Optimizing Local Matrix Computation for Finite Element Methods

Robert C. Kirby and Andy R. Terrel

with Anders Logg and L. Ridgway Scott

University of Chicago, Chicago, IL

Introduction

Several ongoing projects have led to the development of tools for automating important aspects of the finite element method, with the potential for increasing code reliability and decreased development time. While these tools are effective at exploiting modern software engineering to produce workable systems, we believe that additional mathematical insight will lead to even more powerful codes with more general approximating spaces and more powerful algorithms.

One such optimization is how to efficiently (in the sense of operation count) evaluate local stiffness matrices for finite element methods. All entries of the local stiffness matrix for an element e may be expressed as the contraction of some reference element tensor with a "geometric" tensor. To build the $n \times n$ stiffness matrix for one element, one must contract this tensor with n^2 reference element tensors. There are relations between many of these tensors (equality, colinearity, small Hamming distance) that, if exploited, lead to an algorithm with significantly fewer floating point operations.

Finite Element Method

The finite element method is a general methodology for the discretization of differential equations. A discrete version of the variational problem: Find $U \in V$ such that

$$a(v, U) = L(v) \quad \forall v \in \hat{V}, \tag{1}$$

or a multilinear form:

$$A_i = a(\varphi^1_{i_1}, \varphi^2_{i_2}, \ldots, \varphi^r_{i_r}). \tag{2}$$

The element tensor:

$$A^e_i = a_e(\varphi^{e,1}_{i_1}, \varphi^{e,2}_{i_2}, \ldots, \varphi^{e,r}_{i_r}). \tag{3}$$

where $\{\varphi^1_i\}_{i=1}^{M_1}, \{\varphi^2_i\}_{i=1}^{M_2}, \ldots, \{\varphi^r_i\}_{i=1}^{M_r}$ are bases of $V_1, V_2, \ldots, V_r$ and $i = (i_1, i_2, \ldots, i_r)$ is a multiindex.

Building Stiffness Matrix

After some work, we can phrase parts of the Local Stiffness Matrix to a tensor contraction:

$$A_i^e = A_{i\alpha}^{0,k} G_{e,k}^{\alpha}, \tag{4}$$

For example, Poisson's equation $-\Delta u(x) = f(x)$ with homogeneous Dirichlet boundary conditions on a domain Ω. Here E is the reference element.

$$A_{i\alpha}^0 = \int_E \frac{\partial \Phi_{i_1}^1(X)}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2(X)}{\partial X_{\alpha_2}} \,\mathrm{d}X \;,\; G_e^{\alpha} = \det F_e' \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \tag{5}$$

We want to know how to speed this operation up.

Abstract Optimization

We consider tensors as vectors and contraction as the Euclidean inner product.

We let $Y = \{y^i\}_{i=1}^n$ be a collection of n vectors in $\mathbb{R}^m$, allowing for some of the items in Y to be identical.

Corresponding to Y, we must find a process of computing for arbitrary $g \in \mathbb{R}^m$ the collection of items $\{\left(y^i\right)^t g\}_{i=1}^n$.

We will measure the cost of this as the total number of multiply-add pairs required to complete all the dot products. This cost is always bounded by nm, but we hope to improve on that.

Complexity Reducing Relations

Dependencies such as equality, colinearity, and Hamming distance often occur between the different vectors.

Let $\rho : Y \times Y \to [0, m]$. We say that ρ is *complexity-reducing* if for every $y, z \in Y$ and arbitrary vector g with $\rho(y, z) \leq k < m$, $y^t g$ may be computed using the result $z^t g$ in no more than k multiply-add pairs.

Using these complexity reducing relations between vectors will allow us to reduce the time spent multiplying our matrices to build the stiffness matrix.

Some Relations

Example 1 *Let* $c(y,z) = \{0,$ *if* $y = z; 1,$ *if* $y = \alpha z \text{for some} \alpha \in \mathbb{R}, \alpha \neq 0, 1; m,$ *otherwise .*
Then c *is complexity-reducing, for* $y^t g = (\alpha z)^t g = \alpha(z^t g)$*, so* $y^t g$ *may be computed with one additional floating point operation once* $z^t g$ *is known.*

Example 2 *Let* $H^+(y,z)$ *be the Hamming distance, the number entries in which* y *and* z *differ. If* y *and* z *differ at* k *entries, so the difference* $y - z$ *has only* k *nonzero entries. Hence,* $(y-z)^t g$ *costs* k *multiply-add pairs to compute, and we may write* $y^t g = (y-z)^t g + z^t g$*. By the same argument, we can let* $H^-(y,z) = H^+(y,-z)$.

Complexity Reducing Relations

Theorem 1 *Let ρ_1 and ρ_2 be complexity-reducing relations. Define*

$$\rho(y, z) = \min(\rho_1(y, z), \rho_2(y, z)). \tag{6}$$

Then ρ is a complexity-reducing relation.

Proof. Pick $y, z \in Y$, let $1 \leq i \leq 2$ be such that $\rho(y, z) = \rho_i(y, z)$ and let $\rho_i(y, z) \equiv k$. But ρ_i is a complexity-reducing relation, so for any $g \in^m$, $y^t g$ may be computed in no more than $k = \rho(y, z)$ multiply-add pairs. Hence ρ is complexity-reducing. □

Laplacian Example

index	vector				index	vector				index	vector			
(0, 0)	3	3	3	3	(2, 0)	0	0	1	1	(4, 0)	0	0	-4	-4
(0, 1)	1	0	1	0	(2, 1)	0	0	-1	0	(4, 1)	0	0	0	0
(0, 2)	0	1	0	1	(2, 2)	0	0	0	3	(4, 2)	0	-4	0	-4
(0, 3)	0	0	0	0	(2, 3)	0	0	4	0	(4, 3)	-8	-4	-4	0
(0, 4)	0	-4	0	-4	(2, 4)	0	0	-4	-4	(4, 4)	8	4	4	8
(0, 5)	-4	0	-4	0	(2, 5)	0	0	0	0	(4, 5)	0	4	4	0
(1, 0)	1	1	0	0	(3, 0)	0	0	0	0	(5, 0)	-4	-4	0	0
(1, 1)	3	0	0	0	(3, 1)	0	0	4	0	(5, 1)	-4	0	-4	0
(1, 2)	0	-1	0	0	(3, 2)	0	4	0	0	(5, 2)	0	0	0	0
(1, 3)	0	4	0	0	(3, 3)	8	4	4	8	(5, 3)	0	-4	-4	-8
(1, 4)	0	0	0	0	(3, 4)	-8	-4	-4	0	(5, 4)	0	4	4	0
(1, 5)	-4	-4	0	0	(3, 5)	0	-4	-4	-8	(5, 5)	8	4	4	8

Element matrix indices and associated tensors displayed as vectors for the Laplacian.

Using symmetry

Because the element stiffness matrix is symmetric, we only need to build the triangular part. And for every element e, G_e is symmetric. Thus a contraction can be performed in $\binom{m+1}{2}$ rather than m^2 entries. In the two-dimensional case, we contract a symmetric 2×2 tensor G with an arbitrary 2×2 tensor K:

$$\begin{aligned} G : K &= \begin{pmatrix} G_{11} & G_{12} \\ G_{12} & G_{22} \end{pmatrix} : \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} \\ &= G_{11}K_{11} + G_{12}(K_{12} + K_{21}) + G_{22}K_{22} \\ &= \tilde{G}^t \hat{K} \end{aligned} \tag{7}$$

Laplacian Example using symmetry

index	vector			index	vector		
(0, 0)	3	6	3	(2, 2)	0	0	3
(0, 1)	1	1	0	(2, 3)	0	4	0
(0, 2)	0	1	1	(2, 4)	0	-4	-4
(0, 3)	0	0	0	(2, 5)	0	0	0
(0, 4)	0	-4	-4	(3, 3)	8	8	8
(0, 5)	-4	-4	0	(3, 4)	-8	-8	0
(1, 1)	3	0	0	(3, 5)	0	-8	-8
(1, 2)	0	-1	0	(4, 4)	8	8	8
(1, 3)	0	4	0	(4, 5)	0	8	0
(1, 4)	0	0	0	(5, 5)	8	8	8
(1, 5)	-4	-4	0				

This transformation of the reference tensor will not destroy any dependencies. Moreover, the transformation may introduce additional dependencies.

For example, before applying the transformation, entries (0,1) and (1,5) are not closely related by Hamming distance or colinearity, afterwards they are colinear.

Graph Optimizations

After obtaining the relationships, optimization becomes important to achieve the minimum multiply-add cost. We create an undirected, weighted graph $G(V, E)$ in order to optimize and employ the relationships.

The vertices ($v \in V$) are indexes of the vectors in Y. There is an edge ($\{v_1, v_2\} \in E$) between two vertices if there is a dependency between those two vectors. We associate a weight with each edge equal to the multiply-add cost of dotting one vector based on the result from the other.

Minimum Spanning Tree

To minimize the multiply-add pairs, we find a minimum spanning tree. The minimum spanning tree is a graph $T = (V, E')$ such that T is has $n - 1$ edges where n is the number of vertices, and the sum of edge weights in E' is minimal over all spanning trees.

Here we pick any vertex ($v_0 \in V$) and follow the edges in E' to trace to all the vertices in our original graph. It is minimum because it picks the path of minimum weights that visits every vertex of the graph, or in our case the minimum multiply-add pairs from one vector's dot product to any other vector's dot product.

Laplacian Example

Here is the minimum spanning tree of the Laplacian example above computed using the standard Prim's algorithm.

Using relationships of higher arity

If we can utilize relationships between two vectors, we should also be able to do the same with 3 or more vectors. Two factors complicate this use:

1. They are more expensive to search for. Best case scenario is quadratic.
2. Finding an optimized computation becomes more difficult, $G(Y, E)$ is now a *hypergaph*.

Planar Search

After the previous search, Y is our set of vectors such that no two elements are colinear. From Y we want to locate the set of all planes $\mathcal{P}$ such that:

1. If for any distinct $x, y, z \in Y$ with $\dim\left(\mathrm{span}\{x, y, z\}\right) = 2$, then there must exist a unique $P \in \mathcal{P}$ with $\{x, y, z\} \subset P$.
2. For each $P \in \mathcal{P}$, the dimension of $\mathrm{span}(P)$ must be exactly 2.
3. The planes are *maximal* in the sense that for all distinct $P, Q \in \mathcal{P}$, $|P \cap Q| \leq 1$. If $|P \cap Q| \geq 2$, then they define the same plane.

We can form this in $O(n^3)$ by enumerating the triples of Y.

Improved Algorithm

The algorithm can be taken to $O(n^2)$ by reducing the problem to the search above.

- Let $\Pi : \mathbb{R}^m \rightarrow \mathbb{R}^3$ be a random projection
- If (y^i, y^j) and $(y^{i'}, y^{j'})$ are coplanar, then $(\Pi y^i, \Pi y^j)$ and $(\Pi y^{i'}, \Pi y^{j'})$ are as well
- But if $(\Pi y^i, \Pi y^j)$ and $(\Pi y^{i'}, \Pi y^{j'})$ are coplanar, then $\Pi y^i \times \Pi y^j$ and $\Pi y^{i'} \times \Pi y^{j'}$ are colinear
- Search for colinearity among $O(n^2)$ pairs –> $O(n^2)$ complexity by hashing
- Generalizes to k^{th} order dependencies

Exploiting Dependencies

In order to minimize cost, we want to find the smallest set of vectors whose dependencies will compute the largest number of vectors in Y based on $\mathcal{P}$.

We let $Y^k \subseteq Y$ be elements of Y that lie in at least one hyperplanar relations of degree $k-1$.
Let $S \subset Y^k$. We define the *k-hyperplanar closure* of S, denoted Y_S^k, as the closure of S under k^{th} order linear combinations. That is,

1. If $y \in S$, then $y \in Y_S^k$.
2. For any $y \in Y^k$, if $\exists Z = \{z_i\}_{i=1}^k \subset Y_S^k$ such that $y \in \text{span}(Z)$, then $y \in Y_S^k$.

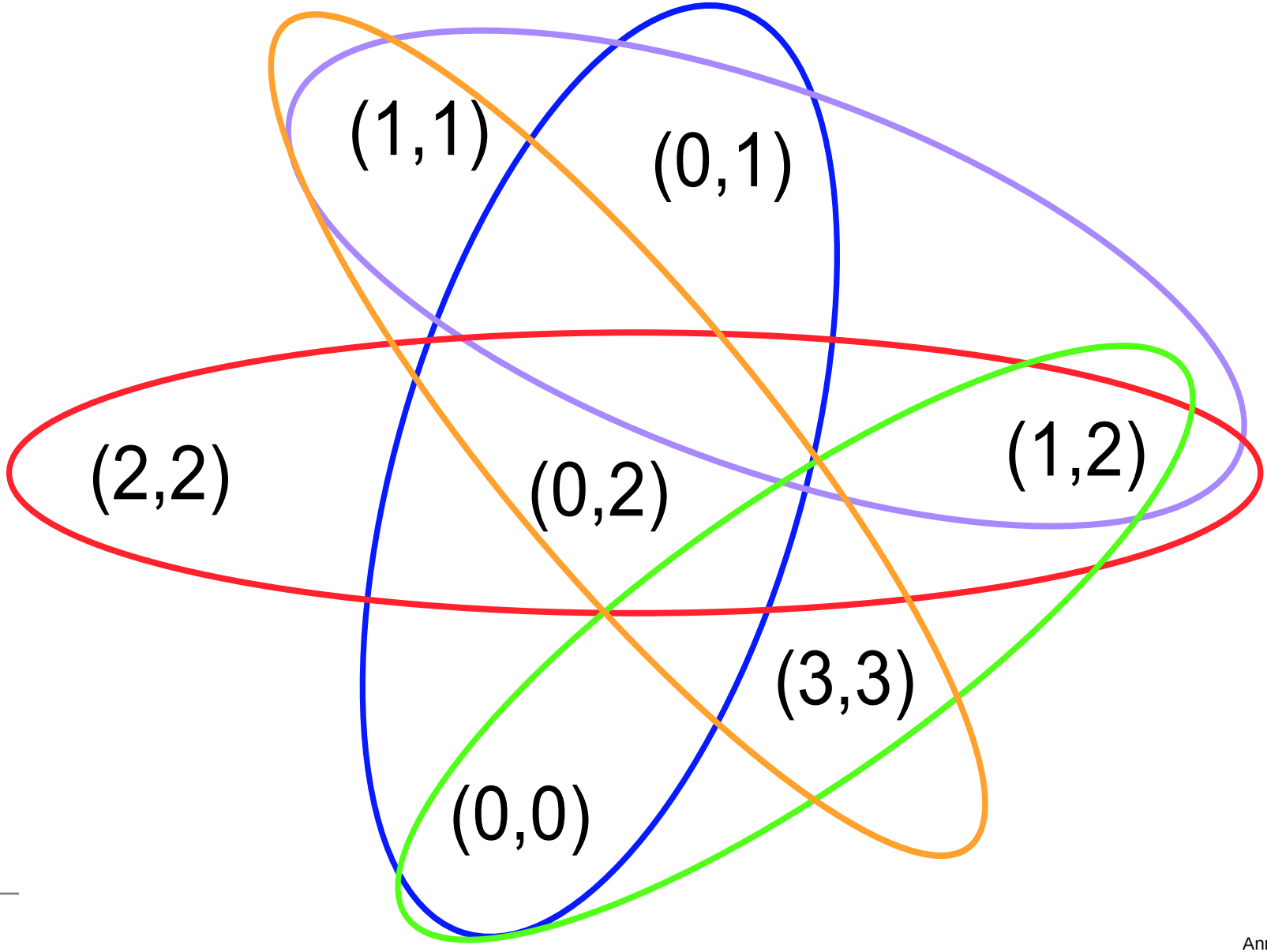
Thus we need to acheive $Y_S^k = Y^k$, while minimizing $|S|$.

Hypergraph structure

Abstractly, we can think of the vectors in a k-hypergraph $H(V, E)$ where $v \in V$ are vector from Y that are in hyperplanar relations and $v_i{}_{i=1}^{k} \in E$ are sets of vectors in the hyper planar relation.

Below is an example from the Laplacian we have been using picking out all the planar relations $k = 3$. The edge sets are circled, with the indexes as the vertices. Here is one possible S: $S = \{(0, 0), (0, 2), (1, 2)\}$

Hypergraph example



Geometric Algorithm

Our algorithm builds up S, Y_S^k by:

1. Selecting the 'most connected' member to add to S,
2. adding any vectors in newly completed planes to Y_S^k
3. Repeat until $Y_S^k = Y^k$.

To pick the 'most connected' member we are using a heuristic that selects an unprocessed vector that has the most unprocessed neighbors.

Results

The results here are on optimizing the Laplacian on tetrahedra using symmetry.

degree	n	m	nm	binary	geometric
1	10	6	60	27	54
2	55	6	330	101	128
3	210	6	1260	370	352

n the number of vectors, m the length of vectors, nm the multiply-add pairs without optimization, binary and geometric are the multiply-add pairs using our optimizations.

Conclusions

Here we have two approaches for optimizing the building of the stiffness matrix for finite element methods, one based on generating a minimum spanning tree of an appropriately weighted graph, and the other based on exploiting linear dependences of increasing degrees.

In ongoing work, we are considering how to get better performance in terms of speed and maximum problem size by either parallelizing our Python code or else developing a C++ version. Moreover, we are studying ways of bringing the binary relations together with the linear dependence relations through generalizations of Prim's algorithm.