 0.400ms. Using the Intel compiler in BP2, for MFEM we observe an r_{max} of 100 MDOFs with $n_{\frac{1}{2}}$ of 15K for corresponding $t_{\frac{1}{2}}$ of 0.300ms.

These values, as well as similar results in the following sections are summarized in a Table 1 in Section 3.4.4. We observe that the Intel compiler gives a significant boost to Nek5000, in particular for BP2, while GCC is more favorable to the MFEM implementation. More detailed in-depth analysis of these and the other results from this section will be a near-term focus for the team.



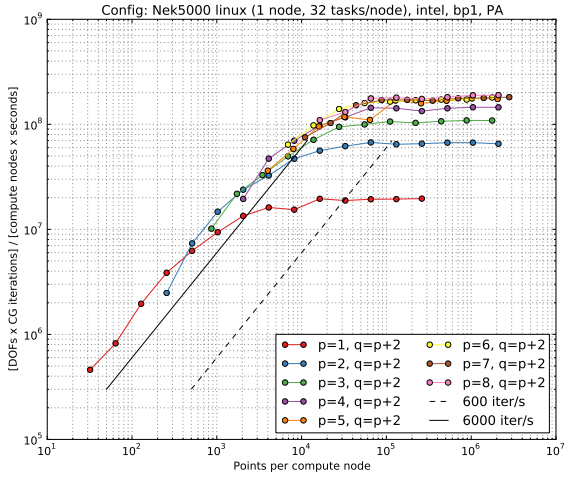
(a) BP1 Nek5000

(b) BP1 MFEM

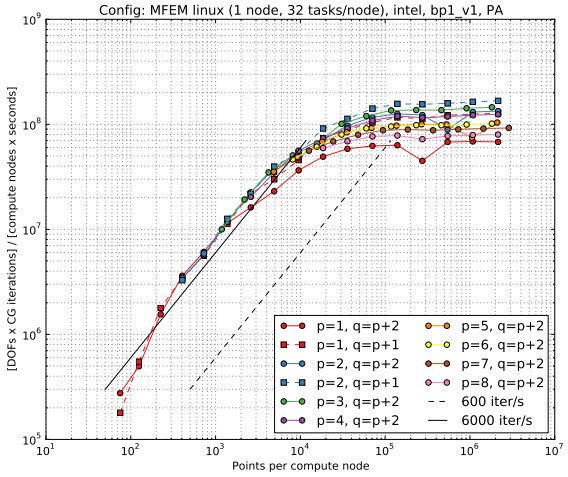
(c) BP2 Nek5000

(d) BP2 MFEM

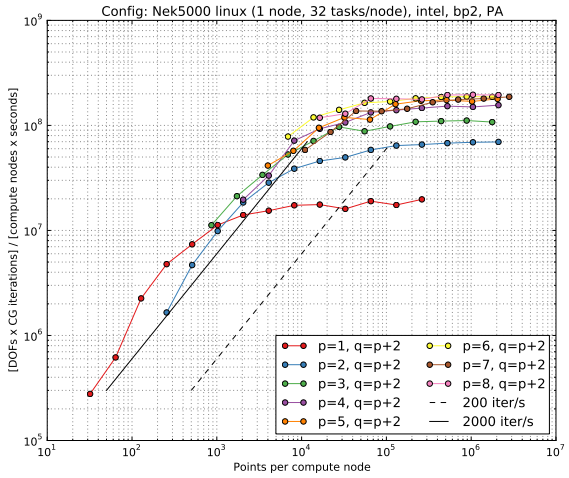
Figure 2: BP1 and BP2 results on a KNL machine using the GCC compiler and various polynomial orders (p), number of quadrature points (q). The x- and y-axes indicate the "scalar" problem size and performance (dps) respectively. The slanted lines corresponds to fixed CG iterations per second, indicating actual computational time.



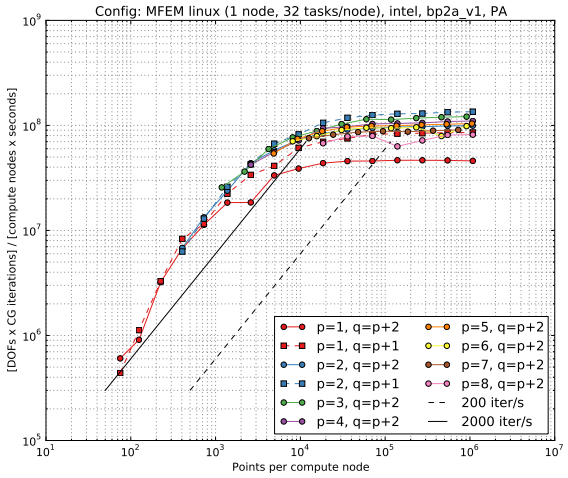
(a) BP1 Nek5000



(b) BP1 MFEM



(c) BP2 Nek5000

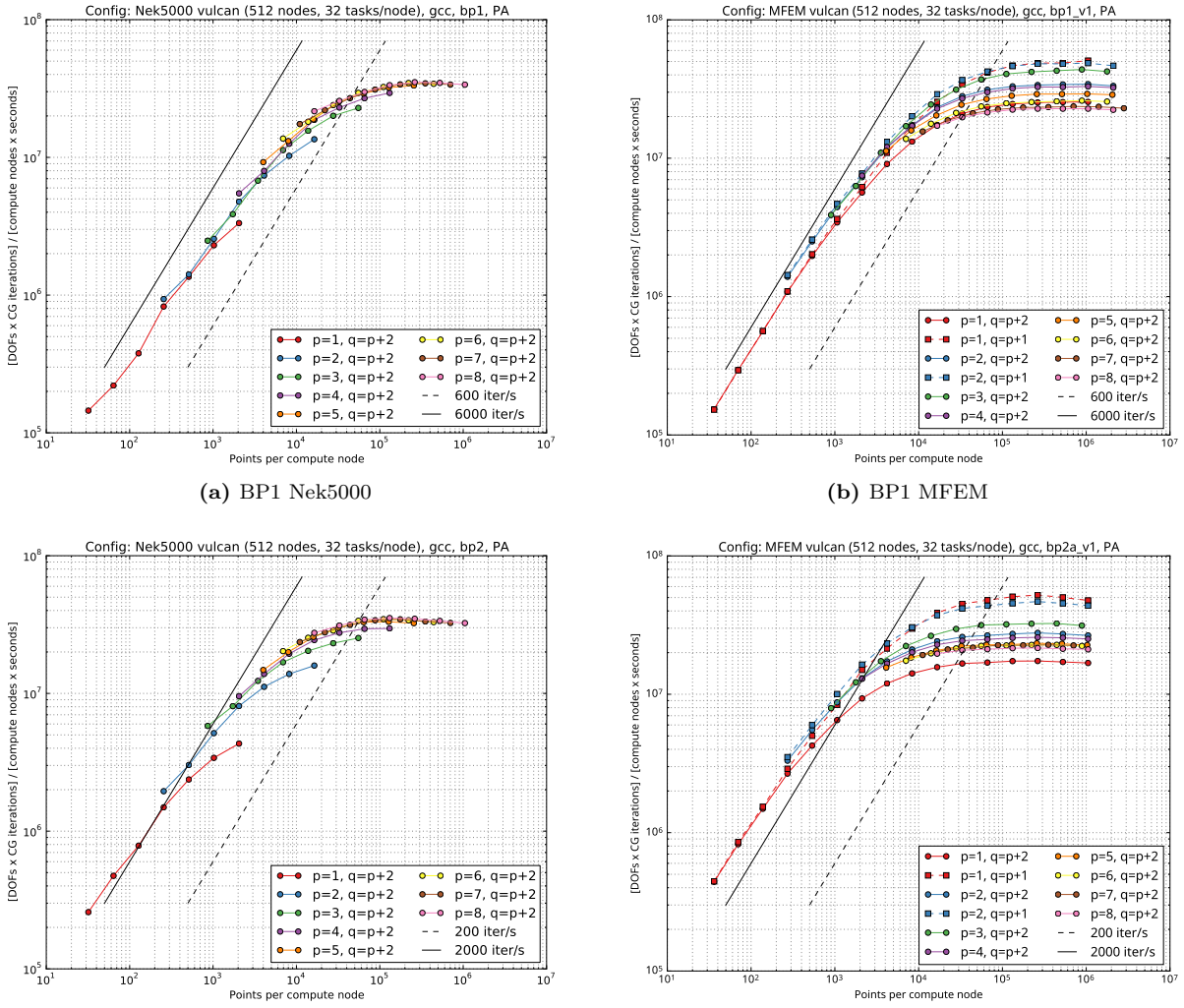


(d) BP2 MFEM

Figure 3: BP1 and BP2 results on a KNL machine using the Intel compiler and various polynomial orders (p), number of quadrature points (q). The x- and y-axes indicate the "scalar" problem size and performance (dps) respectively. The slanted lines corresponds to fixed CG iterations per second, indicating actual computational time.

3.4.2 *Results on a BG/Q Cluster*

In this section we report the results for BP1 and BP2 on Vulcan which is a BG/Q machine at LLNL. BG/Q is a single chip processor which has 16 compute cores and 16 GB DDR 3 memory. In Figure 4 we present the Nek5000 and MFEM results for BP1 and BP2 with 512 nodes with 32 MPI task per node using the GCC compiler. For Nek5000, we observe an r_{max} of 35 MDOFs with $n_{\frac{1}{2}}$ of 15K for corresponding $t_{\frac{1}{2}}$ of 0.857ms. For BP2, r_{max} stays roughly the same for Nek5000 with $n_{\frac{1}{2}}$ of 21K and $t_{\frac{1}{2}}$ of 1.200ms For MFEM, the BP1 results are 40 MDOFs, 10K and 0.50ms for BP1 and 30 MDOFs, 10K and 0.666ms for BP2. We observe that Vulcan is 3–5 $\times$ weaker than the KNL box in terms of maximal performance, but has similar strong-scaling characteristics in terms of $n_{\frac{1}{2}}$ resulting 2–3 $\times$ slower $t_{\frac{1}{2}}$.



(a) BP1 Nek5000

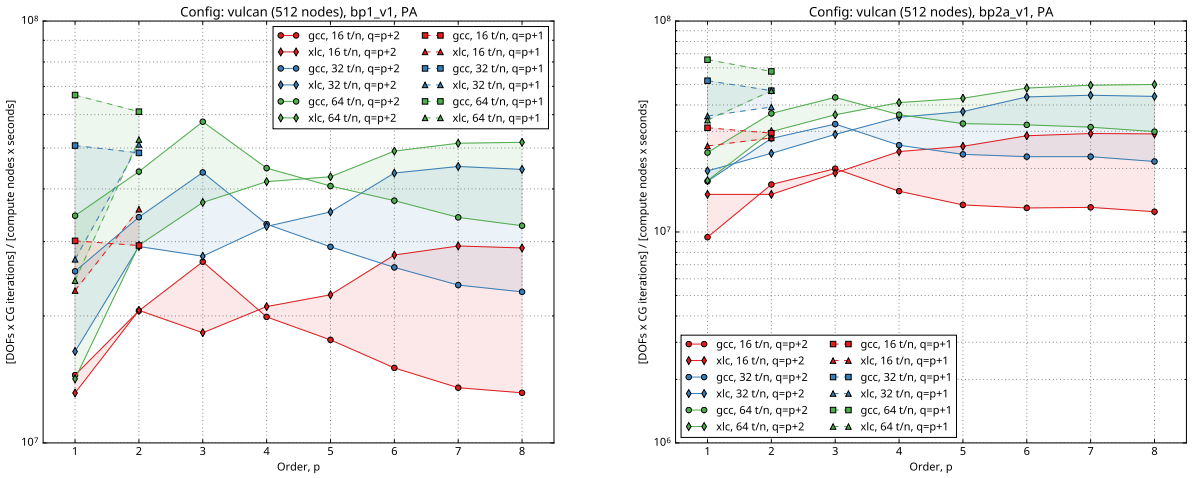
(b) BP1 MFEM

(c) BP2 Nek5000

(d) BP2 MFEM

Figure 4: BP1 and BP2 results on the Vulcan at LLNL using the GCC compiler and various polynomial orders (p), number of quadrature points (q). The x- and y-axes indicate the "scalar" problem size and performance (dps) respectively. The slanted lines corresponds to fixed CG iterations per second, indicating actual computational time.

Additional MFEM results: In the next set of results, we investigate the effects on performance of using different compilers and numbers of MPI tasks per node on Vulcan, see Figure 5.



(a) BP1 **(b)** BP2

Figure 5: Comparison of BP1/BP2 MFEM results with different compilers (GCC v4.7.2 and xlC v12.01) and varying number of MPI tasks per node (t/n). The performance numbers (y-axis) are taken as the maximum over all grid resolutions.

Clearly, the MFEM test code is sensitive to the compiler with widely varying results depending on the polynomial order. This type of strong dependence on the compiler is partially due to the use of C++ templating to fix the order-dependent sizes of inner-most loop at compile time. Improving the sensitivity of our kernels with respect to the compiler and specific optimization options will be one of the goals of our optimization work.

With regards to the impact of using a larger number of MPI tasks per node, we can see that this is beneficial, for the MFEM test code, in almost all cases. This effect is due to the use of multiple hardware threads per core on the BG/Q platform. Since this characteristic will be present (and actually enhanced to 8 hardware threads per core) in the upcoming Sierra and Summit machines at LLNL and ORNL, we plan to continue to use this technique of over-subscribing the cores with multiple MPI tasks.

BP3 and BP4 MFEM results. In Figure 6, we show the summary results (over the various grid resolutions) for BP3 and BP4 using the MFEM benchmark tests. As can be expected, the overall performance is lower compared to BP1 and BP2, due to the more computationally intensive kernels. However, the overall variations with respect to the order and number of MPI tasks per node remain very similar.

In Figure 7, we show the more detailed results for BP3 and BP4 as functions of the resolution (in terms of DOFs per compute node), in the case of 64 MPI tasks per node.

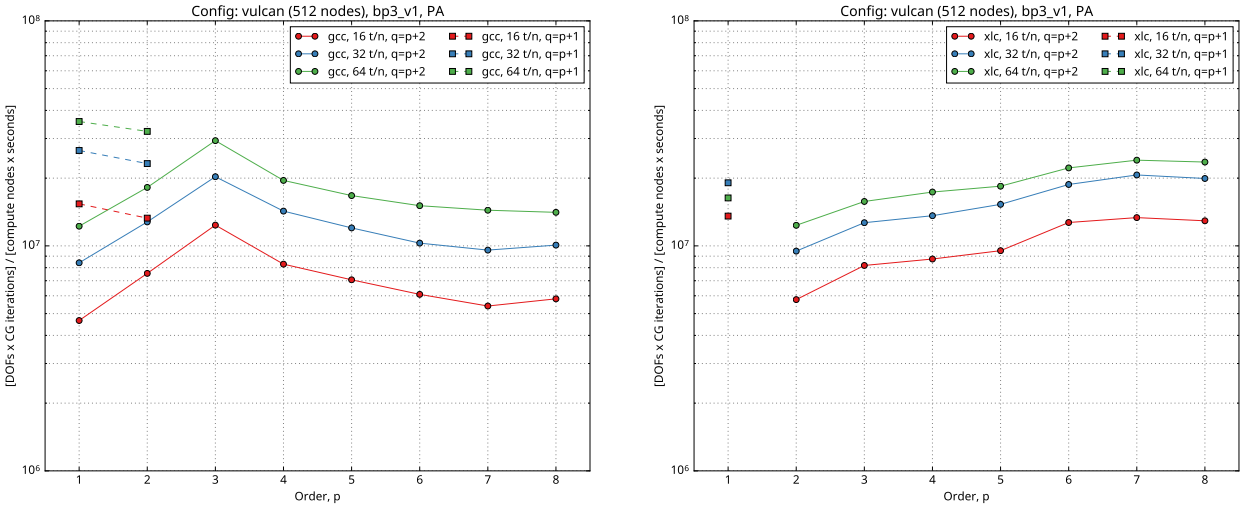
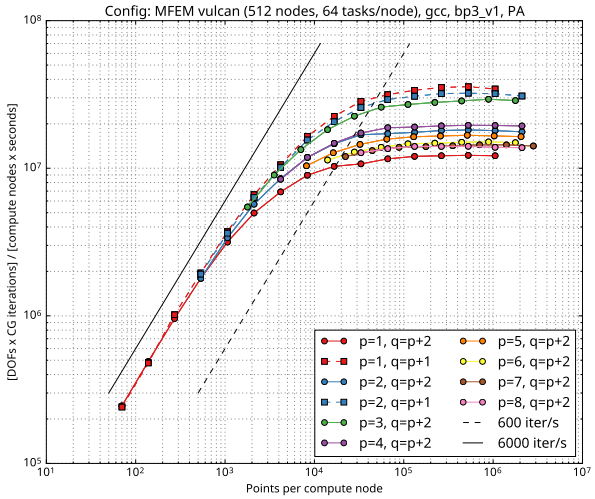
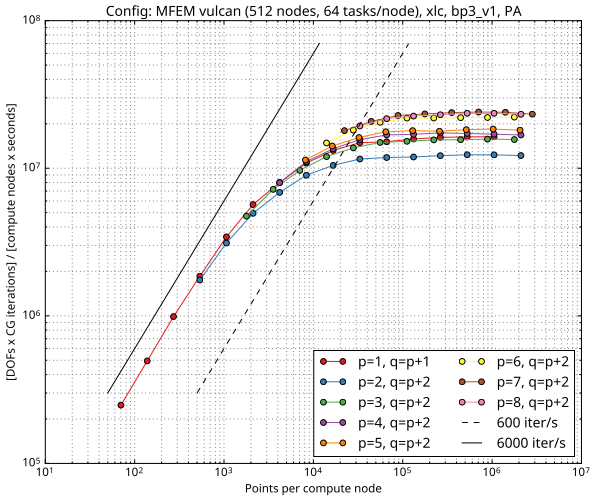


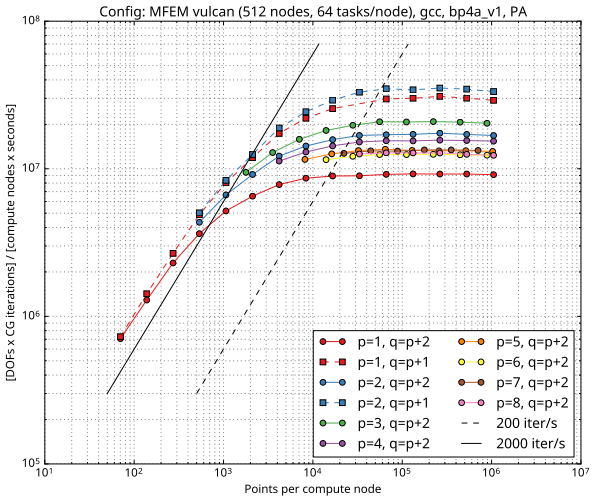
Figure 6: Summary results for BP3 in MFEM with GCC (left) and xlC (right) and varying number of MPI tasks per node (t/n). The performance numbers (y-axis) are taken as the maximum over all grid resolutions.



(a) BP3 MFEM with GCC



(b) BP3 MFEM with xlC

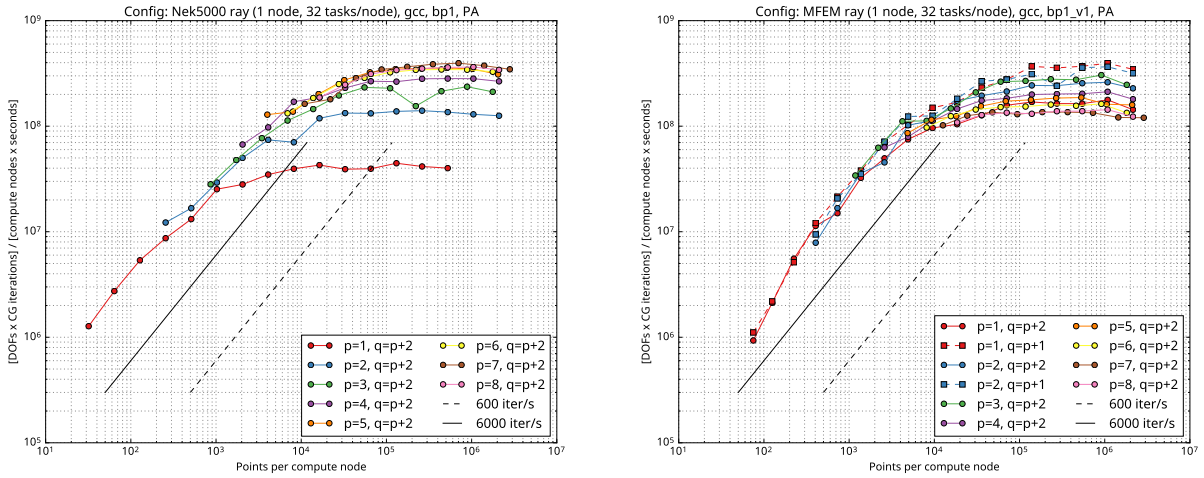


(c) BP4 MFEM with GCC

Figure 7: BP3 and BP4 MFEM results using GCC and xlC compilers, various polynomial orders (p), and number of quadrature points (q). The slanted lines corresponds to fixed CG iterations per second, indicating the same computational time for varying problem sizes (points) per compute node.

3.4.3 *Results on an Early-access GPU Testbed*

In this section we report the BP1 results on Ray, an early-access GPU testbed at LLNL, similar to SummitDev. We run the benchmark on one node of the machine, which has two IBM Power8 CPUs, each with 10 cores with 8 threads per core (the node GPUs were not used for this test). In Figure 8 we present the Nek and MFEM results with 32 MPI tasks using the GCC compiler. For Nek5000, we observe an r_{max} of 400 MDOFs with $n_{\frac{1}{2}}$ of 15K for corresponding $t_{\frac{1}{2}}$ of 0.075ms. For MFEM, we get r_{max} of 300-400 MDOFs, $n_{\frac{1}{2}}$ of 30K-40K and $t_{\frac{1}{2}}$ of 0.2ms. In Figure 9 we compare the BP1 results with MFEM when using different compilers (GCC vs. xlC) and different number of MPI tasks on the node. The plot shows that compiler optimization can play an increasing role at higher orders, where xlC can be up to 3 times faster than GCC.



(a) Nek5000 **(b)** MFEM

Figure 8: BP1 results on Ray using the GCC compiler and various polynomial orders (p), number of quadrature points (q). The x- and y-axes indicate the problem size and performance (dps) respectively. The slanted lines corresponds to fixed CG iterations per second, indicating actual computational time.

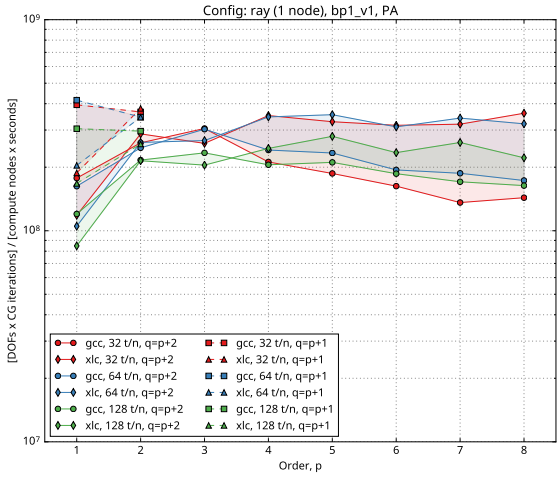


Figure 9: BP1 results for MFEM on one node of Ray comparing the GCC and xlC compilers with various polynomial orders (p), number of quadrature points (q), and number of MPI tasks (32, 64 and 128). The x- and y-axes indicate polynomial order (p) and performance (dps) respectively.

Table 1: Performance results for various machines using metrics defined in Sec. 3.5. Each entry corresponds to Nek/MFEM results. Results sorted by problem and $t_{\frac{1}{2}}$ performance.

Machine/Problem	r_{max} (MDOFs)	$n_{\frac{1}{2}}$ (KDOFs)	$t_{\frac{1}{2}}$ (ms)
Ray/BP1-gcc	400/350	15/35	0.075/0.200
KNL32/BP1-intel	200/150	15/30	0.150/0.400
KNL32/BP1-gcc	100/100	20/8	0.400/0.160
Vulcan/BP1-gcc	35/40	15/10	0.857/0.500
KNL32/BP2-intel	200/100	30/15	0.300/0.300
KNL32/BP2-gcc	100/100	50/15	1.000/0.300
Vulcan/BP2-gcc	35/30	21/10	1.200/0.666
Vulcan/BP3-gcc	/30	/10	/0.666
Vulcan/BP3-xlc	/20	/10	/1.000
Vulcan/BP4-gcc	/20	/10	/1.000

3.4.4 Summary of Initial Bake-Off Results

We summarize the bake-off performance results on various machines in Table 1 below. We emphasize that these are initial results that we will continue to investigate and benchmark against. These results will also serve as a foundation for future performance improvements and API unification efforts in the CEED center.

3.5 Performance Metrics

Assessment and modeling of parallel performance on emerging architectures is essential for the CEED team to reach their objectives. In this section we look at the challenges in assessing the performance across a wide variety of hardware and come up with three different metrics that provide a reasonable assessment about the performance of a given computer system. Three metrics, r_{max}, $n_{\frac{1}{2}}$, and $t_{\frac{1}{2}}$, were identified in the initial bake-off studies of the preceding sections. Here, we define them and explain their relevance.

A significant challenge in assessing future parallel platforms stems from the wide variance in the type and performance of prospective node architectures. (for example, GPUs vs. traditional multi-core machines). An obvious first step in such a comparison is to measure the rate of computation that a node delivers for a given application code (which, in the HPC context, we presume is tuned for each architecture) for given per-node cost or per-node power consumption.

In fact, such a measurement would return three values of interest. Let us suppose that the problem to be solved is a partial differential equation in space and time having N gridpoints in space and taking N_s timesteps. For fixed N, the rate of work, R, will be the number of timesteps per second of wall-clock time, which is readily measurable by a user and (often) predictable on a prospective future platform through use of a simulator. Assuming that the algorithmic cost is linear in N, it is often useful to define an intrinsic rate of work, $r := N \cdot R$, having units of (number-of-points $\times$ number-of-steps) per wall-clock second. Note that r is a multiple of the realized FLOPs rate. For example, using geometric multigrid to solve the 3D Poisson equation with a 7-point finite difference stencil requires about 50 FLOPs per V-cycle per gridpoint, so $r = f/50$, where f is the flop rate and N_s is interpreted as the number of V-cycle iterations. Typically, r asymptotes to a constant $r_{\max}$ for large N and drops towards zero for small N. With these definitions, interesting figures of merit include

- $r_{\max}$, the peak observable rate of execution,
- $r_{\max}$/cost, the rate of execution per dollar invested,
- $r_{\max}$/watt, the rate of execution per unit of energy required , and

- M, the amount of available memory.

Knowing the rate r, the user can predict the time to solution, $T = (N_s\, N)/r$ for a given number of steps, N_s, and gridpoints, N.

For someone purchasing a single node, the above list would likely constitute the most important considerations in evaluating a candidate architecture (aside from the software development overhead, which we assume is already resolved). In an HPC context, however, users are never running on a single node, for two primary reasons. The first is memory. HPC applications will generally not fit on a single node. Fortunately, distributed memory (DM) parallel computing provides a P-fold increase in available memory when P identical nodes are used instead of one. The second is speed. DM architectures offer a P-fold increase in number of cores *and* in memory bandwidth to the cores. Leadership computing facilities support anywhere from $P = 10^4$ to 10^5 nodes, which gives a very large multiplier in R and M but not much direct variation in the relative measures of R/cost or R/watt. (Aside perhaps from some economies of scale, one would expect a machine with 2×10^4 nodes to cost roughly twice that of a machine with 10^4 nodes.)

If one can realize a P-fold speed up by using P nodes instead of one, a natural question that arises is, why not set P equal to the number of nodes in the machine? In traditional DM architectures, the answer is well known. For a fixed problem size, N, the parallel work scales as N/P, which goes to zero with increasing P, but the serial work and communication overhead does not go to zero and may in fact increase with P. At some value of N/P, the parallel efficiency falls off to an unacceptable level and the user will choose, at runtime, to set P to be less than this unacceptable value. Thus, the user is ultimately controlling the *granularity* (N/P) of the problem and it is generally in his best interest to set P as large as possible, that is, at this strong-scale limit where the parallel efficiency is acceptable.

Under the strong-scale conditions to which most applications will gravitate, one has to add an important figure of merit to the preceding list. For HPC applications, each node sees a local problem size of only N/P, rather than N. It is thus important to consider a node's performance rate as a function of *local* problem size N/P. Most often, this functionality is characterized by $n_{\frac{1}{2}}$, the problem size where the node performance for the given application, r, drops below one-half of its peak value. Thus, even neglecting communication overhead, there is a strong-scale limit imposed on P that is governed by $n_{\frac{1}{2}}$. If the problem size is N, and P satisfies

$$\frac{N}{P} \;=\; n_{\frac{1}{2}},$$

we can anticipate that the maximum parallel execution rate will be $r_{ss} = \frac{1}{2} P r_{\max}$, and the DM parallel *time-to-solution* will be

$$T_{\min} \;=\; \frac{N_s N}{r_{ss}} \;=\; \frac{N_s N}{\frac{1}{2} P\, r_{\max}} \;=\; 2 N_s \frac{n_{\frac{1}{2}}}{r_{\max}}. \tag{15}$$

Although increases in model complexity or fidelity will often increase the number of time steps N_s, the best we can hope for the ratio

$$t_{\frac{1}{2}} = 2 \frac{n_{\frac{1}{2}}}{r_{\max}},$$

the fastest execution time per time step at acceptable efficiency, is to remain constant. This is the time seen by the end user, and it is therefore a critical parameter in the design of HPC systems. Note that the pre-factor of 2 stems from the particular choice of $\frac{1}{2}$ as the acceptable level of efficiency. One could equally well choose some other acceptable efficiency, say $\varepsilon = 0.8$, and the same argument would carry through with subdomain size n_ε and associated time $t_\varepsilon = \varepsilon^{-1} n_\varepsilon / r_{\max}$.

Clearly, order-unity values of $n_{\frac{1}{2}}$ are never going to be realizable. If the node supports P_n cores, one certainly must have $n_{\frac{1}{2}} > P_n$. Pipelined execution and devices such as hyperthreading to cover memory latency all help to boost $r_{\max}$, which is good. But they often simultaneously increase $n_{\frac{1}{2}}$. Communication overheads, including on-node and off-node overheads incurred by load-balancing techniques like overdecomposition, may have negligible impact on $r_{\max}$ but generally increase $n_{\frac{1}{2}}$, thus also $t_{\frac{1}{2}}$. Similarly, while coprocessors often greatly increase $r_{\max}$, they also incur overheads that usually increase $t_{\frac{1}{2}}$. *The crucial point is that future architectures and algorithm development must be evaluated with respect to $t_{\frac{1}{2}}$, i.e. both $r_{\max}$ and $n_{\frac{1}{2}}$ if one wishes to maintain or improve time to solution.*

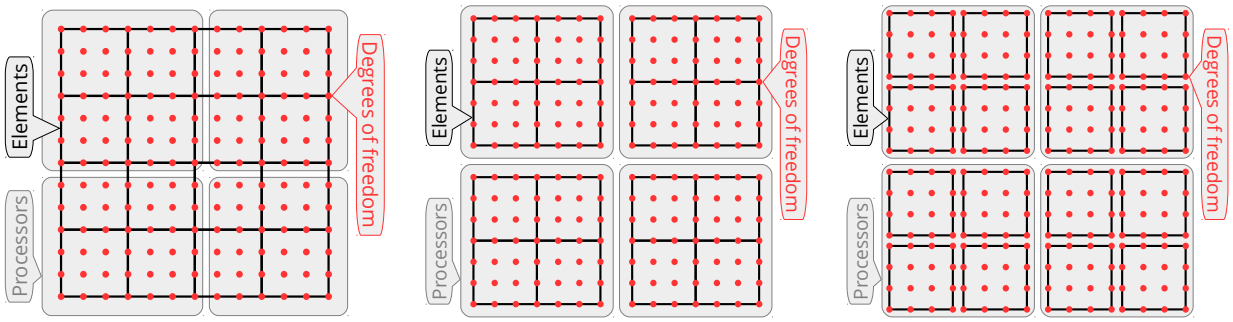


Figure 10: Left to right: T-vector, L-vector, and E-vector representations.

3.6 CEED Library Terminology and Notation

In addition to developing benchmarks and miniapps, the CEED team has been working to identify a unifying framework coupled with central kernels that will provide flexibility and high performance for efficient discretization on future architectures. The following sections describe initial definitions in this space. As mentioned earlier, identifying and improving upon the performance of these kernels, within realistic use settings, is one of the objectives of the bake-off exercise and forms an important part of our outreach to the ECP software groups and vendors.

The principal decomposition of fast matrix-free discretizations is broken down into the following vector-representation/storage-categories (see Figure 10):

- True degrees of freedom/unknowns, **T-vector**:
  - each unknown i has exactly one copy, on exactly one processor, $rank(i)$
  - this is a non-overlapping vector decomposition
  - usually includes any essential (fixed) DOFs.
- Local true degrees of freedom/unknowns, **L-vector**:
  - each unknown i has exactly one copy on each processor that owns an element containing i
  - this is an overlapping vector decomposition with overlaps only across different processors - there is no duplication of unknowns on a single processor
  - the shared DOFs/unknowns are the overlapping DOFs, i.e. the ones that have more than one copy, on different processors.
- Per element decomposition, **E-vector**:
  - each unknown i has as many copies as the number of elements that contain i
  - usually, the copies of the unknowns are grouped by the element they belong to.
- In the case of hanging nodes (giving rise to hanging DOFs):
  - there is another representation similar to L-vector which stores the hanging/dependent DOFs in addition to the true DOFs, **H-vector**
  - the additional hanging DOFs are duplicated when they are shared by multiple processors.
- In the case of variable order spaces:
  - the dependent DOFs (usually on the higher-order side of a face/edge) can be treated just like the hanging/dependent DOFs case
  - have both L- and H-vector representations.

- Quadrature point vector, **Q-vector**:
  - this is similar to E-vector where instead of DOFs, the vector represents values at quadrature points, grouped by element.
- In many cases it is useful to distinguish two types of vectors:
  - X-vector, or **primal** X-vector, and X'-vector, or **dual** X-vector
  - here X can be any of the T, L, H, E, or Q categories
  - for example, the mass matrix operator maps a T-vector to a T'-vector
  - the solutions vector is a T-vector, and the RHS vector is a T'-vector
  - using the parallel prolongation operator, one can map the solution T-vector to a solution L-vector, etc.

Discretized PDE operators can similarly be broken down into the following representation/storage/action categories:

- Full true-DOF parallel assembly, **TA**, or **A**:
  - ParCSR or similar format
  - the T in TA indicates that the data format represents an operator from a T-vector to a T'-vector.
- Full local assembly, **LA**:
  - CSR matrix on each rank
  - the parallel prolongation operator, P, (and its transpose) should use optimized matrix-free action
  - note that P is the operator mapping T-vectors to L-vectors.
- Element matrix assembly, **EA**:
  - each element matrix is stored as a dense matrix
  - optimized element and parallel prolongation operators
  - note that the element prolongation operator is the mapping from an L-vector to an E-vector.
- Quadrature-point/partial assembly, **QA** or **PA**:
  - precompute and store $w \det(\mathcal{J})$, or $\det(\mathcal{J})$ (in the case of mass matrix) at all quadrature points in all mesh elements
  - the stored data can be viewed as a Q-vector.
- Unassembled option, **UA** or **U**:
  - no assembly step
  - the action uses directly the mesh node coordinates, and assumes specific form of the coefficient, e.g. constant, piecewise-constant, or given as a Q-vector (Q-coefficient).

3.6.1 Low-Level Kernels

We have developed a set of reference implementations for kernels performing the action of mass and diffusion finite element operators in 2D and 3D. These implementations are available in the CEED benchmarks repository in the directory `tests/mfem_experiments`, where they can be tested in an example based on MFEM. The reference implementations can easily be replaced in the example with other optimized implementations to facilitate benchmarking of future performance improvement work.

We have also begun the work of combining these low-level kernels in a standalone low-level API library (working name: `libCEED`) that is the topic of a upcoming milestone (CEED-MS10).

3.6.2 Batched BLAS Kernels

We identified a number of initial low-level kernels that are fundamental for the performance portability of high-order finite element computations. In particular, these are the tensor contractions used in the tensor formulations of the high-order FE algorithms. Further, we managed to classify and restrict these contractions to three main cases. This allowed us to directly target these discrete cases for optimization on various architectures. We analyzed these main kernels to show the trade-offs between the various implementations, how different tensor kernel implementations perform on different architectures, and what tuning parameters can have a significant impact on performance. In all cases, we organize the tensor contractions to minimize their communications by considering index reordering that enables their execution as highly efficient batched matrix-matrix multiplications (DGEMMs). For example, contractions by the first index as in

$$C_{i_1,i_2,i_3} = \sum_{k_1} A_{k_1,i_1} B_{k_1,i_2,i_3},$$

can be written as $Reshape(C)^{nd_1 \times (nd_2 nd_3)} = A^T \; Reshape(B)^{nq_1 \times (nd_2 nd_3)}$.

We derived performance models and bounds for the maximum performance that can be achieved under the maximum data-reuse scenario, and showed that our implementations achieve 90+% of these theoretically derived peaks on advanced multi-core x86 CPU, ARM, GPU, and Xeon Phi architectures. These results significantly outperform what is available today in vendor libraries. In particular, we show average performance speedups of 1.3× and 2× compared to Intel MKL on two 10-core Haswell CPUs and KNL Xeon Phi, respectively, and 3× when compared to NVIDIA CUBLAS on the latest P100 NVIDIA GPU [8].

```
extern "C" void
magmablas_dgemm_batched( magma_trans_t transA, magma_trans_t transB,
                         magma_int_t m, magma_int_t n, magma_int_t k,
                         double alpha,
                         double const * const * dA_array, magma_int_t ldda,
                         double const * const * dB_array, magma_int_t lddb,
                         double beta,
                         double **dC_array, magma_int_t lddc,
                         magma_int_t batchCount, magma_queue_t queue )
```

(a) Fixed sizes

```
extern "C" void
magmablas_dgemm_vbatched(
    magma_trans_t transA, magma_trans_t transB,
    magma_int_t* m, magma_int_t* n, magma_int_t* k,
    double alpha,
    double const * const * dA_array, magma_int_t* ldda,
    double const * const * dB_array, magma_int_t* lddb,
    double beta,
    double **dC_array, magma_int_t* lddc,
    magma_int_t batchCount, magma_queue_t queue )
```

(b) Variable sizes

Figure 11: Batched BLAS API – example interfaces for the batched matrix-matrix multiplication in double precision (DGEMM) for fixed-size matrices (top) and variable-size matrices (bottom).

The use of vendor-optimized BLAS has been successfully used in many applications and numerical libraries like LAPACK, ScaLAPACK, and MAGMA (among many other libraries), to provide performance portability across architectures. Similarly, the availability of highly optimized Batched BLAS for various architectures can provide tensor contractions, and consequently high-order applications, performance portability. Therefore, we have been working with vendors and the community on defining a *Batched BLAS API*. The progress in that direction is promising after we organized two workshops on the topic and started work on proposing an API for Batched BLAS [9]. An illustration for the Batched BLAS interfaces is given in Figure 11 for the case of batched matrix-matrix multiplications in double precision (DGEMM). The matrices in the batch can be the same (fixed) size (Figure 11, Top) or variable sizes (Figure 11, Bottom).

We have developed, optimized, and released through MAGMA, see `http://ceed.exascaleproject.org/magma`, all Level 3 Batched BLAS routines for both fixed and variable size matrices. A number of Level 2 Batched BLAS are also developed and released (SYMV and GEMV), as well as a number of other Batched BLAS kernels, as needed in the development of tensor contractions.

4. ENGAGEMENTS WITH VENDORS AND ST PROJECTS

4.1 ECP Vendors

CEED has begun engaging ECP vendors in several directions. For example, Ian Karlin visited AMD and ARM to discuss the CEED technologies including our miniapps and future plans. The vendors shared details of what they are doing (removed from this report due to NDA issues) and we established initial connections for future co-design. We have sent the new Laghos miniapp to AMD to begin collaboration with them. In addition, CEED participated in a meeting with HP that was hosted at LLNL discussing the networking part of their PathForward contract. We are also beginning to engage actively with Intel and will send several participants to the July 2017 co-design meeting at Intel's Hudson MA facility.

Most of the PathForward vendors will have representatives at CEED's annual meeting, see Section 5.3.

4.2 MPICH

Nek5000 and MPICH teams have shared milestones including Nek5000 performance and analysis for standard vs. light-weight MPICH. The results have been submitted to SC17 and have been accepted for publication [10].

Figure 12 shows Nek5000 performance results on the ALCF BG/Q Cetus. The underlying mesh is a tensor product array of brick elements, each of order N (i.e. p from the previous sections), and the problem is perfectly load balanced, with $E = 2k$, for $k = 14, ..., 21$. Here, we consider $N = 3, 5, 7$, such that n/P, the problem size per processor, is in the range $n/P \in [27, 43904]$. Earlier MPICH analysis suggests that BG/Q should yield order-unity efficiency for this problem with $n/P >$ 1000–2000, a transition point well within the range covered here. In Figure 12 (left), for large n/P the problem becomes work-dominated; for small n/P it is communication dominated. The dashed lines show the performance for the standard MPICH, while the solid lines show results for light-weight MPICH. Each pair of curves reflect a different value of N. We see that the lower value of N does not perform well, which is in part due to caching and vectorization strategies in Nek5000, but also due to the $O(M^3N)$ interpolation overhead, which is large when N is small. In all cases, light-weight MPICH is seen to outperform standard MPICH, save for the very largest values of n/P, where they are equal.

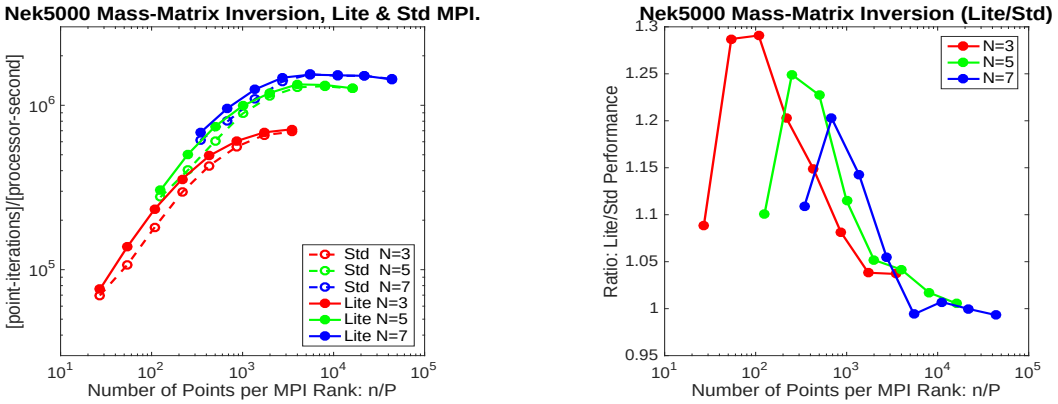


Figure 12: Mass-matrix inversion performance for Nek5000 on Cetus: (left) performance vs. n/P for MPICH/Original (solid lines) and MPICH/CH4 (dashed lines); (right) performance ratio for $N = 5$ and $N = 7$.

4.3 STRUMPACK

We have integrated support for the sparse direct solver and preconditioner STRUMPACK in MFEM. STRUMPACK is being developed at LBNL [11] and is part of the ECP project "Factorization Based Sparse Solvers and Preconditioners" (Xiaoye Sherry Li and Pieter Ghysels). The STRUMPACK solver is based on multifrontal sparse Gaussian elimination and uses hierarchically semi-separable matrices to compress fill-in. It can be used as an exact direct solver or as an algebraic, robust and parallel preconditioner for a range of discretized PDE problems. A new solver class STRUMPACKSolver was added to MFEM wrapping the STRUMPACK solver. Example usage is illustrated by the `ex11p` example, where STRUMPACK can be used as a preconditioner for the LOBPCG eigensolver. STRUMPACK options can be set via the STRUMPACKSolver class interface or via command line arguments that are passed to STRUMPACK. The MFEM documentation, installation instructions, website, changelog, build scripts, etc, have been updated accordingly. An extensive performance evaluation will be performed in the near future and results will be reported at the CEED annual meeting, see Section 5.3.

5. OTHER PROJECT ACTIVITIES

5.1 CEED Software on GitHub

All CEED software products described in the previous sections are available as open-source projects on GitHub, or will be made available soon after a review process at LLNL. The CEED presence on GitHub now includes a user forum at `https://github.com/CEED/Forum` and mirrors of the major CEED packages at `https://github.com/CEED`.

In Q3 FY17, we also released GSLIB, Nek5000's gather-scatter library, as a standalone library on GitHub at `https://github.com/gslib/gslib`. GSLIB provides efficient nearest neighbor data exchanges for spectral element, finite element, and finite-difference applications. It can also be used for efficient transpose and reduction operations and for smoothing, prolongation, and restriction in AMG and other solvers. See `http://ceed.exascaleproject.org/gslib` for more details.

5.2 Public-facing Project Website

The CEED team developed a public-facing website that provides information about the project, documents benchmark problems and miniapps and can serve in the future as a hub for the high-order community (milestone CEED-MS4). The website is available at `http://ceed.exascaleproject.org`, see Figure 13. The site includes pages for the four CEED R&D thrusts, the CEED's software packages and the project

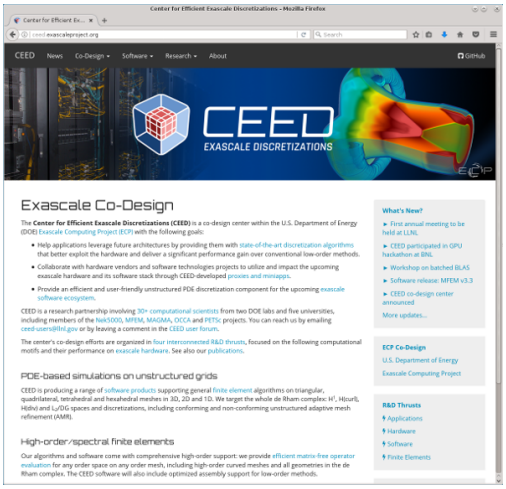


Figure 13: Main page of CEED's public-facing website.

efforts in high-order meshing, high-order physics discretization, linear algebra and high-order visualization.

To encourage frequent contributions by the whole team, the site is build on simple Markdown files and is hosted on GitHub, integrating naturally with the rest of the CEED workflow.

5.3 Planning of the CEED First Annual Meeting

We started preparations for CEED's first annual meeting, to be held August 15–17 at the HPC Innovation Center of Lawrence Livermore National Laboratory. The goal of the 3-day meeting is to report on the progress in the center, deepen existing and establish new connections with ECP hardware vendors, ECP software technologies projects and other collaborators, plan project activities and brainstorm/work as a group to make technical progress. In addition to gathering together many of the CEED researchers, the meeting will include representatives of the ECP management, hardware vendors, software technology and other interested projects. The current list of attendees includes 60 researchers from 20+ different institutions.

6. CONCLUSION

In this milestone we identified CEED's initial kernels, bake-off problems and miniapps and delivered software and documentation for them through the new CEED website, `http://ceed.exascaleproject.org`.

We specifically identified the Nekbone miniapp for the ExaSMR application and developed a new miniapp, Laghos, for the MARBL/LLNLApp application. We formulated four initial benchmark problems (BP1-BP4) with clearly defined performance metrics and performed an initial bake-off comparison between the Nek (spectral element) and MFEM (high-order finite element) technologies. As part of the milestone, we engaged ECP vendors (AMD) and ST projects (MPICH, STRUMPACK).

In this report, we also described additional CEED activities performed in Q3 of FY17, including: making CEED software available at `https://github.com/ceed`, the new public-facing project website `http://ceed.exascaleproject.org` and the preparation of CEED's first annual meeting.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, LLNL-TR-734169.

REFERENCES

[1] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High-order curvilinear finite element methods for Lagrangian hydrodynamics. *SIAM J. Sci. Comp.*, 34(5):B606–B641, 2012.

[2] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-order methods for incompressible fluid flow*. Cambridge University Press, Cambridge, 2002.

[3] S.A. Orszag. Spectral methods for problems in complex geometry. *J. Comput. Phys.*, 37:70–92, 1980.

[4] A.T. Patera. A spectral element method for fluid dynamics : laminar flow in a channel expansion. *J. Comput. Phys.*, 54:468–488, 1984.

[5] Y. Maday and A.T. Patera. Spectral element methods for the Navier-Stokes equations. In A.K. Noor and J.T. Oden, editors, *State-of-the-Art Surveys in Computational Mechanics*, pages 71–143. ASME, New York, 1989.

[6] Mark Ainsworth. Dispersive behaviour of high order finite element schemes for the one-way wave equation. *J. Comput. Phys.*, 259:1–10, 2014.

[7] J. Malm, P. Schlatter, P.F. Fischer, and D.S. Henningson. Stabilization of the spectral-element method in convection dominated flows by recovery of skew symmetry. *J. Sci. Comp.*, 57:254–277, 2013.

[8] Ahmad Abdelfattah, Marc Baboulin, Veselin Dobrev, Jack Dongarra, Christopher Earl, Joël Falcou, Azzam Haidar, Ian Karlin, Tzanio Kolev, Ian Masliah, and Stanimire Tomov. Small Tensor Operations on Advanced Architectures for High-order Applications. Technical Report UT-EECS-17-749, 04-2017 2017.

[9] Workshop on Batched, Reproducible, and Reduced Precision BLAS, 02-2017 2017. Available at: http://bit.ly/Batch-BLAS-2017.

[10] Ken Raffenetti, et.al, P. Fischer, T. Rathnayake, M. Otten, M. Min, and P. Balaji. Why is MPI so slow? analyzing the fundamental limits in implementing MPI-3.1. *SC17*, accepted, 2017.

[11] STRUMPACK: STRUctured Matrices PACKage. http://portal.nersc.gov/project/sparse/strumpack.