J Sci Comput (2010) 45: 48–63
DOI 10.1007/s10915-010-9396-8

Efficient Nonlinear Solvers for Nodal High-Order Finite Elements in 3D

Jed Brown

Received: 1 June 2009 / Revised: 7 March 2010 / Accepted: 23 June 2010 / Published online: 23 July 2010

Abstract Conventional high-order finite element methods are rarely used for industrial problems because the Jacobian rapidly loses sparsity as the order is increased, leading to unaffordable solve times and memory requirements. This effect typically limits order to at most quadratic, despite the favorable accuracy and stability properties offered by quadratic and higher order discretizations. We present a method in which the action of the Jacobian is applied matrix-free exploiting a tensor product basis on hexahedral elements, while much sparser matrices based on Q_1 sub-elements on the nodes of the high-order basis are assembled for preconditioning. With this "dual-order" scheme, storage is independent of spectral order and a natural taping scheme is available to update a full-accuracy matrix-free Jacobian during residual evaluation. Matrix-free Jacobian application circumvents the memory bandwidth bottleneck typical of sparse matrix operations, providing several times greater floating point performance and better use of multiple cores with shared memory bus. Computational results for the p-Laplacian and Stokes problem, using block preconditioners and AMG, demonstrate mesh-independent convergence rates and weak (bounded) dependence on order, even for highly deformed meshes and nonlinear systems with several orders of magnitude dynamic range in coefficients. For spectral orders around 5, the dual-order scheme requires half the memory and similar time to assembled quadratic (Q_2) elements, making it very affordable for general use.

Keywords High-order · Finite element method · Newton-Krylov · Preconditioning

1 Introduction

High order spatial discretization has significant advantages over low order when high accuracy is required, especially when the solution is smooth. When the solution is only piecewise smooth, *hp* finite element methods [5, 26, 28, 30] are capable of exponential convergence, but high order discretization may yield higher quality results before asymptotic convergence

J. Brown (✉)
Versuchsanstalt für Wasserbau, Hydrologie und Glaziologie (VAW), ETH Zürich, Zürich, Switzerland
e-mail: brown@vaw.baug.ethz.ch

rates are realized. Optimal *hp* meshes are usually defined in terms of minimizing the discretization error for a given number of degrees of freedom, and effective refinement strategies (e.g. [1, 6]) have been developed relative to this metric.

Due to rapid loss of sparsity in the Jacobian under p-refinement, the total number of degrees of freedom is only weakly related to the practical performance metrics, computational time and storage requirements. For linear constant coefficient problems, this can be avoided by choosing special hierarchical bases that preserve sparsity [17]. When the spectral element method (a special case of nodal p-FEM which reuses the interpolation nodes for quadrature) is used with (nearly) affine elements, linear constant coefficient problems can be very efficiently solved using the fast diagonalization method combined with a multilevel coarse solve [22].

For nonlinear and variable coefficient problems in 3D, the element matrices are necessarily dense and fast diagonalization is not available. For order p tensor product bases on hexahedral elements, there are $n = (p + 1)^3$ degrees of freedom per element, but the dense element matrices contribute $(p+1)^6$ nonzeros to the global matrix. Since most sparse matrix operations scale linearly with the number of nonzeros, both time and memory costs scale as $\mathcal{O}(n^2)$ under p-refinement. A popular method for reducing linear algebra costs is static condensation which removes the interior degrees of freedom. This method is effective at moderate to high order in 2D because a large portion of the nodes are interior, but in 3D less than half the nodes are interior when $p < 9$. For a scalar problem with $p = 9$, condensation requires manipulating a dense 1000×1000 matrix for each element. This is very expensive in time and memory for only 1000 degrees of freedom, and still contributes a dense 488×488 matrix to the global system. An additional bottleneck of traditional *hp*-FEM is integration of element matrices. Naïve assembly of the true Jacobian using a p-point quadrature rule is $\mathcal{O}(p^9)$ although it can be improved to $\mathcal{O}(p^7)$ by exploiting the tensor product structure using sum factorization. The cost incurred by forming and manipulating these dense matrices is the primary reason why high order (even quadratic) elements are not more popular in applications where extremely high accuracy is not needed.

If extremely high accuracy is desired, conventional high-order methods are attractive because the extra cost is more than made up for by improved convergence rate in the asymptotic range. For practical simulations, the asymptotic range is typically only mildly realized because an acceptable accuracy is obtained shortly after the solution is *resolved*. This effect is illustrated in Fig. 1. High-order methods must have comparable cost *per degree of freedom* to be competitive with low-order methods unless the desired level of accuracy is well into the asymptotic range.

To develop efficient solvers for high order elements, we examine the bottlenecks present in nonlinear solvers, and design the discretization to eliminate as many of these bottlenecks as possible. Our approach to preconditioning involves the method proposed by [27] and further investigated by [7, 8, 15, 18], resulting in a scheme with storage costs equivalent to Q_1 (piecewise trilinear) elements independent of p and total CPU time which is mesh independent and only weakly dependent on p. The principle contributions of this paper are a formulation of this preconditioner that is applicable to general nonlinear systems, a matrix-free representation of the Jacobian, and observation of a much lower "crossover" order p at which this class of methods is to be preferred over conventional methods.

2 Newton-Krylov

Jacobian-free Newton-Krylov (JFNK) methods combine Newton-type methods for superlinearly convergent solution of nonlinear equations with Krylov subspace methods for solving

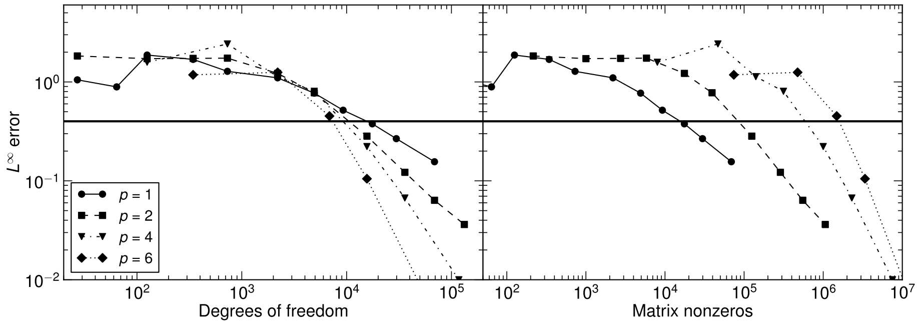

Fig. 1 The *left panel* shows truncation error as a function of total degrees of freedom for a 3D Poisson problem with smooth but rapidly varying solution. The *thick horizontal line* represents an acceptable accuracy and is only modestly within the asymptotic range for all approximation orders. The *right panel* shows the total number of nonzeros in the Jacobian which is an optimistic lower bound for the flops required for the solve

the Newton step without actually forming the true Jacobian [20]. In this section, we introduce the method and highlight the performance bottlenecks.

2.1 The Newton Iteration

The Newton iteration for the nonlinear system $F(x) = 0$ with state vector x involves solving the sequence of linear systems

$$J(x^k)s^k = -F(x^k), \quad x^{k+1} \leftarrow x^k + s^k, \quad k = 0, 1, \ldots \tag{1}$$

until convergence which is frequently measured as a relative tolerance

$$\left\| \frac{F(x^k)}{F(x^0)} \right\| < \epsilon_{\text{tol}}.$$

The structure and conditioning of the Jacobian $J(x) = \frac{\partial F(x)}{\partial x}$ is dependent on the equations and discretization, but for second order elliptic operators, the condition number $\kappa(J) = \|J\| \|J^{-1}\|$ scales as $\mathcal{O}(h^{-2}p^4)$ where h is the mesh size and p is the approximation order.

2.2 Krylov Methods

Krylov methods solve $Jx = b$ using the Krylov subspaces

$$\mathcal{K}_j = \operatorname{span}\{\tilde{b}, \tilde{J}\tilde{b}, \tilde{J}^2\tilde{b}, \ldots, \tilde{J}^{j-1}\tilde{b}\}$$

where $\tilde{J} = P^{-1}J$, $\tilde{b} = P^{-1}b_0$ for left preconditioning and $\tilde{J} = JP^{-1}$, $\tilde{b} = b$ for right preconditioning. The convergence rate of various Krylov methods (e.g. CG, MINRES, GMRES, BiCGStab, QMR) depend in diverse ways on the spectral properties of $\tilde{J}$ (see [25] for matrices where each nonsymmetric iteration beats all others by a large margin) but a useful heuristic is that the iteration count scales as the square root of $\kappa(\tilde{J})$. When solving the Newton step (1) we rely on P^{-1} to overcome the $\mathcal{O}(h^{-2}p^4)$ conditioning of the spatial discretization. Our preconditioners are obtained by applying an algorithm P to data J_p provided by the discretization of the equations, i.e. $P^{-1} = \mathsf{P}(J_p)$. Concretely, J_p almost always contains one or more assembled matrices, but may contain auxiliary information such as a multilevel hierarchy. The fundamental preconditioners P are relaxation (e.g. SOR) and (incomplete) factorization, which are used to build scalable and problem-specific preconditioners such as multigrid, domain decomposition, and Schur-complement. The notation P^{-1} comes from incomplete factorization in which $P = LU$ is available, but multiplication by P is never needed in the Krylov iteration and indeed is rarely available. A complete solver for $Jx = b$ is $\mathsf{K}(J, \mathsf{P}(J_p))$ which represents a choice of Krylov accelerator K, preconditioning method P, and preconditioning data J_p.

As discussed in the introduction, the Jacobian in (1) loses sparsity for high order methods so we will never form it explicitly. The finite difference representation

$$J(x)y = \frac{F(x + \epsilon y) - F(x)}{\epsilon}$$

for appropriately chosen ϵ is attractive because it requires no additional storage and no coding beyond function evaluation, however the adjoint J^T is not available, its accuracy is at best $\sqrt{\epsilon_{\text{machine}}}$, and it performs unnecessary computation if function evaluation involves costly fractional powers or transcendental functions. The inaccuracy can lead to stagnation due to a poor quality Krylov basis, especially when restarts are needed. Automatic differentiation (AD) offers a full accuracy alternative and can produce adjoints in reverse mode, but such evaluations, especially in reverse mode, are usually more expensive than function evaluation due to suboptimal taping[1] strategies. In Sect. 3.3 we detail an alternative with several advantages over finite differencing and which is similar to AD with optimal taping.

2.3 Performance

The most expensive parts of the Newton-Krylov iteration are assembling matrices for the preconditioner and subsequently solving for the Newton step. Most preconditioners require a setup phase after the matrices (and whatever else goes into J_p) are assembled. Scalable preconditioners require a globally coupled coarse level solve and we take multigrid as the canonical example. With geometric multigrid the coarse level can be provided by the application by rediscretizing the governing equations while algebraic multigrid must analyze the matrix structure to determine the coarse component and then compute the Galerkin coarse operator (involving relatively expensive matrix-matrix products and producing a denser coarse operator). The coarse operator needs to be factored, a task that is often done redundantly. Finally, the solve involves matrix multiplication and smoothers on each level.

The relative cost of these three phases (assembly, setup, solve) is highly problem dependent, but we offer some general guidelines for perspective; see [13, 20, 21] for more detailed analysis of these issues. Matrix assembly in finite element methods involves pointwise operations at quadrature points and is typically limited by instruction level parallelism and scheduling (especially for high-order methods). The setup and solve phases are limited by memory bandwidth on the fine levels and by network latency on the coarse levels. An important issue is simultaneously keeping the number of levels small so that relatively little work is done on the less efficient coarse levels while keeping the coarse operator small enough and sparse enough that it is cheap to factor. When strong preconditioners are used so to keep the iteration count low, setup can be much more expensive than the solve. Direct solvers are

[1] This is a slight abuse of terminology, "taping" in the AD context refers to intermediate values stored in memory (as opposed to checkpoints which are typically written to disk) during reverse-mode computation. We use the term more loosely to refer to any intermediate storage that makes multiplication by $J(u)$ and $J^T(u)$ more efficient than only storing the state vector u.

an extreme case of this, but multigrid can also have an expensive setup step, especially with many processors and Galerkin coarse operators.

Efficient linear solvers involve a tradeoff between many iterations with cheap preconditioners and fewer iterations with expensive preconditioners. With JFNK, the setup costs can be reduced by *lagging* the preconditioner (not recomputing it on every Newton step) at the expense of additional Krylov iterations. If matrix-free Jacobian application is inexpensive, the cost of these extra iterations may be affordable, making lagging appealing. On the other hand, if matrix assembly and preconditioner setup is cheap, there is no need to work with a stale preconditioner. Note that use of a stale Jacobian compromises quadratic convergence of the Newton method while a stale preconditioner only affects the convergence rate of the Krylov iteration.

Most of the time required to solve strongly nonlinear equations is spent before entering the neighborhood where Newton methods are quadratically convergent. Until the final phase, it is not important to solve the Newton step to high accuracy. In particular, $J(x^k)s^k = -F(x^k)$ may be "solved" so that

$$\left\| J(x^k)s^k + F(x^k) \right\| \le \eta^k \left\| F(x^k) \right\|$$

and the method converges q-superlinearly as long as $\eta^k \to 0$, and q-quadratically if $\eta^k \in \mathcal{O}(\|F(x^k)\|)$. Various heuristics have been developed to automatically adjust the "forcing term" η^k to avoid oversolving without sacrificing quadratic convergence in the terminal phase, see [9] for further discussion and a particularly useful heuristic which we use in the numerical examples.

3 High Order Finite Element Methods

3.1 Discretization

Let $\{\hat{x}_i\}_{i=0}^p$ denote the Legendre-Gauss-Lobatto (LGL) nodes of degree p in ascending order on the interval $[-1, 1]$, with the corresponding Lagrange interpolants $\{\hat{\phi}_i^p\}_{i=0}^p$. Choose a quadrature rule with nodes $\{r_i^q\}_{i=0}^q$ and weights $\{w_i^q\}$. The basis evaluation, derivative, and integration matrices are $\hat{B}_{ij}^{qp} = \hat{\phi}_j^p(r_i^q)$, $\hat{D}_{ij}^{qp} = \partial_x \hat{\phi}_j^p(r_i^q)$, and $\hat{W}_{ij}^q = w_i^q \delta_{ij}$. In the following, we use index-free notation and suppress the superscripts for clarity.

We extend these definitions to 3D via tensor product

$$\begin{aligned} \hat{\boldsymbol{B}} &= \hat{B} \otimes \hat{B} \otimes \hat{B} \\ \hat{\boldsymbol{D}}_0 &= \hat{D} \otimes \hat{B} \otimes \hat{B} \\ \hat{\boldsymbol{D}}_1 &= \hat{B} \otimes \hat{D} \otimes \hat{B} \\ \hat{\boldsymbol{D}}_2 &= \hat{B} \otimes \hat{B} \otimes \hat{D} \end{aligned} \tag{2}$$

with the diagonal weighting matrix $\hat{\boldsymbol{W}} = \hat{W} \otimes \hat{W} \otimes \hat{W}$. With isotropic basis order p and quadrature order q, these tensor product operations cost $2(p^3q + p^2q^2 + pq^3)$ flops and touch only $\mathcal{O}(p^3 + q^3)$ memory. It should be noted that in the special case of the spectral element method where the same LGL points are reused for quadrature, B is the identity and differentiation reduces to $2p^4$ operations. In the present work, we prefer to use more accurate Gauss quadrature and do not find the extra floating point operations in basis evaluation to be prohibitively expensive. Every other required operation will be $\mathcal{O}(p^3)$ or $\mathcal{O}(q^3)$ so the performance of the operation (2) is crucial to the success of the method, see Sect. 3.5 for further discussion.

Let $\hat{K} = [-1, 1]^3$ be the reference element and partition the domain Ω into hexahedral elements $\{K^e\}_{e=1}^E$ with coordinate map $x^e : \hat{K} \to K^e$ and Jacobian $J_{ij}^e = \partial x_i^e / \partial \hat{x}_j$. The Jacobian must be invertible at every quadrature point so we have $(J^e)^{-1} = \partial \hat{x} / \partial x^e$ and can express the element derivative in direction i as

$$\boldsymbol{D}_i^e = \Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right)\hat{\boldsymbol{D}}_0 + \Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right)\hat{\boldsymbol{D}}_1 + \Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right)\hat{\boldsymbol{D}}_2$$

where we have used the notation $\Lambda(x)_{ij} = x_i \delta_{ij}$ for expressing pointwise multiplication as a diagonal matrix. Note that forming $\boldsymbol{D}^e$ would ruin the tensor product structure so multiplication by $\boldsymbol{D}_i^e$ is done using the definition (2) followed by pointwise multiplication and sum. The transpose is defined similarly

$$(\boldsymbol{D}_i^e)^T = \hat{\boldsymbol{D}}_0^T \Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right) + \hat{\boldsymbol{D}}_1^T \Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right) + \hat{\boldsymbol{D}}_2^T \Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right).$$

With the element integration matrix $\boldsymbol{W}^e = \hat{\boldsymbol{W}}\Lambda(|J^e(\boldsymbol{r})|)$, we are prepared to evaluate weak forms over arbitrary elements. The global problem is defined using the element assembly matrix $\mathcal{E} = [\mathcal{E}^e]$ where each $\mathcal{E}^e$ extracts the degrees of freedom associated with element e from the global vector. In the special case of a conforming mesh and equal approximation order on every element, $\mathcal{E}$ will have a single unit entry per row. When the mesh is h- or p-nonconforming, a global basis can be chosen by using minimum order on the largest faces and edges, then the element basis can be constrained by writing it as a linear combination of global basis functions [5].

3.2 Residual Evaluation

We now turn to evaluation of the discrete residual statement $F(u) = 0$ in weak form. Since the weak form is always linear in the test functions, it can be expressed as a pointwise algebraic operation taking values and derivatives of the current iterate $(u, \nabla u)$ to the coefficients of the test functions and derivatives $(v, \nabla v)$, i.e. the Dirichlet problem is to find u in a suitable space V_D such that

$$\langle v, f(u)\rangle = \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u) = 0 \tag{3}$$

for all v in the corresponding homogeneous space V_0 where f_0 and f_1 contain any possible sources. For an n-component problem in d dimensions, $f_0 \in \mathbb{R}^n$ and $f_1 \in \mathbb{R}^{nd}$. Inhomogeneous Neumann, Robin, and nonlinear boundary conditions will add similar terms integrated over boundary faces. The fully discrete form is

$$\sum_e \mathcal{E}_e^T \left[(\boldsymbol{B}^e)^T \boldsymbol{W}^e \Lambda(f_0(u^e, \nabla u^e)) + \sum_{i=0}^{d} (\boldsymbol{D}_i^e)^T \boldsymbol{W}^e \Lambda(f_1(u^e, \nabla u^e)) \right] = \boldsymbol{0} \tag{4}$$

where $u^e = \boldsymbol{B}^e \mathcal{E}^e u$ and $\nabla u^e = \{\boldsymbol{D}_i^e \mathcal{E}^e u\}_{i=0}^2$. Note that all physics is contained in the pointwise operation $(u, \nabla u) \mapsto (f_0, f_1)$.

3.3 Jacobian Representation

Jacobian application $w \mapsto J(u)w$ can be performed in the same way as residual evaluation. The associated bilinear form is expressible as

$$\langle v, J(u)w\rangle = \int_\Omega \begin{bmatrix} v^T & \nabla v^T \end{bmatrix} \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} w \\ \nabla w \end{bmatrix} \tag{5}$$

with the notation $f_{i,0} = \frac{\partial f_i}{\partial u}(u, \nabla u)$ and $f_{i,1} = \frac{\partial f_i}{\partial \nabla u}(u, \nabla u)$. This construction is completely general and applies to any H^1 Galerkin or Petrov-Galerkin method. The application of $J(u)$ and $J^T(u)$ can clearly be performed if $f_{i,j}(u, \nabla u)$ is stored at quadrature points (including similar terms at boundary quadrature points where boundary integrals are required).

There are two extreme cases for the storage of $f_{i,j}(u, \nabla u)$. The first is to simply store $(u, \nabla u)$ which requires $n(d+1)$ storage per quadrature point for an n-component system in d dimensions, but all the physics must be reevaluated in each Jacobian application. This option is slightly faster than standard use of automatic differentiation since the finite element mechanics to reevaluate $(u, \nabla u)$ is not needed. The other extreme is to fully evaluate $f_{i,j}(u, \nabla u)$ which requires $[n(d+1)]^2$ storage per quadrature point. Compare to $n^2(p+1)^d$ per node for the Q_p element matrices and (asymptotically) for globally assembled matrices.

For many physical systems, $f_{i,j}(u, \nabla u)$ possesses structure (e.g. sparsity, symmetry, isotropic plus rank-1 update) such that compared to the naïve representation, it is both cheap to store and cheap to multiply by. Such representations may be constructed from any form between $(u, \nabla u)$ and $f_{i,j}$. It is common for this efficient representation to be available at low cost during residual evaluation, and it may be stored cheaply since the memory bus is relatively unstressed at this time. The process from (3) to (5) is mechanical and is easily performed using symbolic algebra. Also note that AD may be used as a complementary technique, well-isolated to pointwise operations so that it doesn't interfere with global software design.

Note that with this representation, the adjoint of (5) is readily available with no additional programming effort, at exactly the same cost as the Jacobian itself. A worked example is given in Sect. 4.1, we find that our matrix-free Jacobian application is significantly faster than function evaluation. Since it is full accuracy and provides adjoints, this representation is clearly superior to finite difference Jacobians. Note that naïve automatic differentiation of F would not be aware of this efficient intermediate form, thus an AD Jacobian $J(u)$ would need to re-evaluate the base vector at quadrature points and compute $f_{i,j}\left[\begin{smallmatrix} w \\ \nabla w \end{smallmatrix}\right]$ using $(u, \nabla u)$. The intermediate form is roughly equivalent to an optimal taping strategy.

3.4 Preconditioning

To precondition J, we assemble a matrix by rediscretizing the governing equations on Q_1 sub-elements defined by the LGL interpolation nodes as shown in Fig. 2. In the notation of Sect. 2, this matrix is J_p and any preconditioner P for assembled matrices can be used in the Krylov iteration. This could be a direct solve, but that is rarely the most economical choice. When multigrid works well, such as for elliptic problems with smooth coefficients, we find that $\mathsf{P} = MG$ requires very few extra iterations compared to a direct solve $\mathsf{P} = LU$. This preconditioner has received substantial attention in the collocation and spectral element contexts (e.g. [7, 8, 15, 18, 27]), but is not widely used for Galerkin discretization with independent quadrature.

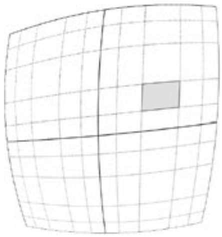

Fig. 2 A patch of four Q_5 elements with one Q_1 subelement shaded

3.5 Performance of Matrix Operations

Our test platform is a 2.5 GHz Intel Core 2 Duo (T9300) which can issue one packed double precision (two 64-bit values packed in each 128-bit register) add and one packed multiply per clock cycle with a latency of 3 and 5 cycles respectively. If there are no data dependencies and both execution units can be kept busy, it is possible to perform 4 flops per clock cycle resulting in a theoretical peak of 10 Gflop/s. The memory controller has a peak read throughput of 5.3 GB/s implying that dual core performance for matrix-vector products cannot exceed 900 Mflop/s, ignoring access costs for the vector (each matrix entry is 8 bytes plus 4 bytes for the column index, and supplies only 2 flops for an arithmetic intensity of 1 flop / 6 bytes). Note that this is less than 5% of the theoretical dual-core peak 20 Gflop/s. Even if the column indices were known, the operation cannot exceed 4 bytes per flop, despite the system being capable of performing nearly 4 flops for each byte loaded from memory.

For multi-component problems, we can exploit structural blocking to reduce the bandwidth required for column indices. For example, a 3-component problem with full coupling only needs to store one index per 3×3 block, improving arithmetic intensity to almost 1 flop / 4 bytes. In addition, it is advantageous to reorder the unknowns owned by each processor to reduce the matrix bandwidth (e.g. with reverse Cuthill-McKee), thus improving cache reuse by the vector during matrix-vector multiplication and preconditioning kernels (relaxation and solves with factors), and also the effectiveness of these preconditioning kernels. On our 3D elasticity test, these two optimizations combined for a 75% speedup relative to scalar formats in the natural ordering, see [13] for further analysis of blocking and reordering.

High order methods allow much more effective utilization of today's multicore hardware. An implementation of the tensor product operation (2) operates entirely within L1 cache for practical basis orders and attains 5.7 Gflops for Q_3 on a single core. (This was written using SSE3 intrinsics since the author was unable to find a compiler that produced competitive code for this operation.) Since this kernel puts no pressure on the shared memory bus, all cores are completely independent and limited only by instruction scheduling and data dependence. Although only the tensor-product operation has been vectorized, multiplication by the matrix-free Q_3 Jacobian for a scalar-valued problem is only 2.5 times slower than the assembled Q_2 Jacobian when using one core of an otherwise idle socket.

4 Numerical Examples

All examples with dual-order preconditioning were implemented as part of a new C library named *Dohp*, available from the author, which is tightly integrated with PETSc [2] and uses the ITAPS [16] interfaces for mesh and geometry services. In particular, we use the MOAB [32] and CGM [31] implementations of the *iMesh* and *iGeom* interfaces. In the examples using algebraic multigrid, we use smoothed aggregation from ML [12] and classical multigrid from BoomerAMG [14], all accessed through the common PETSc interface. The ML interface is designed to only produce aggregates which PETSc uses to construct the multigrid hierarchy and thus exposes all PETSc preconditioners as smoothers, while BoomerAMG is essentially a black-box preconditioner. Smoothed aggregation is known to scale slightly superlinearly, but in our tests ML was always significantly faster and less memory consuming than the typically more robust BoomerAMG. We use PETSc-3.0.0, ML-6.2[2], and BoomerAMG from Hypre-2.4.0b.

The `libMesh` [19] library is used to provide a reference for conventional methods (based on assembling the true Jacobian). Since `libMesh` also uses PETSc for linear algebra, identical solver parameters are used so that the results are representative.

4.1 p-Poisson

The inhomogeneous $\mathfrak{p}$-Laplacian is

$$-\nabla \cdot (|\nabla u|^{\mathfrak{p}-2} \nabla u) - f = 0$$

where $1 \le \mathfrak{p} \le \infty$ (see [11] for discussion of these limiting cases). This equation is singular at $\nabla u = 0$ when $\mathfrak{p} < 2$ and degenerate when $\mathfrak{p} > 2$, so we solve a regularized variant with weak form: find $u \in V_D$ such that

$$\langle v, F(u) \rangle = \int_\Omega \eta \nabla v \cdot \nabla u - f v = 0 \tag{6}$$

for all $v \in V_0$ where $V_D = H^1_D(\Omega)$ includes inhomogeneous Dirichlet conditions, $V_0 = H^1_0(\Omega)$ is the corresponding homogeneous space, and $\eta(\gamma) = (\epsilon + \gamma)^{\frac{\mathfrak{p}-2}{2}}$ is effective viscosity with regularization $\epsilon > 0$ and $\gamma = \frac{1}{2}|\nabla u|^2$. We manufacture the forcing term f so that

$$u(x, y, z) = \cos(ax) \sin(by) \exp(cz)$$

solves (6) in $\Omega = [-1, 1]^3$. Figure 1 shows convergence rates for the linear ($\mathfrak{p} = 2$) case $a, b, c = (16, 15, 14)$ with $1 \le p \le 7$. The expected $\mathcal{O}(h^{p+1})$ is indeed observed in the asymptotic range, but in this section we are concerned with the time and memory needed for solution.

The Newton step for (6) corresponds to the weak form: find $w \in V_0$ such that

$$\langle v, J(u) w \rangle = \int_\Omega \eta \nabla v \cdot \nabla w + \eta' (\nabla v \cdot \nabla u)(\nabla u \cdot \nabla w) = -(v, F(u))$$

for all $v \in V_0$, where $J(u)$ is the Jacobian of F at u. It is informative to rewrite the integrand as

$$\nabla v {:} \big[\eta \mathbf{1} + \eta' \nabla u \otimes \nabla u\big] {:} \nabla w$$

to clarify the effect of the Newton linearization. Since $\eta' < 0$ for $\mathfrak{p} < 2$, the (heterogeneous) isotropic conductivity tensor $\eta \mathbf{1}$ is being squished in the direction ∇u. In the singular limit

[2] We have observed that ML-6.2 produces much higher quality aggregates than ML-5.0 for the assembled Q_2 case, leading to a 50% speedup and significantly lower memory usage.

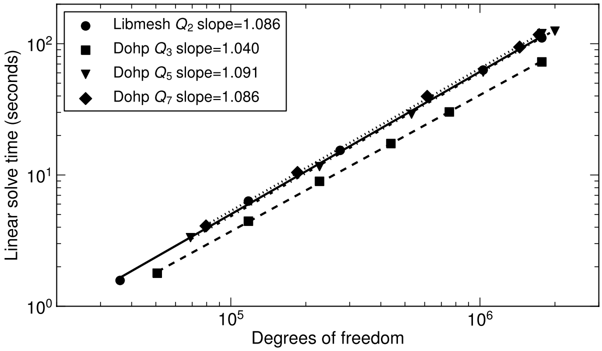

Fig. 3 Linear solve time for 3D Poisson with relative tolerance of 10^{-8} using assembled Q_2 elements and unassembled Q_3, Q_5, and Q_7 elements preconditioned by an assembled Q_1 operator

Table 1 Assembly and solve time (seconds) for a 3D Poisson problem with 121^3 degrees of freedom (120^3 for Q_7), relative tolerance of 10^{-8}

Event	`libMesh` Q_2	Dohp Q_3	Dohp Q_5	Dohp Q_7
Assembly	41	25	24	23
Krylov	111	73	119	117
MF MatMult	–	36	55	55
PCSetUp	16	10	12	9
PCApply	82	27	51	52
CG its	34	23	41	49
Mat nonzeros	111 M	44.7 M	44.7 M	44.3 M

$\mathfrak{p} \to 1$, it flattens completely which has the effect of allowing diffusion only in level sets of u.

To illustrate the taping strategy of Sect. 3.3 for the $\mathfrak{p}$-Laplacian, first note that it is desirable to store ∇u at each quadrature point in order to avoid it's expensive recomputation. Fractional powers are one of the more expensive mathematical operations so we would also like to avoid recomputing them. This suggests taping $(\eta, \nabla u)$ which are explicitly available during residual evaluation (6). If a few Krylov iterations will be required, it becomes beneficial to amortize computation of

$$\eta' = \frac{\partial \eta}{\partial \gamma} = \frac{\mathfrak{p}-2}{2}\frac{\eta}{\epsilon+\gamma}$$

which involves one multiplication and one division when taped during function evaluation, very cheap compared to the fractional power, and less expensive than recomputing γ on the first Krylov iteration.

A further enhancement[3] to taping $(\eta, \eta', \nabla u)$ is $(\eta, \sqrt{-\eta'}\nabla u)$ which is both compact and minimizes the operations necessary to apply the Jacobian.

Figure 3 shows linear solve performance compared to conventional methods for quadratic elements. The matrices for conventional Q_2 elements were assembled using the `libMesh` library and the solver in all cases was conjugate gradients preconditioned by ML. The time spent in each phase for the largest problem size is shown in Table 1. Note the greatly reduced assembly time compared to conventional Q_2 elements.

[3] `sqrt` requires the same number of cycles as division.

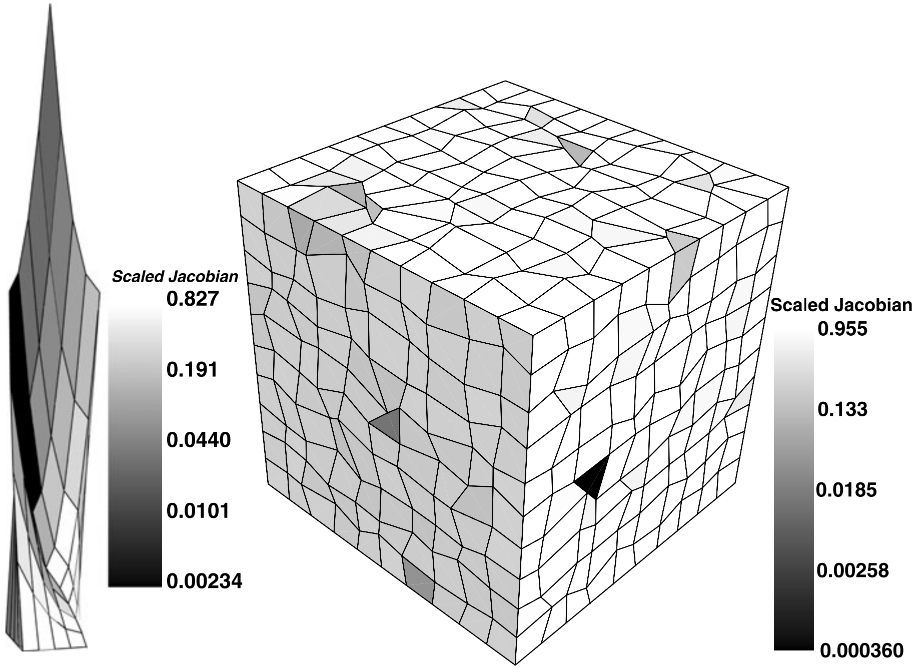

Fig. 4 Two all-hexahedral meshes, `twist` and `random`, containing nearly degenerate elements

Table 2 Iteration counts for four different meshes using ML, BoomerAMG (BMG), and Cholesky (Chol) preconditioning. For Q_1 `random`, BoomerAMG failed, producing NaN. ML on the `flat` mesh was run with stronger smoothers as described in the text. Note that the problem size increases with element order

Mesh	`brick`		`twist`		`random`		`flat`	
element\PC	ML	BMG	ML	BMG	ML	BMG	ML	Chol
Q_1	4	4	4	4	8	–	4	1
Q_2	24	24	25	23	27	27	29	26
Q_3	24	24	29	27	28	27	38	34
Q_4	29	28	39	30	34	33	47	37
Q_5	35	27	47	34	42	35	58	40
Q_6	35	29	60	40	43	41	80	44

4.1.1 Poor-Quality Meshes

Figure 4 shows two low-quality meshes, `twist` and `random`. The first was generated by stretching and twisting a mesh of the reference cube and the second generated using the "random" feature of Cubit [4]. Additionally, we consider `flat`, a uniform Cartesian mesh with dimensions $1 \times 1 \times 10^{-5}$. These should be compared to `brick`, an 8^3 mesh of the reference cube.

While the approximation properties of these meshes are poor, the preconditioner is still effective provided a good preconditioner for the low-order system is available. The iteration count to oversolve the second Newton iteration to a relative tolerance of 10^{-8} for a $\mathfrak{p} = 1.5$ case is shown in Table 2. With the exception of ML on `twist` and `flat`, the mesh has a limited impact on performance.

Table 3 Time (seconds) spent in various stages of a nonlinear solve for Q_3 elements preconditioned by ILU(0) and ILU(1) applied to the corresponding Q_1 matrix. The *first columns* solve the linear system to a relative tolerance of 10^{-4} with a maximum of 60 Krylov iterations per Newton, the latter use the Eisenstat-Walker method for adjusting solver tolerances [9] with a maximum of 30 Krylov iterations. The events *MF MatMult* (matrix-free Jacobian application), *PCSetup*, and *PCApply* are part of the *Krylov* iteration. Additional *Residual* evaluations are needed when the line search is activated

Tolerance	10^{-4} relative		E-W	
event\PC	ILU(0)	ILU(1)	ILU(0)	ILU(1)
Residual	35	29	57	51
Assembly	78	68	127	110
Krylov	295	274	187	166
MF MatMult	259	190	155	110
PCSetup	2	11	5	17
PCApply	26	67	27	39
Total time	413	374	377	333
Newton #	15	13	24	21
Residual #	25	20	40	36
Krylov #	911	667	545	386

Anisotropic meshes such as `flat` require semi-coarsening and/or line smoothers for multigrid performance. Convergence on this mesh is poor with both algebraic multigrid packages. BoomerAMG allows little flexibility in smoothers, and although some semi-coarsening was evident in the hierarchy, coarse spaces suitable for the builtin smoothers were not produced. ML's coarse spaces were less precisely semi-coarsened, and iteration counts could only be marginally controlled through the use of very strong smoothers (8-block overlap-1 additive Schwarz with direct subdomain solves).

4.1.2 Nonlinearities

For strongly nonlinear problems, the majority of the solve time is spent in the pre-asymptotic range. Since a high-accuracy solve is not needed, it is very important to consider matrix assembly and preconditioner setup time when developing an efficient solver. We consider the $\mathfrak{p} = 1.2, \epsilon = 10^{-8}$ case with an initial guess of zero (the state at which the system is most nearly singular) which requires a fair number of Newton steps. Table 3 shows the time spent in various stages of the solve for 20^3 Q_3 elements (226981 degrees of freedom). In the most efficient configuration, roughly one third of the time is spent in matrix-free Jacobian application with a similar amount spent in assembly. This problem demonstrates a situation where low-accuracy linear solves slow the nonlinear convergence, but it is still not cost effective to perform high accuracy linear solves. Lagging the preconditioner was not found to be effective for this problem, therefore only limited returns would be provided by a method with higher assembly costs, even if that setup enabled the use of a stronger preconditioner. We have computed a fourth order accurate solution with essentially the same memory requirements and modest cost increase over a second order scheme (Q_1). This solution is less expensive in both space and time than a conventional third order (Q_2) scheme, the reduced assembly costs offer more flexibility in nonlinear solver, and the sparser matrix offers more preconditioning choices.

4.2 Stokes

The weak form of the Dirichlet Stokes problem is: find $(u, p) \in \boldsymbol{V}_D \times P$ such that

$$\int_\Omega \eta Dv{:}Du - p\nabla \cdot v - q\nabla \cdot u - f \cdot v = 0$$

for all $(v, q) \in \boldsymbol{V}_0 \times P$ where $Du = \frac{1}{2}(\nabla u + (\nabla u)^T)$ is the symmetric gradient, $\boldsymbol{V}_D = \boldsymbol{H}^1_D(\Omega)$ is the inhomogeneous velocity space with $\boldsymbol{V}_0$ the corresponding homogeneous space, and $P = \{p \in L^2_0(\Omega) : \int_\Omega p = 0\}$ is the pressure space. Stability requires satisfaction of the discrete inf-sup condition

$$\inf_p \sup_u \frac{\int_\Omega p\nabla \cdot u}{\|p\|_0 \|u\|_1} \geq \beta > 0.$$

The finite element space $Q_k - Q_{k-2}$ is stable with $\beta \in \mathcal{O}(k^{-(d-1)/2})$ in d dimensions (see [29]) and is thus quite usable for the modest orders we propose.

4.2.1 *Indefinite Preconditioning*

Standard preconditioners perform poorly or fail completely when applied to indefinite problems. Block factorization provides a general framework for constructing effective preconditioners for the Stokes problem. They are based on factoring the Jacobian as

$$J(u) = \begin{bmatrix} A(u) & B^T \\ B & \end{bmatrix} = \begin{bmatrix} 1 & \\ BA^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & \\ & S \end{bmatrix} \begin{bmatrix} 1 & A^{-1}B^T \\ & 1 \end{bmatrix} \tag{7}$$

where $S = -BA^{-1}B^T$ is the Schur complement which is dense and must be preconditioned by other means. When GMRES is used with left (right) preconditioning, the upper (lower) block is typically dropped (since the resulting exactly preconditioned operator has minimal degree 2, see [24]), and all occurrences of A^{-1} are replaced by a suitable preconditioner. There are numerous ways to precondition S (e.g. [3, 10]), here we use only the classic pressure mass matrix M_p, but more sophisticated methods are needed for strongly heterogeneous or anisotropic problems (see [23]), for Navier-Stokes with non-vanishing Reynolds number, and for short time steps in a time-dependent simulation. With the dual-order method, it is only necessary to assemble matrices to precondition A and S (the latter via a diagonal approximation to the mass matrix M_p), all "matrix multiplies" are performed matrix-free.

For the Dirichlet Stokes problem, constant pressure is in the null space of J and S. This does not impact solver performance as long as the right hand side is consistent and the solver removes the null space from the Krylov basis. Table 4 shows iteration counts for a variety of elements and meshes using right-preconditioned GMRES and the right-triangular preconditioner resulting from the factorization (7) with S^{-1} approximated by the diagonal of M_p and A^{-1} approximated by one V-cycle of ML with inter-component coupling dropped. The iteration count is scalable with resolution, but deteriorates much more rapidly with element order than for the definite problem. The corresponding solve times, shown in Fig. 5, confirm the asymptotics under h-refinement with each element type. More sophisticated preconditioners for S were able to reduce the iteration count, but did not reliably reduce solve time across the range of element orders. Further investigation of indefinite preconditioner performance under p-refinement is needed.

Table 4 Problem size and Krylov iterations to solve the Stokes problem to a relative tolerance of 10^{-6} with elements of different orders using right-preconditioned GMRES(30) and field-split ML on the A block. Each restart caused an approximately 2-iteration stall

Element	$Q_3 - Q_1$		$Q_5 - Q_3$		$Q_7 - Q_5$	
mesh	dofs	its	dofs	its	dofs	its
4^3	4118	38	22774	79	68310	92
8^3	37230	38	193582	75	568046	92
12^3	130822	38	666790	76	1942342	95
16^3	316382	40	1596766	77		
20^3	625398	40				
24^3	1089358	40				
30^3	2144698	41				

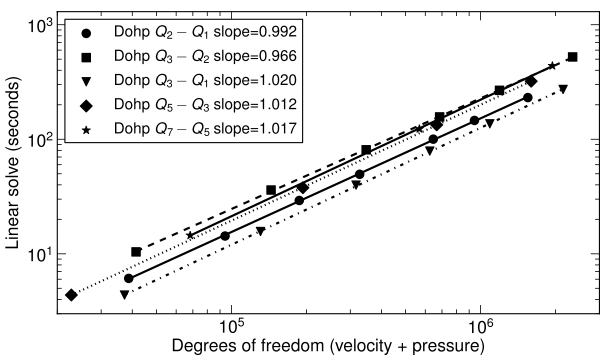

Fig. 5 Linear solve time for 3D Stokes with relative tolerance of 10^{-6}. For $Q_2 - Q_1$ and $Q_3 - Q_2$ elements, convergence is significantly slower than with $Q_k - Q_{k-2}$, but apparently scalable despite being somewhat erratic

5 Discussion

We have presented a practical method of obtaining high-order accuracy with cost similar to conventional methods for lower order accuracy. It is robust on highly deformed meshes and nonlinear problems provided an effective preconditioner is available for the associated Q_1 matrix. The computational kernels remove significant pressure on the memory bus enabling more effective use of floating point units, especially in multicore environments.

The extra structure imposed by high-order methods provides natural coarsening which suggests the use of one or more levels of geometric multigrid. This avoids the cost of computing interpolation operators in algebraic multigrid and enables the use of rediscretized coarse operators which preserve sparsity in comparison to Galerkin operators. Integration with PETSc's geometric multigrid framework is underway.

Since solve time is only weakly dependent on element order, the dual-order scheme is well-suited for use in an *hp*-adaptive simulation where existing refinement strategies for minimizing degrees of freedom will more accurately represent computational cost. Relative to conventional Q_2 and higher elements, the dual-order scheme significantly reduces the cost of matrix assembly and preconditioner setup, while maintaining competitive iteration counts when used with preconditioners such as algebraic multigrid.

Acknowledgements We thank the PETSc team for support and a framework in which this work could be naturally expressed. Thanks also to Tim Tautges for explaining numerous mesh and geometry issues and providing the MOAB and CGM libraries. This work was supported by Swiss National Science Foundation Grant 200021-113503/1.

References

1. Ainsworth, M., Senior, B.: An adaptive refinement strategy for *hp*-finite element computations. Appl. Numer. Math. **26**(1), 165–178 (1998)
2. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., Curfman McInnes, L., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11—Revision 3.0.0, Argonne National Laboratory (2008)
3. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**, 1–137 (2005)
4. Blacker, T., Bohnhoff, W., Edwards, T., Hipp, J., Lober, R., Mitchell, S., Sjaardema, G., Tautges, T., Wilson, T., Oakes, W., et al.: CUBIT mesh generation environment. Technical report, Sandia National Labs., Albuquerque, NM. Cubit Development Team (1994)
5. Demkowicz, L., Oden, J.T., Rachowicz, W., Hardy, O.: Toward a universal hp adaptive finite element strategy. I: Constrained approximation and data structure. Comput. Methods Appl. Mech. Eng. **77**, 79–112 (1989)
6. Demkowicz, L., Rachowicz, W., Devloo, P.: A fully automatic *hp*-adaptivity. J. Sci. Comput. **17**(1), 117–142 (2002)
7. Deville, M., Mund, E.: Chebyshev pseudospectral solution of second-order elliptic equations with finite element preconditioning. J. Comput. Phys. **60**, 517 (1985)
8. Deville, M.O., Mund, E.H.: Finite-element preconditioning for pseudospectral solutions of elliptic problems. SIAM J. Sci. Stat. Comput. **11**, 311 (1990)
9. Eisenstat, S.C., Walker, H.F.: Choosing the forcing terms in an inexact newton method. SIAM J. Sci. Comput. **17**(1), 16–32 (1996)
10. Elman, H.C., Howle, V.E., Shadid, J., Shuttleworth, R., Tuminaro, R.: A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. J. Comput. Phys. **227**(1), 1790–1808 (2008)
11. Evans, L.C.: The 1-Laplacian, the ∞-Laplacian and differential games. Perspect. Nonlinear Partial Differ. Equ.: In Honor of Haim Brezis **446**, 245 (2007)
12. Gee, M.W., Siefert, C.M., Hu, J.J., Tuminaro, R.S., Sala, M.G.: ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories (2006)
13. Gropp, W.D., Kaushik, D.K., Keyes, D.E., Smith, B.: Performance modeling and tuning of an unstructured mesh cfd application. In: Supercomputing '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM), Washington, DC, USA, 2000, p. 34. IEEE Computer Society, New York (2000)
14. Henson, V.E., Yang, U.M.: BoomerAMG: a parallel algebraic multigrid solver and preconditioner. Appl. Numer. Math. **41**(1), 155–177 (2002)
15. Heys, J.J., Manteuffel, T.A., McCormick, S.F., Olson, L.N.: Algebraic multigrid for higher-order finite elements. J. Comput. Phys. **204**(2), 520–532 (2005)
16. Interoperable technologies for advanced petascale simulations (ITAPS). http://www.itaps.org/
17. Karniadakis, G.E., Sherwin, S.J.: Spectral/hp Element Methods for Computational Fluid Dynamics. Oxford University Press, Oxford (2005)
18. Kim, S.D.: Piecewise bilinear preconditioning of high-order finite element methods. Electron. Trans. Numer. Anal. **26**, 228–242 (2007)
19. Kirk, B., Peterson, J.W., Stogner, R.H., Carey, G.F.: `libMesh`: A C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng. Comput. **22**(3–4), 237–254 (2006). http://dx.doi.org/10.1007/s00366-006-0049-3
20. Knoll, D.A., Keyes, D.E.: Jacobian-free Newton–Krylov methods: a survey of approaches and applications. J. Comput. Phys. **193**(2), 357–397 (2004)
21. Knoll, D.A., McHugh, P.R.: Enhanced nonlinear iterative techniques applied to a nonequilibrium plasma flow. SIAM J. Sci. Comput. **19**(1), 291–301 (1998)
22. Lottes, J.W., Fischer, P.F.: Hybrid multigrid/Schwarz algorithms for the spectral element method. J. Sci. Comput. **24**(1), 45–78 (2005)
23. May, D.A., Moresi, L.: Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics. Phys. Earth Planet. Inter. **171**(1–4), 33–47 (2008). Recent Advances in Computational Geodynamics: Theory, Numerics and Applications
24. Murphy, M.F., Golub, G.H., Wathen, A.J.: A note on preconditioning for indefinite linear systems. SIAM J. Sci. Comput. **21**(6), 1969–1972 (2000)
25. Nachtigal, N.M., Reddy, S.C., Trefethen, L.N.: How fast are nonsymmetric matrix iterations? SIAM J. Matrix Anal. Appl. **13**, 778 (1992)
26. Oden, J.T., Demkowicz, L., Rachowicz, W., Westermann, T.A.: Toward a universal *hp* adaptive finite element strategy. II: A posteriori error estimation. Comput. Methods Appl. Mech. Eng. **77**, 113–180 (1989)

27. Orszag, S.A.: Spectral methods for problems in complex geometries. J. Comput. Phys. **37**, 70–92 (1980)
28. Rachowicz, W., Oden, J.T., Demkowicz, L.: Toward a universal *hp* adaptive finite element strategy. III: Design of *hp* meshes. Comput. Methods Appl. Mech. Eng. **77**, 181–212 (1989)
29. Schötzau, D., Schwab, C., Stenberg, R.: Mixed *hp*-FEM on anisotropic meshes. Math. Models Methods Appl. Sci. **8**, 787–820 (1998)
30. Schwab, C.: P- and Hp-Finite Element Methods: Theory and Applications in Solid and Fluid Mechanics. Oxford University Press, Oxford (1998)
31. Tautges, T.J.: CGM: a geometry interface for mesh generation, analysis and other applications. Eng. Comput. **17**(3), 299–314 (2001)
32. Tautges, T.J., Meyers, R., Merkley, K., Stimpson, C., Ernst, C.: MOAB: a mesh-oriented database. Technical report, Sandia National Laboratories, April 2004