SIAM J. SCI. COMPUT.
Vol. 42, No. 3, pp. C97–C123
© 2020 Society for Industrial and Applied Mathematics

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

EFFICIENT MATRIX-FREE HIGH-ORDER FINITE ELEMENT EVALUATION FOR SIMPLICIAL ELEMENTS*

DAVID MOXEY†, ROMAN AMICI‡, AND MIKE KIRBY‡

Abstract. With the gap between processor clock speeds and memory bandwidth speeds continuing to increase, the use of arithmetically intense schemes, such as high-order finite element methods, continues to be of considerable interest. In particular, the use of matrix-free formulations of finite element operators for tensor-product elements of quadrilaterals in two dimensions and hexahedra in three dimensions, in combination with single-instruction multiple-data instruction sets, is a well-studied topic at present for the efficient implicit solution of elliptic equations. However, a considerable limiting factor for this approach is the use of meshes comprising of only quadrilaterals or hexahedra, the creation of which is still an open problem within the mesh generation community. In this article, we study the efficiency of high-order finite element operators for the Helmholtz equation with a focus on extending this approach to unstructured meshes of triangles, tetrahedra, and prismatic elements using the spectral/*hp* element method and corresponding tensor-product bases for these element types. We show that although performance is naturally degraded when going from hexahedra to these simplicial elements, efficient implementations can still be obtained that are capable of attaining 50% through 70% floating point operations of the peak of processors with both AVX2 and AVX512 instruction sets.

Key words. SIMD vectorization, high-order finite elements, spectral/hp element method, high-performance computing

AMS subject classifications. 65N30, 65Y05, 68W10

DOI. 10.1137/19M1246523

1. Introduction.

The development of robust, efficient solvers which utilize high-order or spectral/*hp* element methods is an area of considerable interest at present. The use of higher order polynomial expansions within elements carries a number of benefits as seen from two main perspectives. Numerically, these methods exhibit far lower levels of numerical dispersion and dissipation at higher polynomial orders. This makes them a particularly well-suited approximation choice in areas such as computational fluid dynamics, where the accurate time-advection of energetic structures such as vortices is a key concern [27]. However, perhaps the most appealing property of these methods in recent years has been their computational performance, particularly with respect to the present hardware landscape. Although the cost per degree of freedom in terms of algorithmic floating point operations (FLOPS) increases substantially with polynomial order, the use of higher order expansions leads to formulations of the underlying equations of state that involve dense, compact kernels for key finite element operators, such as inner products and derivatives. This is important from the perspective of modern hardware, where increasingly the bottleneck in performance

*Submitted to the journal's Software and High-Performance Computing section February 25, 2019; accepted for publication (in revised form) March 16, 2020; published electronically May 26, 2020.

https://doi.org/10.1137/19M1246523

Funding: The work of the first author was supported by the PRISM project through EPSRC under grant EP/R029423/1. The work of the second and third authors was supported by the AFRL through grant FA8650-17-C-5269.

†College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, Devon, EX17 1EJ, UK (d.moxey@exeter.ac.uk).

‡Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112 (amicir@gmail.com, kirby@cs.utah.edu).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

is memory bandwidth as opposed to the clock speed of processors. The underlying arithmetic intensity of the algorithm at hand (i.e., the number of FLOPS performed for each memory operation) is therefore key to attaining optimal performance. This is where high-order methods hold a significant advantage over lower-order methods.

This potential for increased performance is an aspect that has attracted considerable attention in the last few years. In particular, examining three key concepts in combination has yielded impressive computational results:

- *matrix-free* implementations of finite element operators, which avoid the explicit construction of either large, assembled global matrices (as is standard at lower orders) or even local, dense elemental operators (which is common at higher orders);
- the use of tensor products of one-dimensional basis functions to construct quadrilateral and hexahedral elements to enable the use of *sum-factorization* [20], which reduces operation counts substantially and makes matrix-free methods computationally attractive; and
- effective approaches for exploiting single-instruction multiple-data (SIMD) vectorization in a manner that aligns with the inherent parallelism of the finite element method in a computationally efficient manner. As modern hardware now relies on wide SIMD instructions to achieve increasing levels of computational power in newer generations of hardware, attaining peak performance relies on the efficient usage of such instructions. This may be done either transparently, by considering techniques such as compiler auto-vectorisation, or more opaquely through the adjustment of data layout to align more closely with the vectorized nature of SIMD instructions.

This combination can be seen in many recent publications and finite element codes. Implementations inside deal.II consider this extensively within both continuous Galerkin (CG) [2, 26] and discontinuous Galerkin (DG) settings [14, 13, 24] with various application areas including the incompressible Navier–Stokes equations. Dune [6] considers similar approaches for their Exa-DUNE implementation [4, 5]. Implementations utilizing a matrix-free approach without explicit SIMD vectorization (but with efficient small matrix multiplication kernels which can adopt this approach) can also be found in the incompressible fluid dynamics solver Nek5000 [15].

A limiting factor in this work, however, is that they rely on domains comprised solely of quadrilateral or hexahedral elements for two-dimensional and three-dimensional simulations, respectively. This is a significant issue when simulating complex geometries, since all-hexahedral mesh generation is an open problem even for linear finite element calculations. Indeed, the issue is further exacerbated when considering that, in order for the geometry to be accurately modeled, curvilinear meshes that are boundary conforming are required, which is itself an area of considerable interest [39, 33]. Although nonconformal adaptive mesh refinement is a potential route to deal with this issue (and is an approach considered in both deal.II [3] and Nek5000 [35] in combination with the parallel `p4est` library [7]), this approach still relies on an initial coarse grid, which can still at present be an issue for sufficiently complex geometries.

The simulation of complex geometries at high-order is instead under consideration by solvers including Nektar++ [8, 31] and PyFR [42], which use general, unstructured meshes of symplicial or hybrid elements: triangles in two dimensions and triangular prisms, tetrahedra, and square-based pyramids in three dimensions, potentially also in combination with quadrilaterals and hexahedra. In particular, Nektar++ permits the matrix-free evaluation of basic finite element operations which still utilize

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

sum-factorization even for non–tensor-product elements [40, 9, 32] through the use of a hierarchical C^0 basis introduced in [20]. Similarly, Berstein–Bézier basis functions such as those studied extensively by Ainsworth [1] and Kirby [22] can leverage the same structure for fast evaluation of finite element kernels. However, there is no particular consideration of data layout and SIMD vectorization in the aforementioned works; furthermore, for implicit solutions of elliptic problems such as the Poisson and Helmholtz equations, Nektar++ typically uses local matrix generation combined with static condensation to reduce operator counts.

The aim of this paper, therefore, is to consider the aforementioned C^0 basis, and other similar tensor-product basis choices, in the context of a matrix-free implementation of the Helmholtz operator for meshes of high-order unstructured elements. Adopting a similar approach to the previously cited work and, in particular [2], we consider a memory layout that interleaves elemental degrees of freedom to allow explicit exploitation of SIMD vectorization. We combine this with hand-written kernels for the key components of the Helmholtz operator, which make heavy use of intrinsics to avoid the pitfalls of compiler auto-vectorization. The efficiency of these kernels is then examined by considering their application to a number of two- and three-dimensional geometries in various hardware architectures.

The paper is structured as follows. In section 2, we lay out basic theory and background for high-order unstructured elements and the matrix-free evaluation of the Helmholtz operator. In section 3, we discuss the particulars of implementation. Section 4 applies these kernels to a variety of geometries in order to detail their performance properties. Finally, we conclude with a brief discussion of the results and future work in section 5.

2. Theory and background. The starting point for this article is the consideration of a finite element decomposition of the Helmholtz equation

$$\nabla^2 u - \lambda u = f(x) \tag{2.1}$$

on a domain $\Omega \subset \mathbb{R}^d$ for $d = 2, 3$, with $\lambda > 0$ being a positive constant, and $u, f : \Omega \to \mathbb{R}$ scalar functions denoting the desired solution and forcing term, respectively. For simplicity of implementation, we assume homogeneous Neumann boundary conditions on the boundary $\partial\Omega$. We select the Helmholtz equation as a representative elliptic problem that demonstrates various building-block finite element operators, as well as itself playing a significant role in the solution of more complex systems of equations. A prominent example of this is an implicit-explicit operator splitting scheme for the incompressible Navier–Stokes equations [21], which is widely used and implemented within a number of high-order codes [35, 15, 8, 14]. This scheme involves the solution of a Poisson equation for pressure and d Helmholtz equations to perform a correction step for each velocity component. Efficient techniques for the evaluation and solution of these equations are therefore highly desired.

As in any other finite element decomposition, the starting point is to consider the tessellation of the domain $\mathcal{T}(\Omega)$ into discrete elements $\Omega^e \in \mathcal{T}(\Omega)$. In two dimensions, we consider elements of potentially curvilinear triangles and quadrilaterals; in three dimensions, hexahedra, triangular prisms, and tetrahedra. In this paper, we are mostly concerned with the evaluation of finite element operators without consideration of elemental connectivity. By focusing on elemental operations, the techniques we propose are amenable to a wide range of schemes, including DG and CG discretizations, for both conformal and nonconformal meshes. However, for the purposes of demonstration and discussion of basis choice later, we consider a conformal grid of elements and select test functions from a space of continuous, piecewise polynomial functions given by

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$$\mathcal{D}(\Omega) = \{u \in C^0(\Omega) \mid u|_{\Omega^e} \in \mathbb{P}_k(\Omega^e), \forall\ \Omega^e \in \mathcal{T}(\Omega)\},$$

where $\mathbb{P}_k(\Omega^e)$ denotes an appropriate polynomial space for each element type. This comes from using the definition of a basis of polynomial functions $\phi_p : \Omega_{\text{st}} \to \mathbb{R}$ within reference elements $\Omega_{\text{st}} \subseteq [-1, 1]^d$. For example a reference triangle is defined as

$$\Omega_{\text{st}} = \{(\xi_1, \xi_2) \mid \xi_1, \xi_2 \in [-1, 1],\ \xi_1 + \xi_2 \leq 0\},$$

where ξ is used to denote a coordinate position within Ω_{st}. Subsequently we choose a basis $\phi_p(\xi)$ which form a basis of the polynomial space

$$\mathbb{P}_P(\Omega_{\text{st}}) = \text{span}\{\xi_1^p \xi_2^q \xi_3^r \mid (\xi_1, \xi_2, \xi_3) \in \Omega^e, (pqr) \in \mathcal{I}\},$$

where the index set $\mathcal{I}$ defines the spanning polynomial space for each element type. For the elements we consider here, these are given by

$$\begin{aligned}
\mathcal{I}^{\text{quad}} &= \{(pqr) \mid 0 \leq p \leq P,\ 0 \leq q \leq Q,\ r = 0\} \\
\mathcal{I}^{\text{tri}} &= \{(pqr) \mid 0 \leq p \leq P,\ 0 \leq p + q \leq Q,\ r = 0,\ P \leq Q\} \\
\mathcal{I}^{\text{hex}} &= \{(pqr) \mid 0 \leq p \leq P,\ 0 \leq q \leq Q,\ 0 \leq r \leq R\} \\
\mathcal{I}^{\text{pri}} &= \{(pqr) \mid 0 \leq p \leq P,\ 0 \leq q \leq Q,\ 0 \leq p + r \leq P,\ P \leq R\} \\
\mathcal{I}^{\text{tet}} &= \{(pqr) \mid 0 \leq p \leq P,\ 0 \leq p + q \leq Q,\ 0 \leq p + q + r \leq R,\ P \leq Q \leq R\}
\end{aligned}$$

with P, Q and R defining a possibly heterogeneous polynomial order for each coordinate direction. Finally, $N(P, e) = |\mathcal{I}|$ defines the number of local degrees of freedom contained within each element.

In order to construct a discrete representation $u^\delta \in \mathcal{D}(\Omega)$ of the scalar function u, we follow a standard approach and construct a sub- or iso-parametric polynomial mapping $\chi^e : \Omega_{\text{st}} \to \Omega^e$, using the same selection of basis functions ϕ_p, to define both the world-space coordinates $x \in \Omega^e$ of a given element, as well as the elemental shape functions $\phi_p^e = \phi_p \circ (\chi^e)^{-1}$ and the corresponding polynomial space $\mathbb{P}_k(\Omega^e)$. These shape functions can then be used within expansions (alongside appropriate projections) to construct discrete representations u^δ of the form

$$u^\delta(x) = \sum_{e=1}^{|\mathcal{T}(\Omega)|} \sum_{n=1}^{N(P,e)} \hat{u}_n^e \phi_n^e(x). \tag{2.2}$$

For a C^0 basis, appropriate conditions need to be imposed on $\hat{u}_p^e$ to ensure continuity of the global basis functions, typically through the use of an appropriate global assembly operation [28]. This can be viewed as a nonsquare matrix-vector multiplication by an assembly matrix $\mathcal{A}$, so that the vector of all element local coefficients $\boldsymbol{u}_l$ and the corresponding global modes of the system $\boldsymbol{u}_g$ are connected through the relationship $\boldsymbol{u}_g = \mathcal{A}\boldsymbol{u}_l$ and $\boldsymbol{u}_l = \mathcal{A}^T \boldsymbol{u}_g$.

Finally then, proceeding in a standard fashion and recalling the use of homogeneous Neumann boundary conditions, we multiply (2.1) by a test function $v \in \mathcal{D}(\Omega)$, integrate both sides, and apply the divergence theorem to arrive at the weak form

$$(\nabla u, \nabla v)_\Omega + \lambda(u, v)_\Omega = -(f, v)_\Omega,$$

where

$$(u, v)_\Omega = \int_\Omega u(x) v(x)\, dx.$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Substituting an expansion for u and v of the form of (2.2) and considering a single element of the mesh, the left-hand side leads to the of a discrete Helmholtz matrix $\boldsymbol{H}^e$ with

$$
\begin{aligned}
[\boldsymbol{H}^e]_{pq} &= \int_{\Omega^e} \lambda \phi_p^e(x)\phi_q^e(x) + \nabla\phi_p^e(x)\cdot\nabla\phi_q^e(x)\,dx \\
&= \int_{\Omega_{\mathrm{st}}} \left[\lambda\phi_p(\xi)\phi_q(\xi) + (\boldsymbol{J}^e)^{-1}\nabla\phi_p(\xi)\cdot(\boldsymbol{J}^e)^{-T}\nabla\phi_q^e(\xi)\right]|\boldsymbol{J}^e|\,d\xi, \qquad (2.3)
\end{aligned}
$$

where $\boldsymbol{J}^e$ is the corresponding Jacobian matrix of the coordinate transformation $\chi^e(\xi)$.

2.1. Operator evaluation. Obtaining solutions to (2.1) relies on finding solutions to the linear system $\boldsymbol{H}\hat{\boldsymbol{u}}_g = \boldsymbol{f}$, where $\boldsymbol{f}$ denotes the vector of coefficients resulting from the projection of the forcing function $f(x)$ onto the global modes that span $\mathcal{D}(\Omega)$, $\hat{\boldsymbol{u}}_g$ is a corresponding vector of global coefficients for the unknown solution, and $\boldsymbol{H} = \mathcal{A}[\bigoplus_{e=1}^{N_{\mathrm{el}}} \boldsymbol{H}^e]\mathcal{A}^T$ the globally assembled Helmholtz matrix. In parallel execution, where memory is distributed across multiple computational nodes, the explicit construction of $\boldsymbol{H}$ is generally infeasible. Solutions to this system are therefore commonly found using iterative Krylov-type solvers, and in this specific case, the symmetry of the system leads to the preconditioned conjugate gradient method being a popular choice [43]. The action of the matrix-vector multiplication $\boldsymbol{H}\hat{\boldsymbol{u}}_g$ is therefore evaluated in a manner which gives the same mathematical outcome but without the explicit construction of the matrix. The main costs in solving this system can therefore be attributed to the time spent in evaluation of $\boldsymbol{H}$; communication costs in the distributed assembly of the operator and the reductions necessary for iterative methods; and preconditioner performance that governs the number of iterations required for convergence. In this paper, we only consider the effect of operator evaluation, since for large problems this is frequently the dominant cost in computation [43]. Evaluation of $\boldsymbol{H}$ can be performed in a number of ways, each of which yield different performance characteristics:

- Assembly of a process-local sparse matrix $\boldsymbol{H}$ combined with a distributed assembly. In the past this has been demonstrated to yield good performance at lower polynomial orders [40].
- Assembly of local, dense, elemental matrices $\boldsymbol{H}^e$. The direct sum $\bigoplus_{e=1}^{N_{\mathrm{el}}} \boldsymbol{H}^e$ may be evaluated element-by-element using a series of dense linear algebra routines from, for example, BLAS (e.g., `dgemv`). This can then be combined with a process-local and distributed assembly operation to evaluate $\mathcal{A}$, thereby evaluating the action of $\boldsymbol{H}$ without explicitly constructing the full global system.
- Optionally, elemental matrices can be combined with *static condensation*, in which degrees of freedom are associated with either the boundary or interior of the element. A Schur complement technique is then applied to solve a system comprising of the "shell" of degrees of freedom lying on the boundary. This is then coupled with an embarrassingly parallel solve for interior degrees of freedom on each element, which can be precomputed for additional performance. As this system is considerably smaller than that arising from the full element, particularly in three dimensions owing to favorable surface-to-volume ratios, this can result in substantial cost savings, particularly when combined with a suitable preconditioner (see [17]). However, it does require the choice of a local basis that admits a boundary/interior decomposition; common choices such as Lagrange basis with appropriate nodes admit this decomposition, but,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

e.g., the Legendre basis does not. Furthermore, the tensor-product structure for general elements is lost under this operation, meaning that local matrix generation is required. The exception to this is for Cartesian quadrilateral and hexahedral elements, which can use appropriate factorizations to recover this structure [19].

- Finally, where the local elemental basis is constructed from a tensor product of one-dimensional functions, the sum-factorization technique can be applied to construct a *matrix-free* operator evaluation, to explicitly preclude the construction of $\boldsymbol{H}^e$. As an example, given a one-dimensional basis $\phi_p(\xi)$ for a segment, a basis for a quadrilateral can be formed as $\phi_{pq}(\xi_1, \xi_2) = \phi_p(\xi_1)\phi_q(\xi_2)$. Evaluation of an expansion at a given point can then be represented as

$$u(\xi_1, \xi_2) = \sum_{p=0}^{P} \sum_{q=0}^{P} \hat{u}_{pq} \phi_p(\xi_1) \phi_q(\xi_2) = \sum_{p=0}^{P} \phi_p(\xi_1) \left[\sum_{q=0}^{P} \hat{u}_{pq} \phi_q(\xi_2) \right],$$

  where the brackets denote the use of a temporary storage. At a given dimension d, and considering a tensor product of quadrature or solution points that require evaluation, this technique substantially reduces operator evaluations from $\mathcal{O}(P^{2d})$ to $\mathcal{O}(P^{d+1})$. We note that in this matrix-free setting, preconditioning poses a problem as many traditional techniques (e.g., algebraic multigrid or incomplete LU) typically rely on the presence of a globally assembled sparse matrix. However, for elliptic problems, the use of p-multigrid techniques for high-order simulations is particularly prevalent in the literature at present (see, e.g., [25]). By using a pointwise Jacobi-type smoother, performant preconditioning can be achieved in a matrix-free manner.

The relative performance of these approaches, specifically on modern hardware, has been considered previously in separate work (e.g., [26]) but only for quadrilateral and hexahedral elements that readily admit the definition of a tensor-product basis. Although in theory these element types can be used in arbitrary complex geometries, the generation of unstructured hexahedral and quadrilateral meshes is presently an open problem. In this paper, we therefore aim to consider the effectiveness of this matrix-free evaluation in the context of simplicial-type elements such as triangles, tetrahedra, and prisms, which more readily align with current mesh generation capabilities. To do this requires the selection of a basis permitting tensor product decomposition, which we discuss in the following section.

2.2. Choice of polynomial basis. The selection of the polynomial basis on each element is a key consideration of this paper. Much of the prior work considered in section 1 exploits the use of a tensor product of one-dimensional nodal Lagrange basis functions, where on the standard segment $[-1, 1]$, these are defined as

$$\ell_p(\xi) = \prod_{\substack{0 \le q \le P \\ q \neq p}} \frac{\xi - \hat{\xi}_q}{\hat{\xi}_p - \hat{\xi}_q}, \tag{2.4}$$

where $\hat{\xi}_q \in [-1, 1]$ denote a set of $P+1$ points. Frequently, these are chosen to align or collocate with an underlying quadrature (e.g., Gauss or Gauss–Lobatto points). This "classical" nodal spectral element approach yields the performance benefit of trivial interpolations, making the mass matrix diagonal (albeit without an exact integration of its entries) and reducing the cost of Helmholtz operator evaluations. Although this approach can readily be extended to higher dimensional tensor-product elements,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

a formulation of these basis functions inside hybrid or simplicial elements such as triangles and tetrahedra leads to a set of basis functions that lack the tensor product structure required to enable the use of sum factorization. More details on this approach can be found in, e.g., [18].

To arrive at a tensor product formulation, we follow standard practice [20] and employ the use of a square-to-triangle Duffy transformation [12] to define two independent coordinate directions over which to perform the decomposition (or otherwise use other similar mappings, e.g., [36]). This process is shown in Figure 1. Analytically, for a triangle this mapping is defined as

$$\eta_1 = 2\frac{1+\xi_1}{1-\xi_2} - 1, \quad \eta_2 = \xi_2.$$

Although this mapping introduces a singularity, this can be mitigated (without a loss of convergence order) using an appropriate choice of quadrature, as we outline in section 2.3. Multiple applications of this transformation can be used to arrive at similar coordinate spaces for higher dimensional elements. The evaluation of integrals such as (2.3) then takes place on the collapsed coordinate space, leading to a double application of the chain rule so that

$$\nabla u^\delta(x) = \boldsymbol{G}\boldsymbol{J}^e \nabla u^\delta(\eta).$$

Specifically, for the elements we consider here, we have that

$$\mathbf{G}_{\text{tri}} = \begin{bmatrix} \dfrac{2}{1-\eta_2} & \dfrac{1+\eta_1}{1-\eta_2} \\ 0 & 1 \end{bmatrix}, \quad \mathbf{G}_{\text{prism}} = \begin{bmatrix} \dfrac{2}{1-\eta_3} & 0 & \dfrac{1+\eta_1}{1-\eta_3} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{G}_{\text{tet}} = \begin{bmatrix} \dfrac{4}{(1-\eta_2)(1-\eta_3)} & \dfrac{2(1+\eta_1)}{(1-\eta_2)(1-\eta_3)} & \dfrac{2(1+\eta_1)}{(1-\eta_2)(1-\eta_3)} \\ 0 & \dfrac{2}{1-\eta_3} & \dfrac{1+\eta_2}{1-\eta_3} \\ 0 & 0 & 1 \end{bmatrix}.$$

The integrals in (2.3) are then evaluated over the collapsed coordinates η as opposed to standard coordinates ξ.

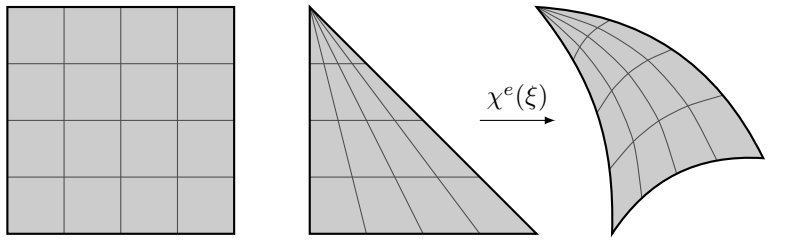


FIG. 1. *Illustration of the three coordinate systems used in the representation of a high-order triangle. A representative equally spaced distribution of points is used to highlight the distribution of quadrature points in the resulting high-order element.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

With these coordinates in place, we may define two common choices of basis functions which permit tensor-product decompositions and that we will consider in this work. The first are Dubiner-type basis functions [11], which form an orthogonal modal basis on the elemental space. For example, the triangular Dubiner basis is of the form

$$\psi_{pq}(\eta) = \underbrace{\sqrt{2}P_p^{(0,0)}(\eta_1)}_{\psi_p^a(\eta_1)} \underbrace{P_q^{(2p+1,0)}(\eta_2)(1-\eta_2)^p}_{\psi_{pq}^b(\eta_2)},$$

where $P^{(\alpha,\beta)}$ denotes the standard Jacobi polynomial and the indices p and q lie in the indexing set $\mathcal{I}^{\text{tri}}$. It is clear that, under a different indexing strategy where the limits on inner summations now rely on outer summations, this basis can be decomposed into the product of two one-dimensional functions for η_1 and η_2, respectively. For example the interpolation of a expansion at a point η can be expressed as

$$u^\delta(\eta) = \sum_{p=0}^{P} \psi_p^a(\eta_1) \left[\sum_{q=0}^{Q-p} \hat{u}_{pq} \psi_{pq}^b(\eta_2) \right].$$

Triangular prisms and tetrahedra admit similar decompositions.

The second choice of tensor product basis is that presented in [37] and [20] and used in Nektar++, in which "standard" linear finite element modes which are 1 at each vertex and linearly decay to zero at other vertices, are augmented with a set of orthogonal interior high-order polynomials. This leads to a basis that is not strictly orthogonal within across the entire basis for each element but allows a natural route to impose C^0 connectivity and a separation into boundary and interior modes. For completeness, this basis and its use in the various elemental expansions is defined in Appendix A.

2.3. Discrete evaluation of the Helmholtz operator. To discretely evaluate (2.1), a final point of concern is that of quadrature, as the evaluation of the weak Helmholtz matrices requires an integration over each element. With the selection of an appropriate quadrature we obtain

$$[\boldsymbol{H}]_{pq} \approx \sum_{n=1}^{N_Q} \nabla\phi_p(\eta^n) \boldsymbol{J}^e(\eta^n)^{-1} |\boldsymbol{J}^e(\eta^n)| w_n \boldsymbol{J}^e(\eta^n)^{-T} \nabla\phi_q(\eta^n), \tag{2.5}$$

where η^n denotes the distribution of quadrature points and w_n the corresponding weights. For tensor-product elements, common choices of quadrature include Gauss–Lobatto points to allow accurate and fast integration. This naturally incorporates tensor product orderings of quadrature points and thereby enables sum-factorization to be used in either forward projections to polynomial space or interpolation from polynomial to physical space.

On the other hand, the other element types do not typically utilize such quadrature, instead opting for cubature-type rules that lose this structure, such as those seen in [18]. In the formulation of a collapsed coordinate η, however, we may opt to use a similar distribution of Gauss points in each collapsed coordinate direction. Indeed, an appropriate choice of quadrature in the direction of the collapsed coordinate also permits us to effectively deal with the singularity that occurs as a function of collapsing vertices. Typically this is accomplished with appropriately weighted Gauss–Radau points, which exclude the endpoints corresponding to the collapsed vertices. This also allows us to use one fewer integration point in this direction, owing to the increased

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

accuracy of integration compared to Gauss–Lobatto points. An illustrated distribution of quadrature in the resulting world-space element, which describes this process for a triangle and the effects of the Duffy transform, can be viewed in Figure 1.

Finally, one further choice that must be made is the number of quadrature points to select in each coordinate direction. As noted previously, for the classical Lagrange interpolants of (2.4), one might opt to select $Q = P + 1$ quadrature points, so as to recover a diagonal mass matrix. For more complex models such as the Navier–Stokes equations, higher orders of quadrature are frequently used to exactly integrate nonlinear terms (such as the convection operator) or highly curvilinear elements where $\boldsymbol{J}^e$ is a now a high-order polynomial [23, 29]. In this work, we opt to exactly integrate the mass matrix but omit any over-integration effects, so that we select $Q = P + 2$ Gauss–Lobatto points in coordinate directions that are not collapsed, and $Q = P + 1$ Gauss–Radau points in coordinate directions that are collapsed.

3. Matrix-free SIMD implementation. The main challenge in designing efficient tensor-product matrix-free algorithms for simplicial elements is the increased complexity of data layout and indexing as opposed to standard tensor-product elements. For example, tetrahedral expansions are represented by the summation

$$u^{\delta}(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \sum_{r=0}^{R-p-q} \hat{u}_{pqr} \phi_p^a(\eta_1) \phi_{pq}^b(\eta_2) \phi_{pqr}^c(\eta_3). \tag{3.1}$$

The full form of summations for each element type is given in Appendix A. Although the dependency of indexes here is clearly more complex than is seen in the hexahedron, and will likely result in performance degradation owing to this property, our aim is to quantify this and determine whether highly performant implementations are still attainable in a simplicial element setting.

In this section, we give a brief overview of our implementation choices which we believe permits an efficient evaluation of the discrete Helmholtz operator from (2.5) for unstructured elements. The main result of this work is to consider this implementation in section 4, where it will be evaluated on different architectures of varying SIMD widths.

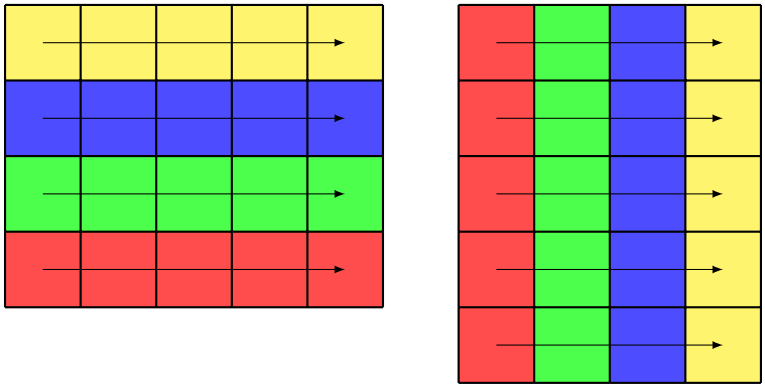
3.1. Data layout and SIMD strategy. The implementation of the matrix-free problem in (2.5) is done in a standalone benchmarking utility for the Helmholtz operator, although the initial construction of basis data, their derivatives, quadrature points, and weights as well as other ancillary functions such as mesh connectivity and parallelization is performed using the *Nektar++* framework [8, 31]. The main purpose of this utility is to examine the use of explicit SIMD instructions in order to achieve optimal performance for kernels which evaluate the Helmholtz operator. As the name implies, these instructions allow more than one data entry to be operated on (through, e.g., multiplication or addition) during a single CPU cycle. On modern hardware, this typically takes the form of 4 or 8 FLOPS in a single cycle using either 256-bit or 512-bit advanced vector instructions (AVX), denoted as AVX2 and AVX512, respectively. Furthermore, the use of one or more fused-multiply add (FMA) units, which combine the multiplication and addition operation $a{\cdot}b{+}c$ into a single cycle, further enhance the potential FLOPS available. In order to attain the maximum peak performance of these architectures, codes must be written using these important instruction sets in mind.

The precise way in which SIMD can be used in finite element formulations has been the consideration of various previous studies. Broadly, SIMD may be applied in three different ways:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

- Assuming the data regarding expansion coefficients are stored in an element-by-element ordering, we may choose to iterate over either 4 or 8 degrees of freedom at a time and load them into a vector register. The main drawback for this method is that the number of degrees of freedom is rarely divisible by the vector width, and so padding must be used for each element. This approach can be seen in [4, 5].
- We may alternatively choose to combine element data into groups corresponding to the vector width of the architecture, as is seen in, e.g., [14, 13, 26, 24, 2] and visualised in Figure 2. For example, on an AVX2 machine with a 256-bit vector width corresponding to 4 double-precision floating point numbers, we may group 4 elements so that their data are interleaved in memory. In this case, no padding per element is required; however, if the number of elements is not divisible by the vector width then a small and indeed negligible degree of padding will be required to mask the missing elements.
- Alternatively, if considering problems involving the function u and its three-dimensional gradient ∇u, as would appear in, e.g., the DG method, the four components $(u, \partial_x u, \partial_y u, \partial_z u)$ can be loaded into a single AVX register. However the limiting factor here is the restriction to three-dimensional DG, as well as the need to combine this approach with these previously mentioned in order to capitalize on wider vector widths such as AVX512.



(a) contiguous layout (b) interleaved layout

FIG. 2. *Contiguous vs. interleaved memory layout for a group of four elements, denoted by different coloured blocks. Arrows denote the memory storage direction.*

For simplicial-type elements, the interdependence of mode indices in each coordinate direction means that the first choice above is vastly more difficult than the second and, in all likelihood, more expensive, since each contraction would require a different amount of padding as a function of the mode number. Moreover, since we wish to make use of a C^0 basis, rather than a DG setting in which derivatives are desirable, as well as to consider three-dimensional problems and the use of AVX512, it is clear that the third choice here is not desirable either. We therefore adopt the second approach, where we interleave element data into groups corresponding to the vector width of the architecture. The same strategy is used to interleave storage of basis data and associated structures on the standard element such as quadrature points and weights. This allows us to act on the group of elements simultaneously in, for example, interpolation of the polynomial data at some point in an element,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

as well as other storage required for, e.g., the Jacobian $\boldsymbol{J}^e$ and its determinant. The difference between a "standard" contiguous ordering of element degrees of freedom and this interleaved ordering is highlighted in Figure 2.

We additionally note that several performance optimizations can be made depending on whether the reference-to-world mapping χ^e is affine or nonlinear. In the former case, elements are planar-faced and parallelepipeds in the case of hexahedral/prismatic/quadrilateral elements, whereas the latter gives the flexibility of curvature to adapt to an underlying geometry. However, the affine case leads to elemental Jacobian matrices $\boldsymbol{J}^e$ that are constant, meaning that the d^2 entries and its determinant can be stored once per element. In the nonlinear case, we store these entries at each quadrature point, meaning that far more memory is required to store the mapping data in this case. The benchmarking code considers these two cases separately and optimizes through the use of both template programming and appropriate pointer arithmetic to reduce the memory footprint and/or FLOPS required in kernel evaluations. We refer to these two cases as *regular* and *deformed*, respectively, in the discussion below. We note that in some previous studies, further optimization can be made for Cartesian formulations, as each element has the same Jacobian determinant. Since here we consider generally unstructured meshes where this will not be the case, we do not consider this approach in this work.

In terms of other memory considerations, we also note that adjustments were made throughout the code to ensure that any allocated memory are aligned to the appropriate cache line sizes, to ensure the use of more optimal aligned vector loading instructions. Additionally, basis functions, which can be thought of mathematically as entries in a two dimensional array $\boldsymbol{B}$ where $[\boldsymbol{B}]_{ij} = \phi_i(\xi_j)$ are stored in one dimensional flat arrays, indexed by both mode and quadrature point. The index over quadrature point is always the fastest index. This was done to align with the memory access pattern of the inner product kernel, described in the subsequent section, which is the most frequently computed kernel within the Helmholtz operator. For basis functions comprising multiple directions, such as the Dubiner basis ϕ_{pq}, the last index is stored first so as to align with the layout of elemental degrees of freedom and ensure contiguous memory access.

3.2. Programming considerations. We make a number of deliberate design decisions in the implementation which are noted in this section. As noted in [2, 5] and elsewhere, autovectorization of code using a compiler is still a relatively difficult problem. Reliance on a compiler alone to generate efficient SIMD code for even relatively simple loop structures is therefore not usually possible. This is an important factor in performance, since clearly as many AVX and FMA instructions as possible are required to ensure good CPU utilization. Additionally, however, performance penalties are incurred when frequent transitions are made between different kinds of vector and legacy instructions are mixed; the density of these instructions can therefore be maintained through the use of intrinsics.

To this end, our benchmarking code is written using C++ and makes use of compiler intrinsics to ensure the consistent and dense use of vector instructions. Template programming, alongside operator overloading and a custom data type that encapsulates the vector width of the processor and common vector operations on such data, is used to produce accessible code without the need to call intrinsics functions directly but retain performance. The vector data structure contains explicit support for FMA instructions to enable these to be used where possible. Decompilation of the resulting

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

object code was performed to ensure that no additional instructions were inserted due to the use of this technique.

Further to this, templates were used more generally within kernels to improve the compiler's ability to unroll loops and handle the complex structure of the nested loops of the form of (3.1) occurring within the operator evaluations. Specifically, template parameters include the polynomial order and number of quadrature points in each coordinate direction, the vector width being used and whether the element is deformed or regular. In testing, we found the use of templating on these parameters gave significant 20% to 30% improvements in execution times for more complex loop structures found in triangles, prisms, and tetrahedra. To enhance usability and preclude the necessity for recompilation due to a change in polynomial order, a jump table is used to select precompiled kernels from a range of common polynomial orders between $1 \leq P \leq 10$.

As noted in [32], the Helmholtz operator can be decomposed into a combination of three routines:

- `BwdTrans`: performs a backwards transformation (polynomial interpolation) onto the physical space, defined by (2.2), given the elemental coefficients $\hat{\boldsymbol{u}}$ and standard basis functions $\boldsymbol{B}$;
- `InnerProduct`: calculates the L^2 inner product $(\cdot,\cdot)_\Omega$ given a vector $\boldsymbol{u}$ denoting the function at a set of coordinates in physical space, along with quadrature weights $\boldsymbol{w}$ and the basis (or its derivatives).
- `TensorDerivative`: calculates the partial derivatives of a polynomial expansion at physical points $\boldsymbol{u}$, as represented on a tensor-product of Gauss quadrature points, given the one-dimensional derivative matrix $\boldsymbol{D}$ and appropriate mapping derivatives $(\boldsymbol{J}^e)^{-1}$.

The combination of these operations can be seen in Algorithm 3.1 for the Helmholtz operator. In the subsequent section, we will also consider their performance as standalone operators, as examined in [32, 40, 9].

3.3. Correction for C^0 modified basis. As a final remark on implementation, we give a brief note on the consideration of an effect of the tensor product storage of the modified C^0 basis, as described in [20]. This choice of basis relies on an additional correction step in evaluating the modes of an expansion corresponding to any vertex or edge mode that have been collapsed under the Duffy transform. As an example, consider a linear expansion of the C^0 basis on a triangle, which is given by

$$\sum_{p=0}^{1}\sum_{q=0}^{1-p}\hat{u}_{pq}\phi_{pq}(\eta) = \sum_{p=0}^{1}\sum_{q=0}^{1-p}\hat{u}_{pq}\phi_p^a(\eta_1)\phi_{pq}^b(\eta_2).$$

Furthermore, we have that

$$\phi_0^a(\eta_1) = \frac{1-\eta_1}{2}, \quad \phi_1^a(\eta_1) = \frac{1+\eta_1}{2}, \quad \phi_{01}^b(\eta_1) = \frac{1+\eta_2}{2}.$$

The first two modes in this expansion recover the standard linear finite element modes for a triangle; namely,

$$\phi_{00}(\eta) = \frac{\xi_1-\xi_2}{2}, \quad \phi_{10}(\eta) = \frac{1+\xi_1}{2}.$$

However the mode corresponding to the collapsed vertex under this combination, where $p=0$ and $q=1$, is given by

$$\phi_{01}(\eta) = \frac{1-\eta_1}{2}\frac{1+\eta_2}{2} \neq \frac{1+\xi_2}{2}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 3.1. Overview of the matrix free evaluation of the Helmholtz operator.

procedure HELMHOLTZ($\hat{\boldsymbol{u}}, w, \boldsymbol{B}, \boldsymbol{\nabla B}, \boldsymbol{D}, (\boldsymbol{J}^e)^{-1}, |\boldsymbol{J}^e|$)
  for each element group **do**
    $\boldsymbol{u} \leftarrow$BWDTRANS($\hat{\boldsymbol{u}}, \boldsymbol{B}$)
    $\boldsymbol{out} \leftarrow \lambda\cdot$INNERPRODUCT($\boldsymbol{u}, \boldsymbol{B}, w, |\boldsymbol{J}^e|$)
    $\boldsymbol{Du} \leftarrow$TENSORDERIVATIVE($\boldsymbol{u}, \boldsymbol{D}, (\boldsymbol{J}^e)^{-1}$)
    if element is deformed **then**
      for each quadrature point $\hat{\xi}^n$ **do**
        calculate Helmholtz metric $\boldsymbol{M} = (\boldsymbol{J}^e)^{-1}(\boldsymbol{J}^e)^{-T}$
        $\boldsymbol{Du}[n] \leftarrow \boldsymbol{MDu}[n]$
      end for
    else
      calculate element constant metric $\boldsymbol{M} = (\boldsymbol{J}^e)^{-1}(\boldsymbol{J}^e)^{-T}$
      for each quadrature point $\hat{\xi}^n$ **do**
        $\boldsymbol{Du}[n] \leftarrow \boldsymbol{MDu}[n]$
      end for
    end if
  end for
  for each dimension d **do**
    $\boldsymbol{out} \leftarrow \boldsymbol{out}+$INNERPRODUCT($\boldsymbol{Du}_d, \partial_d \boldsymbol{B}, w, |\boldsymbol{J}^e|$)
  end for
end procedure

as desired for a linear finite element mode. The reason is that there is a missing contribution arising from the collapsed vertex where $\eta_1 = \eta_2 = 1$. By constructing an additional mode from the tensor product basis that corresponds to this contribution, namely,

$$\frac{1+\eta_1}{2}\frac{1+\eta_2}{2},$$

we can correct for this omission to modify the mode so that

$$\phi_{01}(\eta) = \left(\frac{1-\eta_1}{2} + \frac{1+\eta_1}{2}\right)\frac{1+\eta_2}{2} = \frac{1+\xi_2}{2}.$$

The `BwdTrans` and `InnerProduct` kernels therefore need to be modified in order to take this correction step into account. For a triangle, this amounts to a single correction of the mode corresponding to the top vertex. For the prismatic element, this is required at all modes corresponding to the top collapsed edge. Finally, for the tetrahedron, this is required for the two collapsed vertices as well as a collapsed edge. Naturally, these corrections add additional FLOPS to the overall evaluation but have the potential to be masked by memory accesses if they occur within a particularly memory-bound regime. In the subsequent section, we will determine the overhead in terms of throughput of degrees of freedom that is incurred by the choice of this basis.

4. Results. In this section, we perform hardware tests of the SIMD Helmholtz implementation of the previous section. After describing the hardware used for testing, as well as our methodology for the tests, we will consider three methods of evaluation of the Helmholtz kernel. The first will observe the throughput of evaluation, i.e., the number of degrees of freedom that can be processed by the Helmholtz kernel per second. We will then aim to more rigorously quantify the performance

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

of the kernel in terms of the percentage of peak performance through a roofline and attained GFLOPS/s analysis. Finally, we consider the effects of the C^0 basis correction described in the previous section.

4.1. Hardware and test methodology. Tests have been conducted on two CPU models with varying SIMD vector widths in order to evaluate the effectiveness of the implementations across hardware architectures. An outline of the key hardware characteristics can be found in Table 1. In particular we consider tests on both a Broadwell architecture, with a 256-bit AVX2 SIMD (8 DP FLOPS/cycle) and a Skylake architecture, with a 512-bit AVX512 SIMD (16 DP FLOPS/cycle). Both CPUs in these tests support two FMA units, doubling their FLOPS/cycle count to 16 and 32, respectively. Only a single node is considered in this work, and execution is performed using solely MPI parallelization.

TABLE 1
Specifications for Intel CPUs used for testing of SIMD evaluation.

Model	**Xeon E5-2697 v4**	**Xeon Gold 6130**
Architecture	Broadwell	Skylake
SIMD width	256 bit	512 bit
Standard clock speed	2.3 GHz	2.1 GHz
AVX2 clock speed	2.0 GHz	1.7 GHz
AVX512 clock speed	-	1.3 GHz
L3 cache size	46080 KB	22528 KB
Cores per socket	18	16
Sockets per node	2	2
Max node GFLOPS/s (AVX)	1152	870
Max node GFLOPS/s (AVX512)	-	1331
Peak memory bandwidth	110 GB/s	-

For each element type under consideration, structured meshes of the domain $\Omega = [0, 1]^d$ were generated using the Gmsh mesh generation software [16]. The number of elements generated is designed to be divisible by both the number of cores on each node as well as the vector width of the architecture. In this manner, data are not padded to align to vector widths and furthermore when running across all cores the mesh is perfectly partitioned and therefore each core theoretically performs equal computational work. In order to examine the balance between memory bandwidth and floating point computations at different polynomial orders, mesh sizes were also adjusted to ensure that the memory footprint of all elemental degrees of freedom exceed the L3 cache size reported in Table 1. On Broadwell nodes, `gcc` version 6.3.0 was used to compile the code using the compiler flags `-march=native -funroll-loops` and run using OpenMPI 2.0.2. On Skylake nodes, we use the `gcc` compiler version 6.4.0 and run using Intel MPI 2018. Power governors for the machines were set to `performance` mode to ensure optimal performance, and MPI ranks were pinned to individual cores to ensure memory allocation in the appropriate NUMA region and prevent excessive memory transfer between sockets. As well as ensuring the dedicated use of these nodes, all tests results below report average times for around 1 minute of kernel executions per polynomial order and element type in order to mitigate any underlying machine noise or lag due to other processes.

4.2. Throughput analysis. In this first section, we consider the performance of the Helmholtz kernels from the perspective of throughput, i.e., the number of degrees of freedom processed by the kernels per second of execution. All of the tests in this

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

section consider only the modified C^0 basis. However, as we do not consider the effects of full C^0 connectivity here (i.e., in the assembly of the overall system), in this sense "degrees of freedom" can be equated with the total number of local degrees of freedom at a given polynomial order P, i.e.,

$$N_{\text{dof}} = \sum_{e=1}^{N_{\text{el}}} N(P, e). \tag{4.1}$$

Figure 3 outlines the throughput analysis for both two- and three-dimensional elements on the Broadwell architecture. A number of immediate trends can be identified.

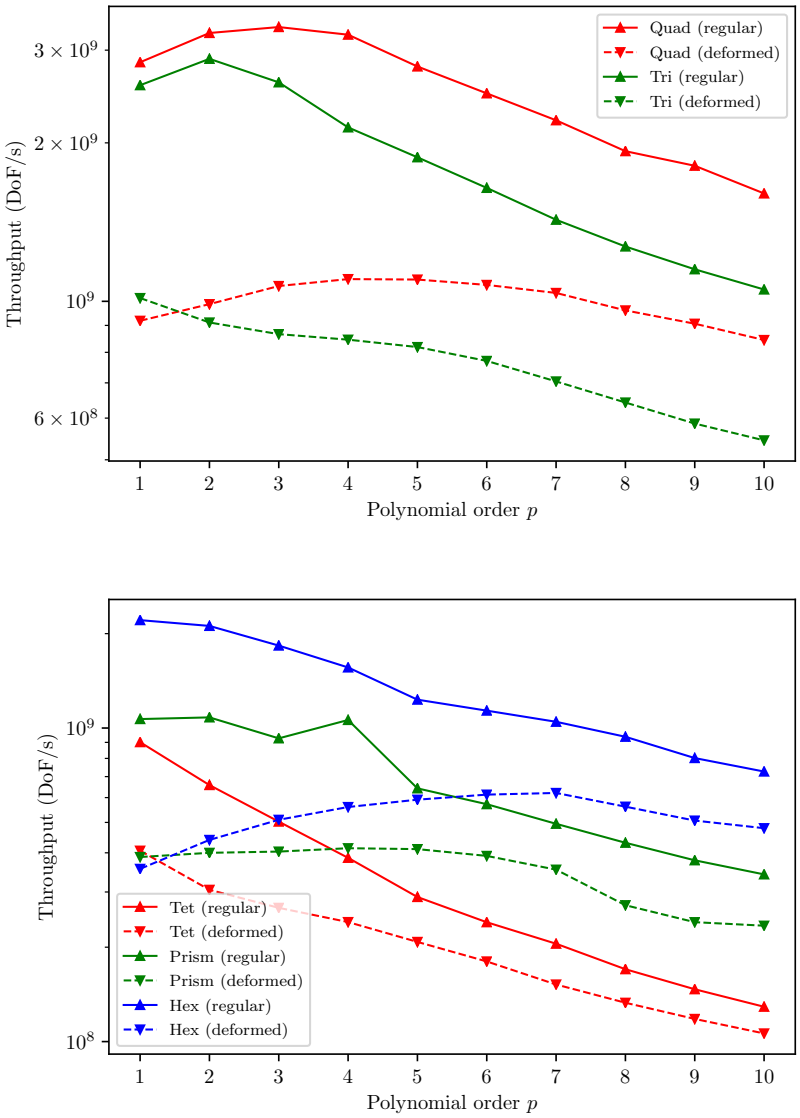


FIG. 3. *Throughput analysis of the Helmholtz implementation for two-dimensional (top) and three-dimensional (bottom) elements on the Broadwell E5-2697v4 processor, for regular and deformed geometries.*

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

In terms of regular elements, which have the least amount of memory required for the operator evaluations, there is a clear hierarchy of performance between dimensions and element type. Two-dimensional elements outperform three-dimensional, and overall the "naturally" tensor product quadrilateral and hexahedral elements outperform their simplicial counterparts. The prismatic element, which has greater tensor product structure than the tetrahedron, owing to the single direction of collapsed coordinates, sees this represented through a moderate performance increase over the tetrahedron. There is a further clear trend in the data for regular elements, in that throughput monotonically decays as polynomial order is increased. This is indicative that the kernels are performant mostly in a FLOPS-bound regime: i.e., the increase in polynomial order is matched by a corresponding increase in computational work and thus a decrease in throughput.

On the other hand, deformed curvilinear elements exhibit a richer spectrum of performance characteristics. One immediately clear conclusion is that the introduction of curvature into the kernel leads to a significant drop in throughput, as can be anticipated from the increased memory storage required to represent the curvilinear mapping χ^e and its spatially variable Jacobian $\boldsymbol{J}^e$. This is a characteristic observed in other matrix-free implementations such as [26]. However, at moderate polynomial orders, there is generally either a mild increase in throughput from the linear order, or at least a reduction in the rate of decrease compared against their regular element counterparts. This is indicative that these operations are more memory bound, as the lack of decrease in throughput indicates the increased computational work is masked by memory transfer. The exception to this is at higher polynomial orders, where the rate of decrease matches regular elements, indicating that the increased computational effort has now pushed these cases into a more FLOPS-bound regime.

4.2.1. Skylake AVX512 throughput. To examine the effect of SIMD vector width on performance, we also consider throughput on the Skylake architecture. In particular, this architecture supports both 256- and 512-bit SIMD through AVX2 and AVX512 instruction sets. In theory and neglecting implementation-specific issues, the change from 256-bit to 512-bit SIMD implies a theoretical doubling of available FLOPS and potential for doubling of throughput. Observing this rise in throughput should therefore point towards the efficiency of implementation. Importantly, however, we note that on Skylake processors this doubling of performance is not possible, as the AVX512 base clock speed (1.3 GHz) is considerably reduced in comparison to the AVX2 base clock speed (1.7 GHz) by around 25%, so as to ensure the processor sits within the intended thermal envelope. This leads to a total theoretical performance increase of around 53%, as seen from the processor statistics in Table 1.

Figure 4 shows the relative throughput of AVX2 versus AVX512 on the Skylake hardware for both regular and deformed three-dimensional elements. For regular elements, where the regime is mostly FLOPS-bound, a ratio of 40% to 45% improvement can be seen across most polynomial orders. Deformed elements, on the other hand, exhibit less improvement at lower polynomial orders, presumably due to the increased memory footprint of this regime. However, at higher orders, similar performance increases can be observed as in the regular element case. The conclusions from these figures therefore is that kernel efficiency appears to be high; however, in the next section we will confirm this assertion through a more rigorous roofline model analysis.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

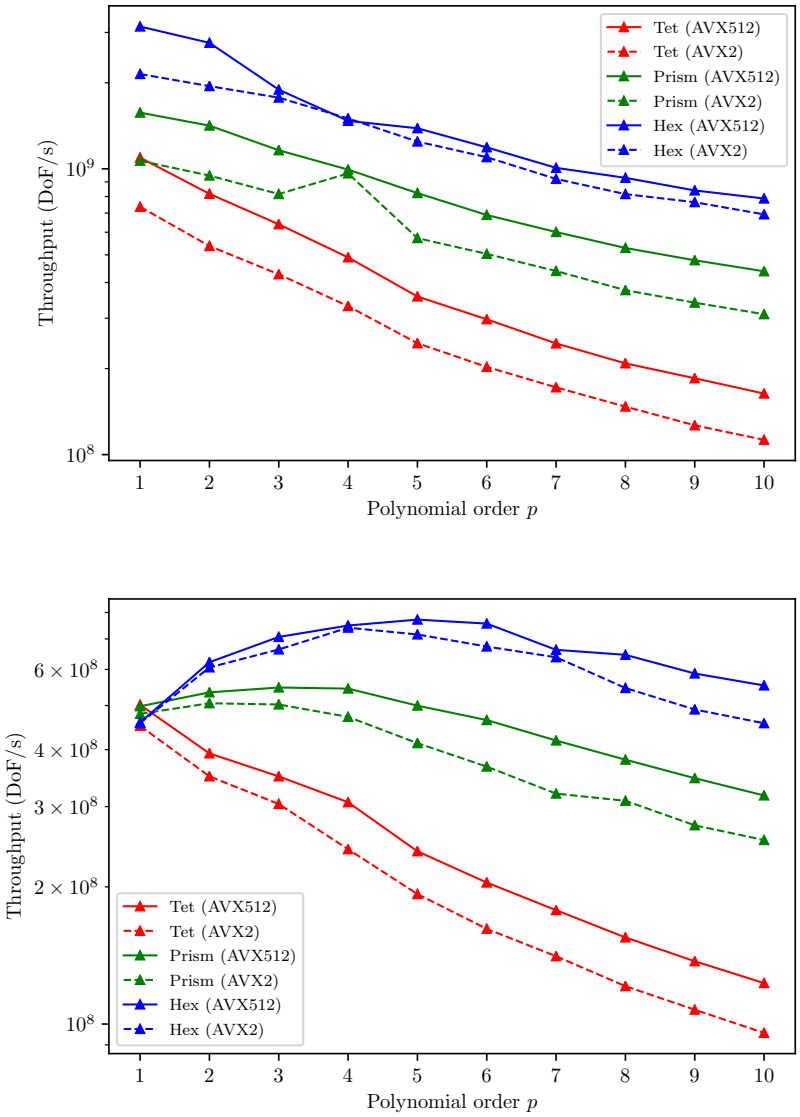


FIG. 4. *Throughput comparison of the Helmholtz kernel for three-dimensional elements, comparing the use of AVX versus AVX*512 *instructions for regular (top) and deformed (bottom) element types.*

4.2.2. Comparison against other work. In terms of determining the general implementation efficiency, one possibility is to try to compare throughput results against other studies in the literature, since these are generally reported in most other work in this area. We note that the general order of magnitude of $\sim 10^9$ DoF/second in terms of throughput attained for quadrilaterals and hexahedral elements on Broadwell hardware is very similar to that seen in other existing studies such as [26]. However, specific comparison to other work in the literature is complex due to the rich variety of choices in terms of implementation and numerical setup that can be considered

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

on a study-by-study basis, as well as the specific hardware under consideration. For example, the aforementioned work can capitalize on the symmetricity of Lagrangian basis functions defined on Gauss–Lobatto points to reduce computational work (the "even-odd" decomposition), which we do not consider here. Additionally, most other work in this area considers only the matrix-free evaluation of the Laplacian operator, which has one fewer mass-matrix evaluation than the Helmholtz operator. In our experiments, evaluation of the Laplacian leads to around a 10% to 20% increase in throughput, so again can be seen to be roughly similar to existing work in this area.

4.3. Roofline analysis. To more adequately quantify the performance of the kernel implementation in relation to the theoretical peak performance of the hardware, in this section we consider an analysis of the computational performance by examining the commonly-used roofline performance model [41]. This model considers that the two main performance bottlenecks in algorithmic implementation are FLOPS/s and memory bandwidth. By considering arithmetic intensity α of the algorithm as the main independent variable, one can then consider a "roofline" of performance defined by

$$\text{Max GFLOPS/s} = \min(\text{peak GFLOPS/s},\ \text{peak memory bandwidth} \times \alpha).$$

Although this model has limitations, such as the lack of modelling of CPU caching effects, generally it is recognized as a straightforward and visual way through which to judge the potential of algorithms in attaining full utilization of the available FLOPS in relation to memory bandwidth limits.

To capture performance data required for this model, namely, FLOPS/s and memory bandwidth, we make use of the Likwid performance monitoring and benchmarking suite [38], which uses either per-core or socket-based "uncore" hardware counters to determine key performance characteristics. Tests were performed using Likwid version 4.3.0 and the `MEM_DP` performance group used to record memory bandwidth and GFLOPS/s attained using the `likwid-mpirun` utility for parallel execution. Likwid was also used to determine the peak memory bandwidth of the computational node, recorded in Table 1, using `likwid-bench` with the `stream_mem_avx_fma` test.

The results of this analysis are presented in Figure 5. This figure displays recorded arithmetic intensity against GFLOPS/s for each shape at a range of polynomial orders between $1 \leq P \leq 10$. A general trend between all simulations, which is not presented on the figure for clarity, is a steady increase in arithmetic intensity as polynomial order is increased, so that marker points from left to right generally denote simulations at increasing polynomial order. From the figure, it is evident that in all cases there is a clear distinction here between the FLOPS-bound regular elements and the memory-bound deformed elements, which was highlighted in the previous section in terms of throughput trends.

Furthermore, these roofline models allow us to firmly validate the efficiency of the implementation. In the case of deformed elements, simulations are close to the memory-bandwidth imposed roofline, aside from at higher polynomial orders where the simulations tend to become more FLOPS-bound. Similarly, for regular elements, the results clearly indicate that simulations are running at FLOPS counts that require heavy use of vectorization and consistent use of FMA, particularly in two dimensions. We also note that, when not limited by memory bandwidth, FLOPS counts are typically within 50% to 70% of the peak attainable. To highlight this, we consider the GFLOPS/s attained in Helmholtz evaluation of a regular tetrahedron in Figure 6 as a

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

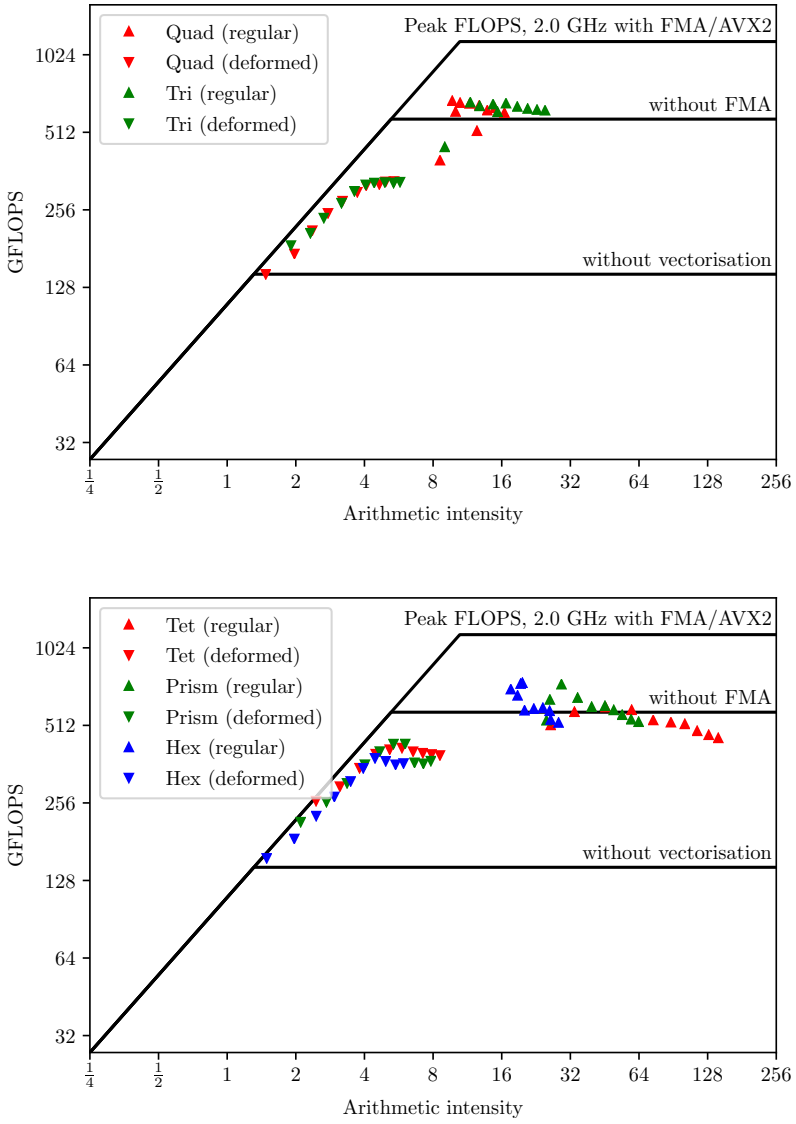


FIG. 5. *Roofline analysis of the Helmholtz implementation for two-dimensional (top) and three-dimensional (bottom) elements on the Broadwell E5-2697v4 processor, for both regular and deformed geometries.*

typical example of performance in this regime. This figure additionally highlights the same trend of increased performance through use of AVX512 instructions observed in the previous section and Figure 4.

Returning to the roofline plots, another clear observation is that simplicial elements result in broadly larger arithmetic intensity as the element type is made more compact; that is, tetrahedra offer greater arithmetic intensity than prisms, and prisms offer greater arithmetic intensity than hexahedra. Although it may appear somehow

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

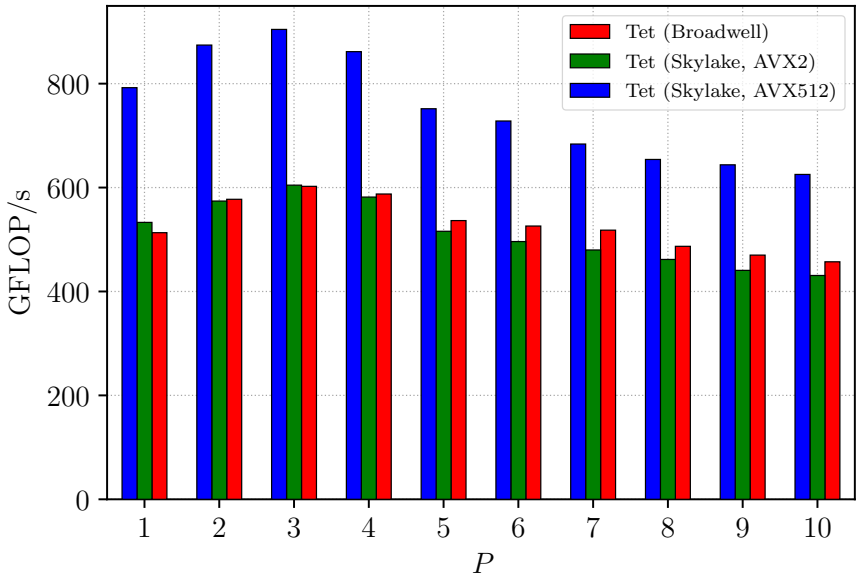


FIG. 6. *GFLOPS/s of Helmholtz operator between the three architectures under consideration for regular tetrahedral elements.*

counterintuitive to observe this trend, the overall memory footprint of these elements reduces as the element is made more compact. At the same time, for the C^0 basis, the use of repeated Duffy transformations leads to more corrections in tetrahedra than prisms and thus offers the opportunity to increase FLOPS further for the same memory transfer.

4.4. Effect of correction in modified C^0 basis. As a final consideration for implementation of tensor-product kernels for simplicial elements, in this section we examine the use of the orthogonal Dubiner-type basis in place of the C^0 basis. It is worth emphasizing that the orthogonal basis is frequently used in popular DG formulations for various systems of equations, which further generalize the findings of this study beyond simply the C^0 formulation.

Figure 7 highlights the difference in throughput between the two choices of basis. As can be expected, it is clear that the use of the orthogonal basis results in a higher throughput of degrees of freedom, as no additional corrections of the form of section 3.3 are required. Further to this, as the number of modes that require correction increases as we move from triangle to prism to tetrahedron, so too does the gap in performance.

In Figure 8, we consider the effects of the orthogonal basis in the context of the roofline model. For simplicity of observation, we consider only regular elements and omit quadrilateral and hexahedral elements. Comparing against the results from the modified C^0 basis in Figure 5, it is clear that the peak FLOPS attained is roughly comparable (or slightly higher) than that previously observed. However, at the same time, arithmetic intensity has been somewhat reduced in this regime, as viewed by the distinct shift to the left of all results, owing to the lack of correction step required in this basis.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

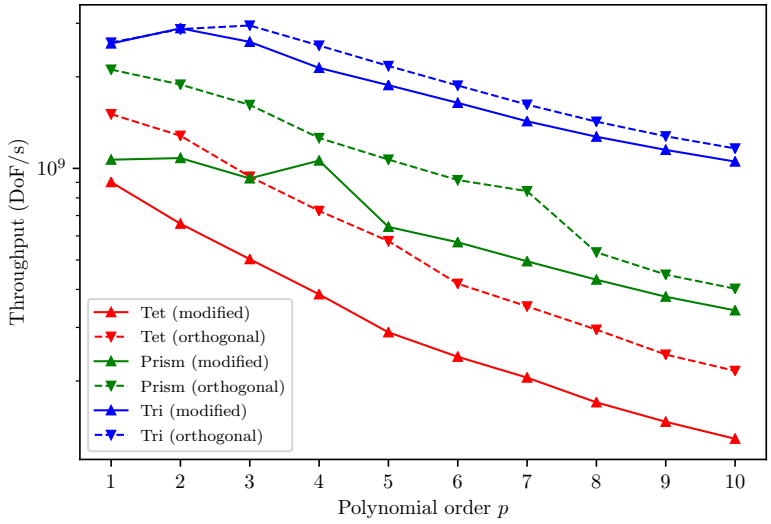


FIG. 7. *Throughput analysis to examine the correction step used within the C^0 basis, against the orthogonal basis in which no correction is necessary.*

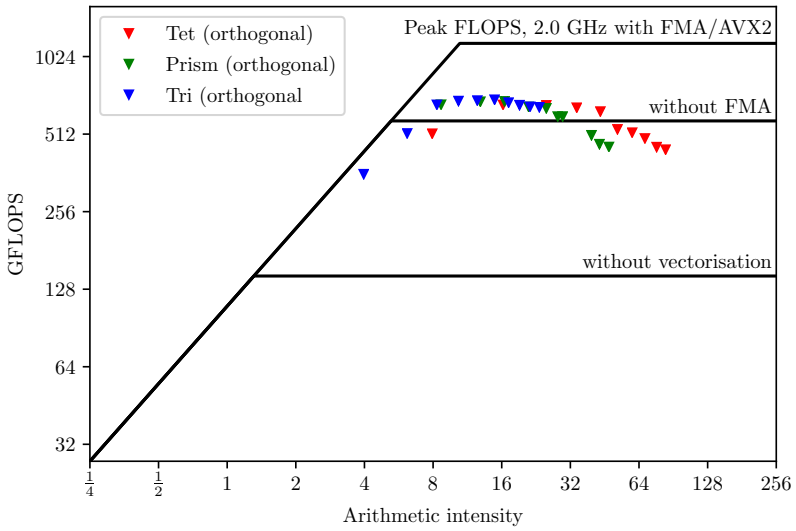


FIG. 8. *Roofline analysis of the Helmholtz implementation for simplicial elements on the Broadwell E5-2697v4 processor using an orthogonal basis.*

4.5. Effect of C^0 assembly operation. All of the results of the previous sections have focused on the implementation of elementally-local kernels; i.e., without the consideration of how basis functions across elements are connected as part of the appropriate CG or DG function space. In this section, we perform a further series of tests to incorporate the effects of global assembly in a C^0 setting, so that

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 4.1. Overview of the evaluation of the global Helmholtz operator.

procedure GLOBALHELMHOLTZ($\hat{\boldsymbol{u}}_g, w, \boldsymbol{B}, \boldsymbol{\nabla B}, \boldsymbol{D}, (\boldsymbol{J}^e)^{-1}, |\boldsymbol{J}^e|, \mathcal{A}$)
  $\hat{\boldsymbol{v}}_g \leftarrow 0$ ▷ zero resulting output vector $\hat{\boldsymbol{v}}_g$
  for each element group e **do**
    $\hat{\boldsymbol{u}}_l \leftarrow \mathcal{A}^T \hat{\boldsymbol{u}}_g[e]$ ▷ global-to-local into small, reused temporary storage $\hat{\boldsymbol{u}}_l$
    $\hat{\boldsymbol{u}}_l \leftarrow$HELMHOLTZ($\hat{\boldsymbol{u}}_l, w, \boldsymbol{B}, \boldsymbol{\nabla B}, \boldsymbol{D}, (\boldsymbol{J}^e)^{-1}, |\boldsymbol{J}^e|$)
    $\hat{\boldsymbol{v}}_g \leftarrow \hat{\boldsymbol{v}}_g + \mathcal{A}\hat{\boldsymbol{u}}_l$ ▷ assembly of local contributions from group e
  end for
  $\hat{\boldsymbol{v}}_g \leftarrow$GSMPI($\hat{\boldsymbol{v}}_g, \mathcal{A}$) ▷ perform parallel assembly
end procedure

basis functions are forced to be continuous across elemental boundaries. This is imposed through the local-to-global mapping $\mathcal{A}$. Since this operation incurs a memory indirection, we expect to see a reduction in throughput and achieveable FLOPS.

In order to examine this effect, the benchmarking utility leverages the graph partitioner SCOTCH [10] to perform the initial domain decomposition, so as to give each processor equal work and minimize communication costs between MPI ranks. Additionally, for assembly across processors, we use the `gslib` gather-scatter library (described briefly in [30]) which is part of the Nek5000 spectral element solver [15]. The global Helmholtz operator is then performed in the manner described in Algorithm 4.1.

To highlight the performance effects arising from the assembly operator, we perform the same throughput and roofline tests as in the previous sections. Throughput is measured using the same definition as before; i.e., we measure the number of local degrees of freedom processed per second, so as to provide a common baseline for comparison between the two schemes. Figure 9 shows the results from these tests compared against the local elemental operators measured in the previous section. As expected, all element types incur a reduction in throughput due to the additional cost of assembly. Although there is a higher relative cost for assembly at lower polynomial orders, likely owing to the larger number of elements and higher average node valencies in this regime, there is a fairly constant drop in throughput across polynomial order and element type, indicating that assembly costs are comparable between polynomial orders. This is expected since, in these tests, mesh sizes vary with polynomial order so as to keep the number of degrees of freedom relatively constant, meaning that communication and memory indirection costs will be roughly equivalent. This trend is also reflected in the roofline modeling, where all element types see a reduction in the FLOPS obtained, as well as a shift to the left in terms of arithmetic intensity. Both of these features can be attributed to the additional memory bandwidth required for assembly.

5. Conclusions. In this paper, we have presented formulations of the Helmholtz operator evaluation for simplicial elements that leverages explicit SIMD instructions, sum-factorisation and tensor-product basis functions to achieve near-peak performance across a range of both polynomial orders and AVX2 and AVX512 hardware types. Formulations such as these are highly important in the context of simulation across complex geometries, for which the generation of all-hexahedral meshes is still an open challenge.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

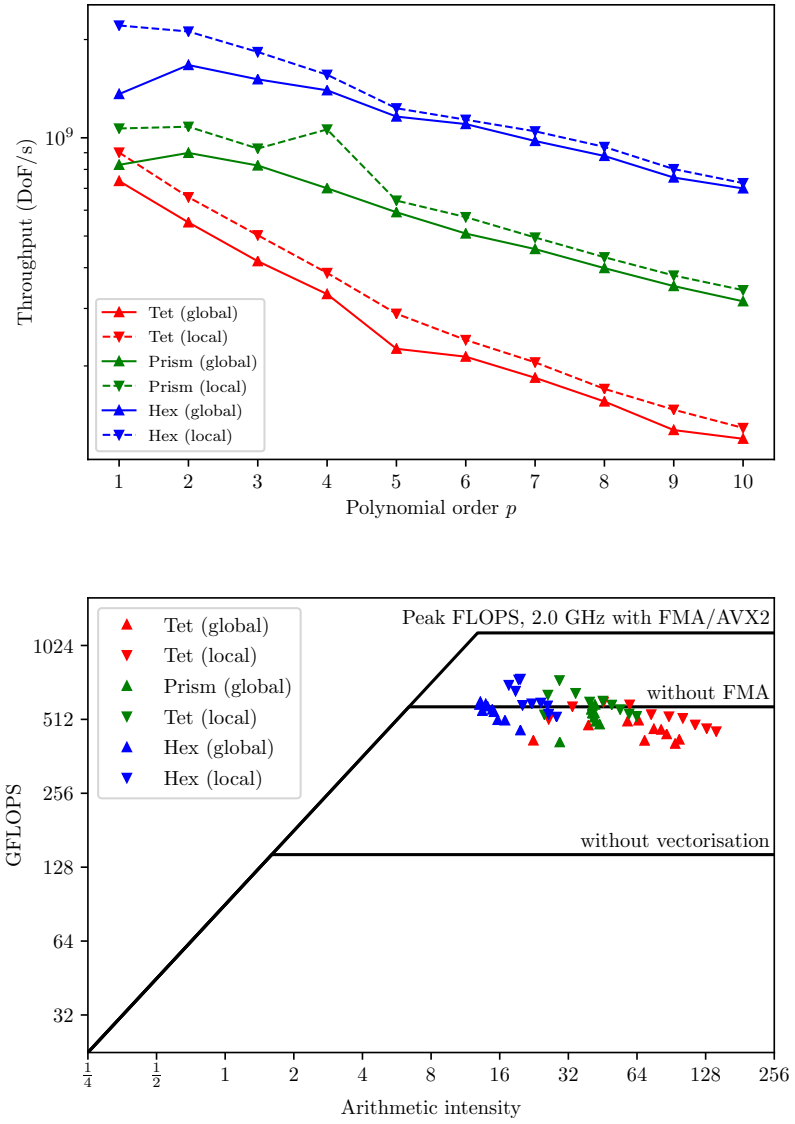


FIG. 9. *Throughput (top) and roofline (bottom) analysis of the Helmholtz implementation for three-dimensional elements on the Broadwell E5 − 2697v4 processor for global C^0 operator evaluations.*

It is clear and inevitable that simplicial element types will struggle to attain the same performance of the more naturally tensor-product structure of quadrilateral and hexahedral elements. With this said, in the worst case of regular elements which have no curvature, prismastic and triangular elements in particular suffer a reduction of throughput by only a factor of around 1.5 to 2 when compared against hexahedra and quadrilaterals, respectively, depending on the polynomial order. Tetrahedra inevitably suffer a more significant drop in throughput by a factor of around 5 to 8. This is in line with expectations: since we use a square or cube of (collapsed) quadrature

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

points for all element types, and our meshes contain two or six times the number of elements for triangles/prisms and tetrahedra, respectively, we see that evaluations costs of a triangle are roughly equivalent to a quadrilateral, and a prism or tetrahedron equivalent to a hexahedron.

On the other hand, deformed elements present a wider range of performance characteristics, owing to a larger memory footprint due to the storage of a nonlinear Jacobian $\boldsymbol{J}^e$ matrix at each quadrature point. Here, at low to moderate polynomial orders, the difference in performance between the element types is considerably less pronounced. In all cases, roofline modeling of the implementation has demonstrated the high performance of resulting kernels as being within 50% and 70% of peak attainable FLOPS across all architectures under consideration.

We note that the performance of prismatic elements is important in particular, since for complex geometries in, e.g., fluid dynamics, meshes tend to be generated using a combination of prismatic elements in the boundary layer, and tetrahedra in the remainder of the interior [34]. The presence of a boundary layer comprising prismatic elements can account for between 30% and 50% of the total element count, indicating the potential for significant performance increases against standard full-system or static-condensation solves.

Further to this, we have demonstrated that the choice of basis (between the modified C^0 basis and orthogonal Dubiner basis) does impact on performance due to the correction steps required for the C^0 basis. However, this reduction in throughput does not necessarily paint the full picture. In the context of CG, this performance gap is likely recovered by the increased complexity of imposing continuity through the assembly mapping $\mathcal{A}$ for the Dubiner basis. In a DG setting, unlike the orthogonal basis, the C^0 basis offers a boundary-interior decomposition which makes the introduction of face flux terms straightforward. On the other hand, additional computational cost is required in order to impose this flux term on all degrees of freedom of the orthogonal basis, which does not have the boundary-interior property.

Finally, we have examined the additional costs incurred in a global C^0 assembly operation. Our tests indicate that there is an expected reduction in performance across element types and polynomial orders. We note that the use of structured grids in this study represents a more favorable case for the simplicial element types, where for complex geometries one may find much larger node valencies, requiring a higher level of memory indirection and therefore further reductions in the observed throughput. However, we do see that the trends observed in the local operator are still seen in this regime.

In this light, there is a clear direction to pursue further research in this area. In particular, there is the possibility to consider the application in a wider variety of use cases, with a particular focus on elliptic solvers for the incompressible Navier–Stokes equations but under a number of different discretization choices in order to examine the resulting performance characteristics. Other areas of potential study include the generalization of these results to other many-core architectures; in particular, the use of graphics processing units in which vector widths are considerably larger than the AVX512 instructions considered here.

Appendix A. Modified C^0 basis.

The modified C^0 basis for the elements we consider in this work are defined using three tensor product bases:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

$$\phi_p^a(\xi) = \begin{cases} \frac{1-\xi}{2} & p = 0 \\ \frac{1+\xi}{2} & p = 1 \\ \left(\frac{1-\xi}{2}\right)\left(\frac{1+\xi}{2}\right) P_{p-1}^{1,1}(\xi) & 2 \le p < P \end{cases}$$

$$\phi_{pq}^b(\xi) = \begin{cases} \phi_q(\xi) & p = 0 \quad 0 \le q < P \\ \left(\frac{1-\xi}{2}\right)^p & 1 \le p < P \quad q = 0 \\ \left(\frac{1-\xi}{2}\right)^p \left(\frac{1+\xi}{2}\right) P_{q-1}^{2p-1,1}(\xi) & 1 \le p < P, \quad 1 \le q < P - p \end{cases}$$

$$\phi_{pqr}^c(\xi) = \begin{cases} \phi_{qr}(\xi) & p = 0 \quad 0 \le q < P \\ & 0 < r < P - q \\ \left(\frac{1-\xi}{2}\right)^{p+q} & 1 \le p < P, \quad 0 \le q < P - p, \quad r = 0 \\ \left(\frac{1-\xi}{2}\right)^{p+q} \left(\frac{1+\xi}{2}\right) P_{r-1}^{2p+2q-1,r}(\xi) & 1 \le p < P, \quad 0 \le q < P - p, \\ & 1 \le r < P - p - q, \end{cases}$$

where P, Q, and R denote the polynomial order in each coordinate direction, and $P_p^{\alpha,\beta}(\xi)$ is the standard Jacobi polynomial. Elemental expansions then take the form

$$u_{\text{quad}}^\delta(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \hat{u}_{pq} \phi_p^a(\xi_1) \phi_q^a(\xi_2)$$

$$u_{\text{tri}}^\delta(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \hat{u}_{pq} \phi_p^a(\eta_1) \phi_{pq}^b(\eta_2)$$

$$u_{\text{hex}}^\delta(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \sum_{r=0}^{R} \hat{u}_{pqr} \phi_p^a(\xi_1) \phi_q^a(\xi_2) \phi_r^a(\xi_3)$$

$$u_{\text{prism}}^\delta(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \sum_{r=0}^{R-p} \hat{u}_{pqr} \phi_p^a(\eta_1) \phi_q^a(\eta_2) \phi_{pr}^b(\eta_3)$$

$$u_{\text{tet}}^\delta(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \sum_{r=0}^{R-p-q} \hat{u}_{pqr} \phi_p^a(\eta_1) \phi_{pq}^b(\eta_2) \phi_{pqr}^c(\eta_3),$$

where η denote appropriate collapsed coordinates, and the above are amended with appropriate corrections as noted in [20] and section 3.3.

Acknowledgments. We would like to thank M. Kronbichler for a number of helpful discussions.

REFERENCES

[1] M. Ainsworth, G. Andriamaro, and O. Davydov, *Bernstein–Bézier finite elements of arbitrary order and optimal assembly procedures*, SIAM J. Sci. Comput., 33 (2011), pp. 3087–3109.

[2] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler, *Algorithms and data structures for massively parallel generic adaptive finite element codes*, ACM Trans. Math. Softw., 38 (2011), 14.

[3] W. Bangerth, R. Hartmann, and G. Kanschat, *Deal.* II– *a general-purpose object-oriented finite element library*, ACM Trans. Math. Softw., 33 (2007), 24.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[4] P. Bastian, C. Engwer, J. Fahlke, M. Geveler, D. Göddeke, O. Iliev, O. Ippisch, R. Milk, J. Mohring, and S. Müthing, *Hardware-based Efficiency Advances in the EXA-DUNE Project*, in Software for Exascale Computing-SPPEXA 2013-2015, Springer, New York, 2016, pp. 3–23.

[5] P. Bastian, C. Engwer, D. Göddeke, O. Iliev, O. Ippisch, M. Ohlberger, S. Turek, J. Fahlke, S. Kaulmann, and S. Müthing, *EXA-DUNE: Flexible PDE Solvers, Numerical Methods, and Applications*, in European Conference on Parallel Processing, Springer, New York, 2014, pp. 530–541.

[6] P. Bastian, F. Heimann, and S. Marnach, *Generic implementation of finite element methods in the distributed and unified numerics environment (DUNE)*, Kybernetika, 46 (2010), pp. 294–315.

[7] C. Burstedde, L. C. Wilcox, and O. Ghattas, *P4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM J. Sci. Comput., 33 (2011), pp. 1103–1133.

[8] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. de Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin, *Nektar++: An open-source spectral/hp element framework*, Comput. Phys. Commun., 192 (2015), pp. 205–219.

[9] C. D. Cantwell, S. J. Sherwin, R. M. Kirby, and P. H. J. Kelly, *From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements*, Comput. Fluids, 43 (2011), pp. 23–28.

[10] C. Chevalier and F. Pellegrini, *PT-Scotch: A tool for efficient parallel graph ordering*, Parallel Comput., 34 (2008), pp. 318–331.

[11] M. Dubiner, *Spectral methods on triangles and other domains*, J. Sci. Comput., 6 (1991), pp. 345–390.

[12] M. G. Duffy, *Quadrature over a pyramid or cube of integrands with a singularity at a vertex*, SIAM J. Numer. Anal., 19 (1982), pp. 1260–1262.

[13] N. Fehn, W. A. Wall, and M. Kronbichler, *Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows*, Int. J. Numer. Methods Fluids, 88 (2018), pp. 32–54.

[14] N. Fehn, W. A. Wall, and M. Kronbichler, *Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows*, J. Comput. Phys., 372 (2018), pp. 667–693.

[15] P. F. Fischer, *An overlapping Schwarz method for spectral element solution of the incompressible Navier–Stokes equations*, J. Comput. Phys., 133 (1997), pp. 84–101.

[16] C. Geuzaine and J.-F. Remacle, *GMSH: A 3-D finite element mesh generator with built-in pre-and post-processing facilities*, Int. J. Numer. Methods Engrg., 79 (2009), pp. 1309–1331.

[17] L. Grinberg, D. Pekurovsky, S. J. Sherwin, and G. E. Karniadakis, *Parallel performance of the coarse space linear vertex solver and low energy basis preconditioner for spectral/hp elements*, Parallel Comput., 35 (2009), pp. 284–304.

[18] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Springer Science & Business Media, New York, 2007.

[19] I. Huismann, J. Stiller, and J. Fröhlich, *Factorizing the factorization–a spectral-element solver for elliptic equations with linear operation count*, J. Comput. Phys., 346 (2017), pp. 437–448.

[20] G. Karniadakis and S. Sherwin, *Spectral/Hp Element Methods for Computational Fluid Dynamics*, Oxford University Press, Oxford, 2013.

[21] G. E. Karniadakis, M. Israeli, and S. A. Orszag, *High-order splitting methods for the incompressible Navier-Stokes equations*, J. Comput. Phys., 97 (1991), pp. 414–443.

[22] R. C. Kirby, *Fast simplicial finite element algorithms using Bernstein polynomials*, Numer. Math., 117 (2011), pp. 631–652.

[23] R. M. Kirby and G. E. Karniadakis, *De-aliasing on non-uniform grids: Algorithms and applications*, J. Comput. Phys., 191 (2003), pp. 249–264.

[24] B. Krank, N. Fehn, W. A. Wall, and M. Kronbichler, *A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow*, J. Comput. Phys., 348 (2017), pp. 634–659.

[25] M. Kronbichler and W. A. Wall, *A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers*, preprint, arXiv:1611.03029, 2016.

[26] M. Kronbichler and W. A. Wall, *A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers*, SIAM J. Sci. Comput., 40 (2018), pp. A3423–A3448.

[27] J.-E. W. Lombard, D. Moxey, S. J. Sherwin, J. F. A. Hoessler, S. Dhandapani, and M. J. Taylor, *Implicit large-eddy simulation of a wingtip vortex*, AIAA J., 54 (2016), pp. 506–518.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[28] G. R. Markall, A. Slemmer, D. A. Ham, P. H. J. Kelly, C. D. Cantwell, and S. J. Sherwin, *Finite element assembly strategies on multi-core and many-core architectures*, Int. J. Numer. Methods Fluids, 71 (2013), pp. 80–97.

[29] G. Mengaldo, D. de Grazia, D. Moxey, P. E. Vincent, and S. J. Sherwin, *Dealiasing techniques for high-order spectral element methods on regular and irregular grids*, J. Comput. Phys., 299 (2015), pp. 56–81.

[30] K. Mittal, S. Dutta, and P. Fischer, *Nonconforming Schwarz-spectral element methods for incompressible flow*, Comput. Fluids, 191 (2019), 104237.

[31] D. Moxey, C. D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, G. Mengaldo, D. Serson, M. Turner, H. Xu, J. Peiró, R. M. Kirby, and S. J. Sherwin, *Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods*, Comput. Phys. Commun., (2019), 107110.

[32] D. Moxey, C. D. Cantwell, R. M. Kirby, and S. J. Sherwin, *Optimizing the performance of the spectral/hp element method with collective linear algebra operations*, Comput. Methods Appl. Mech. Engrg., 310 (2016), pp. 628–645.

[33] D. Moxey, D. Ekelschot, Ü. Keskin, S. J. Sherwin, and J. Peiró, *High-order curvilinear meshing using a thermo-elastic analogy*, Computer-Aided Design, 72 (2016), pp. 130–139.

[34] D. Moxey, M. D. Green, S. J. Sherwin, and J. Peiró, *An isoparametric approach to high-order curvilinear boundary-layer meshing*, Comput. Methods Appl. Mech. Engrg., 283 (2015), pp. 636–650.

[35] A. Peplinski, P. F. Fischer, and P. Schlatter, *Parallel performance of h-type Adaptive Mesh Refinement for Nek*5000, in Proceedings of the Exascale Applications and Software Conference 2016, ACM, 2016, 4.

[36] M. D. Samson, H. Li, and L.-L. Wang, *A new triangular spectral element method* I*: Implementation and analysis on a triangle*, Numer. Algorithms, 64 (2013), pp. 519–547.

[37] S. J. Sherwin and G. E. Karniadakis, *A new triangular and tetrahedral basis for high-order (hp) finite element methods*, Int. J. Numer. Methods Engrg., 38 (1995), pp. 3775–3802.

[38] J. Treibig, G. Hager, and G. Wellein, *Likwid: A lightweight performance-oriented tool suite for x*86 *multicore environments*, in Parallel Processing Workshops (ICPPW), 2010 39th International Conference On, IEEE, 2010, pp. 207–216.

[39] M. Turner, J. Peiró, and D. Moxey, *Curvilinear mesh generation using a variational framework*, Computer-Aided Design, 103 (2018), pp. 73–91.

[40] P. E. Vos, S. J. Sherwin, and R. M. Kirby, *From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low-and high-order discretisations*, J. Comput. Phys., 229 (2010), pp. 5161–5181.

[41] S. Williams, A. Waterman, and D. Patterson, *Roofline: An insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), pp. 65–76.

[42] F. D. Witherden, A. M. Farrington, and P. E. Vincent, *PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach*, Comput. Phys. Commun., 185 (2014), pp. 3028–3040.

[43] S. Yakovlev, D. Moxey, S. J. Sherwin, and R. M. Kirby, *To CG or to HDG: A comparative study in* 3*D*, J. Sci. Comput., 67 (2016), pp. 192–220.

Downloaded 07/13/20 to 138.195.214.230. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.