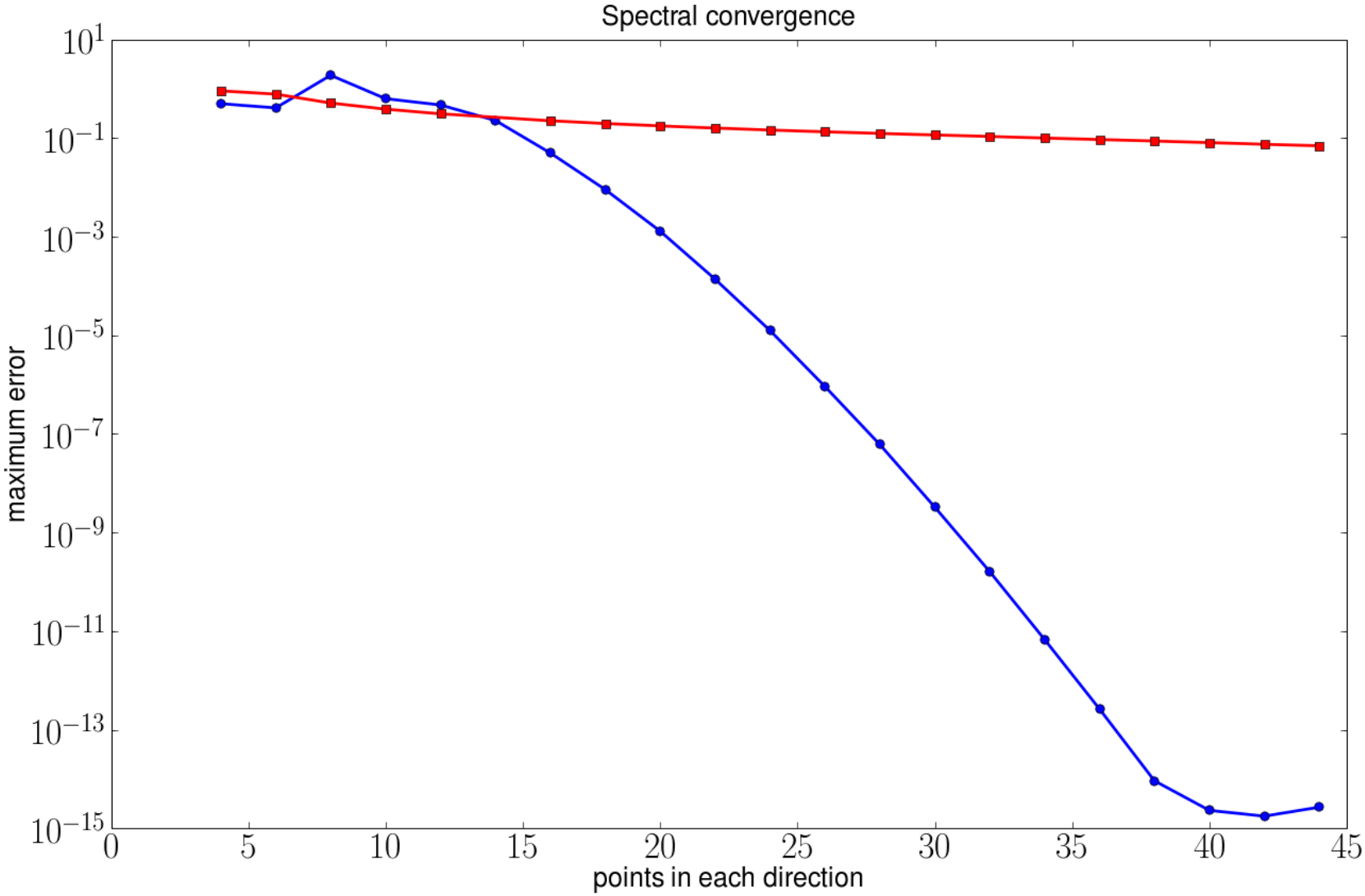
Dual Order hp version of the finite element method

Jed Brown

VAW, ETH Zürich

2009-02-05

Why high order?



For geophysical flows

- Very large domain
  - Most of the domain is rather boring
  - Accuracy still very important due to constitutive relation (for ice)
  - Optimal: very large high-order elements where the flow is boring
- High aspect ratio
  - Discrete inf-sup stability (Ladyzhenskaja-Babuška-Brezzi)
    - The divergence of the velocity space should span the pressure space in a nice way.

$$\inf_{p \in P} \sup_{\boldsymbol{u} \in \boldsymbol{V}} \frac{\int_\Omega p \nabla \cdot \boldsymbol{u}}{\|p\|_0 \|\boldsymbol{u}\|_1} \geq \beta > 0$$

  - Stability with corners for aspect ratio ρ^{-1} and velocity-pressure pair order $(k+1, k-1)$ [Ainsworth and Coggins, 2000]

$$\beta \geq C k^{-1/2} \min(1, k\sqrt{\rho})$$

Taking $k \approx \rho^{-1/2}$, $\mu = 0$ leads to $\beta \geq C\rho^{1/4}$, better than the usual $\beta \geq C\rho^{1/2}$

For geophysical flows

- Very large domain
  - Most of the domain is rather boring
  - Accuracy still very important due to constitutive relation (for ice)
  - Optimal: very large high-order elements where the flow is boring
- High aspect ratio
  - Discrete inf-sup stability (Ladyzhenskaja-Babuška-Brezzi)
    - The divergence of the velocity space should span the pressure space in a nice way.

$$\inf_{p \in P} \sup_{\boldsymbol{u} \in \boldsymbol{V}} \frac{\int_\Omega p \nabla \cdot \boldsymbol{u}}{\|p\|_0 \|\boldsymbol{u}\|_1} \geq \beta > 0$$

  - Stability with corners for aspect ratio ρ^{-1} and velocity-pressure pair order $(k+1, k-1)$ [Ainsworth and Coggins, 2000]

$$\beta \geq C k^{-1/2} \min(1, k\sqrt{\rho})$$

    Taking $k \approx \rho^{-1/2}$, $\mu = 0$ leads to $\beta \geq C\rho^{1/4}$, better than the usual $\beta \geq C\rho^{1/2}$
- Also: It's faster!

Weak forms

- Strong form (requires two derivatives):
  Find u such that

$$\begin{aligned} -\nabla \cdot (\eta \nabla u) - e^u - f &= 0 && \text{in } \Omega \\ u &= g_D && \text{in } \Gamma_D \\ \nabla u \cdot n &= g_N && \text{in } \Gamma_N \end{aligned}$$

  - Define solution space by values at nodes
  - Choose rule for interpolation and differentiation
  - Require strong form to be true at nodes
- Weak form (minimum regularity requirements):
  Find $u \in V_D = \{H^1(\Omega) : \tau(u)|_{\Gamma_D} = g_D\}$ such that

$$\int_\Omega \eta \nabla v \cdot \nabla u - v e^u - f v - \int_{\Gamma_N} g_N v = 0$$

  for all $v \in V_0$.
  - Define solution space as a discrete subspace of V_D
  - Choose test space as a discrete subspace of V_0
    - *Galerkin* if same as solution space, *Petrov-Galerkin* if different
  - Choose integration rule

Choosing discrete approximation spaces

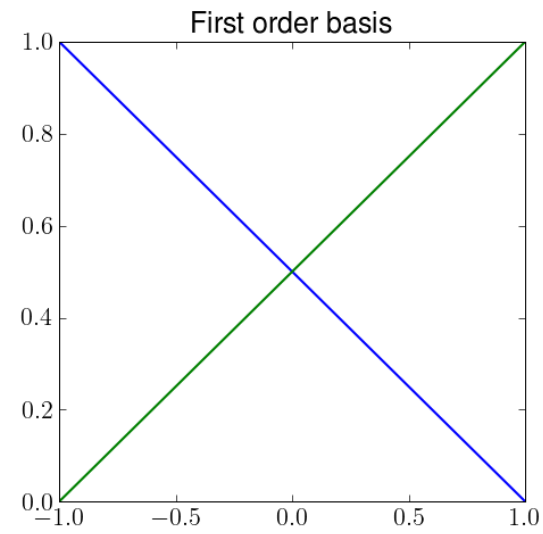
- Partition the domain into elements $\{K_e\}$
- Choose a basis $\hat{X}$ for the reference element $\hat{K} = [-1, 1]^d$
- Basis functions on K_e are $X_e = \hat{X} \circ F_e^{-1}$ where $F_e : \hat{K} \to K_e$
  - Derivatives obtained as

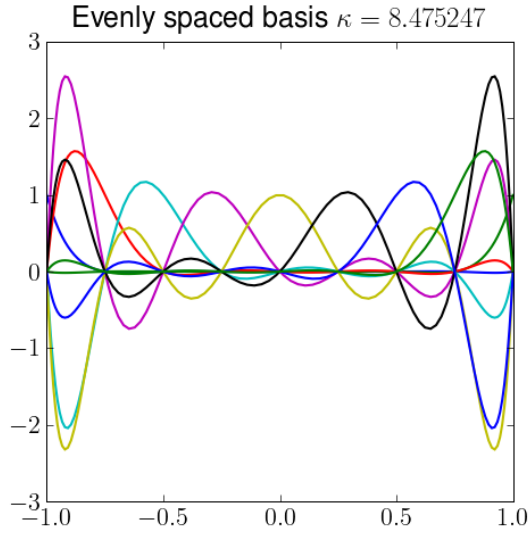
$$\nabla X_e = \Lambda(J_e^{-1}) \circ (\nabla \hat{X}) \circ F_e^{-1}$$

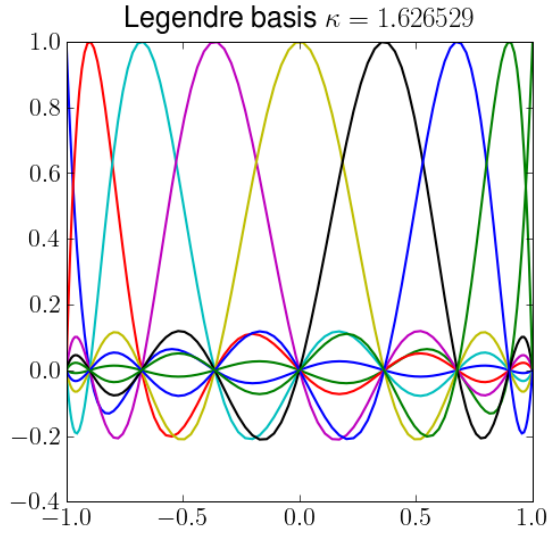
    where
    - $J_e = \partial F_e(\hat{x})/\partial \hat{x}$ is the element Jacobian
    - $\Lambda(\,\cdot\,)$ is pointwise multiplication
- Choose global degrees of freedom to obtain continuity
  - Might require constraints

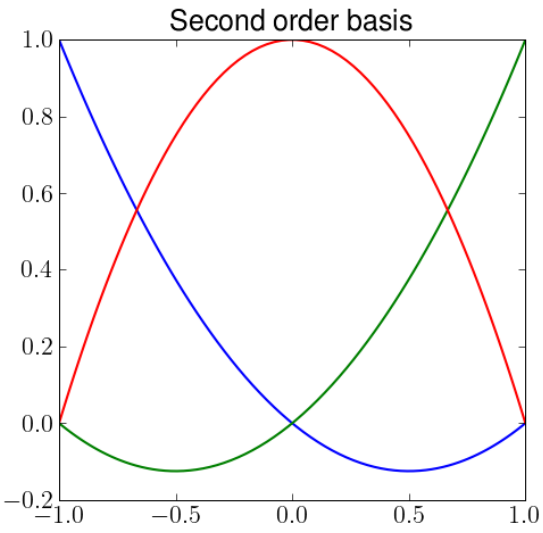
Jed: draw pictures!

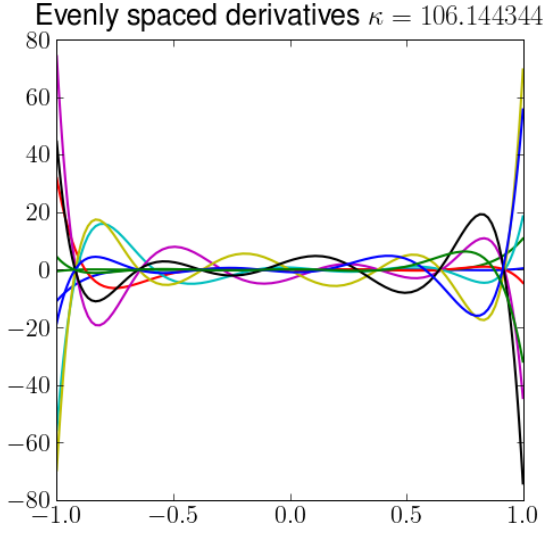
Nodal p-version finite elements, 1-dimensional bases

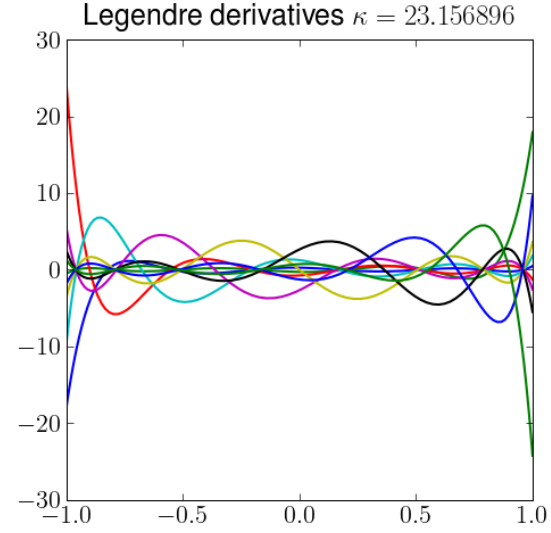












Multiple dimensions: Tensor product bases

- One dimension

$$u(x) = \sum_{i=0}^{p} u_i h_i^p(x) = \sum_{i=0}^{p} u_i \hat{h}_i^p\left(F_e^{-1}(x)\right)$$

- Tensor product

$$u(x,y) = \sum_{i,j=0}^{p,q} u_{ij} h_i^p(x) h_j^q(y)$$

$$= \sum_{i,j=0}^{p,q} u_{ij} \hat{h}_i^p\left(F_{e,x}^{-1}(x,y)\right) \hat{h}_j^q\left(F_{e,y}^{-1}(x,y)\right)$$

Quadrature

- Write basis functions evaluated at quadrature points as matrix

$$B = h_j(q_i)$$

  - Each column is a basis function evaluated at quadrature points
- Likewise for derivatives on reference element: $D_x = \partial_x h_j(q_i)$
  - Derivatives on physical element become: $D_x^e = \Lambda(J^{-1})D_x$
  - In multiple dimensions

$$D_x^e = \Lambda(J_{xx}^{-1})D_x + \Lambda(J_{xy}^{-1})D_y)$$

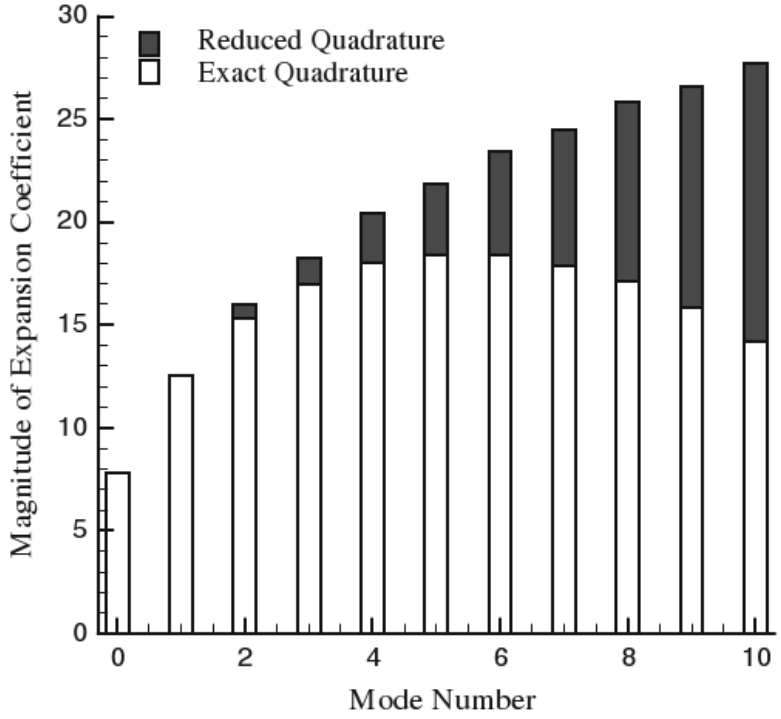
$$D_y^e = \Lambda(J_{yx}^{-1})D_x + \Lambda(J_{yy}^{-1})D_y)$$

- Integrate on physical element by multiplying weights by $|J|$

$$\int_{K_e} \phi(F^{-1}(x)) = \sum_{i=0}^{Q} w_i \left|J(q_i)\right| \phi(q_i)$$

- Linear and simple nonlinear forms can be integrated exactly using sufficiently high quadrature
- Usually use ordinary Gauss quadrature and don't integrate nonlinear terms exactly (causes aliasing)

Aliasing



- Projection onto 10th degree polynomial space using inexact quadrature.

Evaluating weak forms (nonlinear function evaluation)

$$\int_\Omega \eta \nabla v \cdot \nabla u - v e^u - f v = 0$$

Sum contributions from each element:

1. Obtain values of $u, \nabla u$ at quadrature points using B, D_x^e, D_y^e
2. Compute coefficients of $v, \nabla v$
3. Weight with $W_e = W \Lambda(|J|)$
4. Transform back to test space using $B^T, (D_x^e)^T, (D_y^e)^T$

$$(D_x^e)^T W_e \eta(\frac{1}{2} |\nabla u|^2) \partial_x u + (D_x^e)^T W_e \eta(\frac{1}{2} |\nabla u|^2) \partial_y u - B^T W_e e^u$$

Assembling matrices

- Must have a bilinear form

$$b(v,u) = \int_\Omega \eta \nabla v \cdot \nabla u + \eta'(\nabla v \cdot \nabla \tilde{u})(\nabla \tilde{u} \cdot \nabla u) - v e^{\tilde{u}} u$$

- Compute element stiffness matrix as

$$A_e = (D_x^e)^T W_e \left(\Lambda(\eta) D_x^e + \Lambda\left(\eta' \nabla \tilde{u} (\nabla \tilde{u})^T\right) D_x^e \right) - B^T W_e \Lambda(e^{\tilde{u}}) B$$

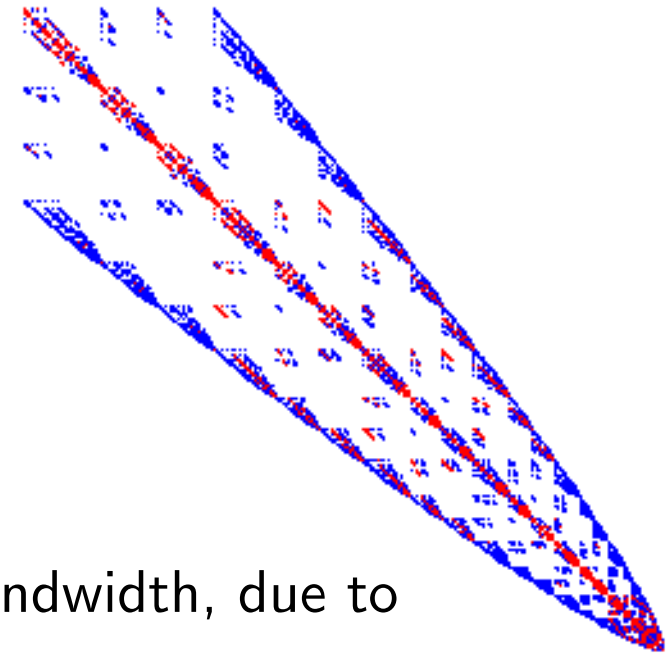
- Standard method: just multiply these matrices together
  - $\mathcal{O}(p^{3d})$ operations in d dimensions
  - $\mathcal{O}(p^{2d})$ memory

Jed: Show code

Sparse mat-vec and current hardware

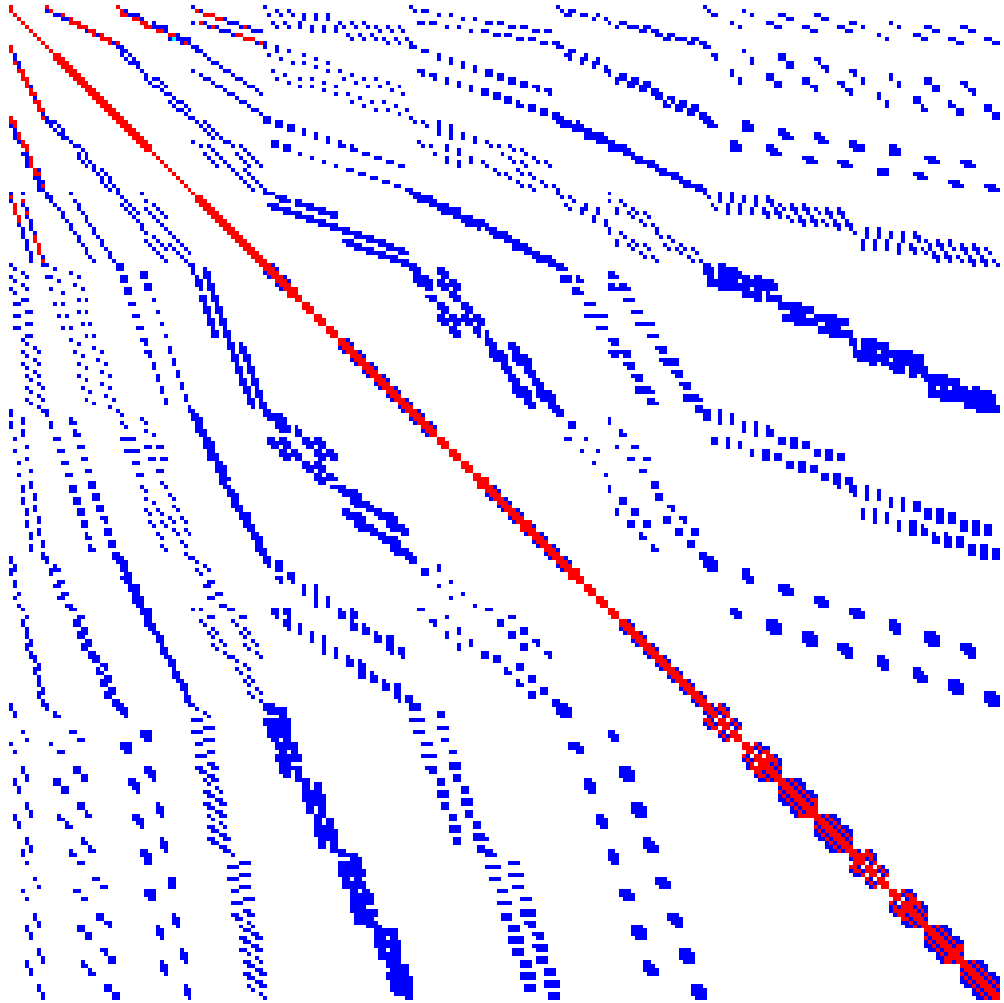
memory bandwidth limited

- Current architectures achieve at best 4% (Intel) and 20% (AMD) of peak flops
- Huge amount of architecture-specific effort [Oliker et. al., 2008]
  - explicit threads
  - thread affinity
  - hand-tuned SSE
  - software prefetch
  - TLB blocking
  - compression
  - array padding for cache coherence



- Only 50% to 60% of peak memory bandwidth, due to irregular access and coherence issues
- In practice, about half is achieved by good library code: 2%/10% flops, 25% bandwidth

Naïve high order methods lead to extremely dense matrices



- This matrix comes from first-order discretization of 5^3 subelements on each of 2^3 elements
- High-order matrix is the sum of 8 dense $6^3 \times 6^3$ matrices

Removing the memory bottleneck: exploiting the tensor product

$$u(x,y,z) = \sum_{i,j,k=0}^{p,q,r} \hat{u}_{ijk} h_i^p(x) h_j^q(y) h_k^r(z)$$
$$= \sum_{i=0}^{p} \left\{ \sum_{j=0}^{q} \left[\sum_{k=0}^{r} \hat{u}_{ijk} h_k^r(z) \right] h_j^q(y) \right\} h_i^p(x)$$

- Quadrature points are also a tensor product
- B, D_x, D_y become $\mathcal{O}(p^{d+1})$ operations, $\mathcal{O}(p^d)$ memory
  - Extremely regular computational kernel
- Application of the Jacobian is like function evaluation
- Jacobian needs values of $\eta, \eta', \nabla u$ stored at quadrature points, only $\mathcal{O}(p^d)$ space, computed for free during function evaluation

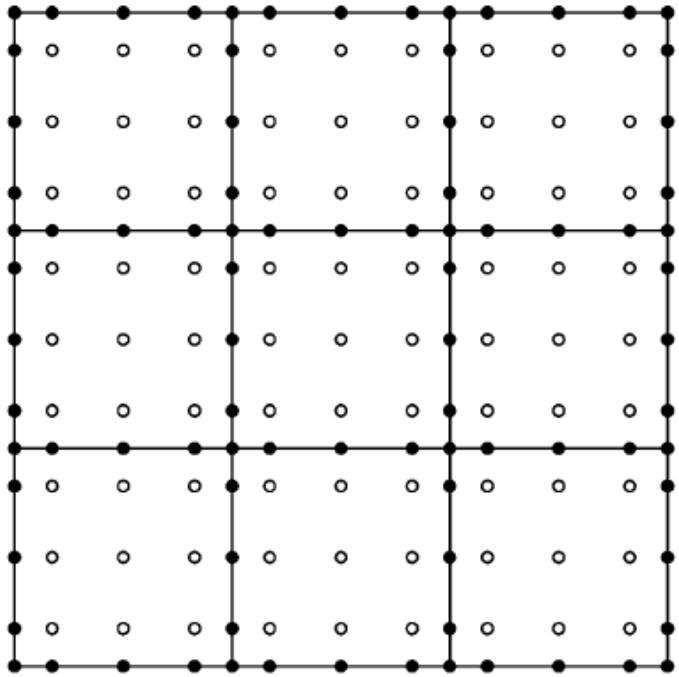
Jed: show code

Preconditioning

- Use piecewise (bi-/tri-) linear approximation on subelements
- Assemble matrix for same/similar bilinear form on subelements
- Assembled matrices need not satisfy inf-sup stability conditions, they are only used for preconditioning

Prior work

- Deville and Mund, 1985: Chebyshev collocation
- Fischer, 1997: Spectral element, incompressible flow
- Kim, 2007: Proof of spectral equivalence



Performance

Benchmark problem

3D Poisson problem in $(-1, 1)^3$. PETSc, GMRES, ML-5.0/6.2

- **Dohp Q5** 20^3 Hexes / Q5 nodal Legendre elements, preconditioned with Q1 finite elements on the LGL nodes
- **Libmesh Q2** 50^3 Hexes / Q2 Lagrange elements
- **Dohp Q1** Same 20^3 Hexes as above but apply the solver to the Q1 preconditioning matrix
- **Libmesh Q1** 100^3 Hexes / Q1 Lagrange elements

Event	Dohp Q5	Libmesh Q2	Dohp Q1	Libmesh Q1
Assembly	**16.8**	26.4	**17.1**	50.7
MatMult	26.7	28.3	**5.08**	7.25
PCSetUp	8.3	14.7	8.47	**6.9**
PCApply	25.8	88.4	**13.8**	23.2
KSPSolve	58.3	110.7	**22.3**	31.5
Peak memory (MB)	**1194**	2300	**1044**	1700
Krylov it. count	29	21	12	**9**