Combinatorial Optimization of Matrix-Vector Multiplication for Finite Element Assembly*

Michael M. Wolf† Michael T. Heath†

February 5, 2009

Abstract

It has been shown that combinatorial optimization of matrix-vector multiplication can lead to faster evaluation of finite element stiffness matrices. Based on a graph model characterizing relationships between rows, an efficient set of operations can be generated to perform matrix-vector multiplication for this problem. We improve the graph model by extending the set of binary row relationships and solve this combinatorial optimization problem optimally for the binary row relationships implemented, yielding significantly improved results over previous published graph models. We also extend the representation by using hypergraphs to model more complicated row relationships, expressing a three-row relationship with a three-vertex hyperedge, for example. Our initial greedy algorithm for this hypergraph model has yielded significantly better results than the graph model for many matrices.

1 Introduction

1.1 Finite Element Compilers

The motivation behind this work comes from "compilers" for finite element methods in the FEniCS project [5, 6, 8, 7]. The construction of finite element stiffness matrices for large unstructured problems can be costly in terms of execution time, especially for higher-order methods. The standard algorithm for constructing local stiffness matrices through naive integration is suboptimal in operation count and can be greatly improved by creating a "compiler" for the construction of these matrices. This "compiler" identifies a reduced set of operations to use in the construction of these matrices for a given equation and set of basis functions [6, 8]. Optimization of local stiffness matrix assembly is important because local stiffness matrices are generated for each element. Thus, a billion element finite element problem yields a billion local stiffness matrices. Kirby, et al. modify the local stiffness matrix assembly algorithm and introduce a tensor formulation (see subsection 1.2) that makes it easier to identify redundant operations [6]. In the software project FErari, they implement methods based on this formulation to reduce these redundant operations and generate a reduced set of instructions to assemble the local stiffness matrices [6, 8].

1.2 Local Stiffness Matrix Tensor Formulation

The entries in a local stiffness matrix for a given element (see appendix for derivation of an example) can be written as the Frobenius product of a tensor and a matrix $(\mathbf{K}_{i,j} : \mathbf{G}^e)$[6]. Let e be a given element in the domain with coordinates ξ and $\hat{e}$ be the reference element with coordinates $\hat{\xi}$. For the simple example

*This work was supported by the DOE CSGF, DE-FG02-97ER25308

†Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin, Urbana, IL, 61801 (mmwolf@uiuc.edu, heath@uiuc.edu).

below, we assume that the map from the reference element is affine. In this tensor formulation of the 2D Laplace equation, for example, the local stiffness matrix $\mathbf{S}^e$ is given by

$$\mathbf{S}^e_{i,j} = \sum_{m=1}^{2}\sum_{n=1}^{2} \mathbf{G}^e_{m,n}\mathbf{K}_{i,j,m,n} = \mathbf{K}_{i,j} : \mathbf{G}^e,$$

where

$$\mathbf{G}^e = \begin{bmatrix} \left[\det(\mathbf{J})\left(\frac{\partial\hat{\xi}_1}{\partial\xi_1}\frac{\partial\hat{\xi}_1}{\partial\xi_1} + \frac{\partial\hat{\xi}_1}{\partial\xi_2}\frac{\partial\hat{\xi}_1}{\partial\xi_2}\right)\right]_e & \left[\det(\mathbf{J})\left(\frac{\partial\hat{\xi}_1}{\partial\xi_1}\frac{\partial\hat{\xi}_2}{\partial\xi_1} + \frac{\partial\hat{\xi}_1}{\partial\xi_2}\frac{\partial\hat{\xi}_2}{\partial\xi_2}\right)\right]_e \\ \left[\det(\mathbf{J})\left(\frac{\partial\hat{\xi}_2}{\partial\xi_1}\frac{\partial\hat{\xi}_1}{\partial\xi_1} + \frac{\partial\hat{\xi}_2}{\partial\xi_2}\frac{\partial\hat{\xi}_1}{\partial\xi_2}\right)\right]_e & \left[\det(\mathbf{J})\left(\frac{\partial\hat{\xi}_2}{\partial\xi_1}\frac{\partial\hat{\xi}_2}{\partial\xi_1} + \frac{\partial\hat{\xi}_2}{\partial\xi_2}\frac{\partial\hat{\xi}_2}{\partial\xi_2}\right)\right]_e \end{bmatrix}$$

is an element-dependent matrix, $\mathbf{K}$ is an element-independent tensor with entries

$$\mathbf{K}_{i,j} = \begin{bmatrix} \left(\frac{\partial\phi_i}{\partial\hat{\xi}_1}, \frac{\partial\phi_j}{\partial\hat{\xi}_1}\right)_{\hat{e}} & \left(\frac{\partial\phi_i}{\partial\hat{\xi}_1}, \frac{\partial\phi_j}{\partial\hat{\xi}_2}\right)_{\hat{e}} \\ \left(\frac{\partial\phi_i}{\partial\hat{\xi}_2}, \frac{\partial\phi_j}{\partial\hat{\xi}_1}\right)_{\hat{e}} & \left(\frac{\partial\phi_i}{\partial\hat{\xi}_2}, \frac{\partial\phi_j}{\partial\hat{\xi}_2}\right)_{\hat{e}} \end{bmatrix},$$

and

$$\mathbf{J} = \begin{bmatrix} \frac{\partial\xi_1}{\partial\hat{\xi}_1} & \frac{\partial\xi_1}{\partial\hat{\xi}_2} \\ \frac{\partial\xi_2}{\partial\hat{\xi}_1} & \frac{\partial\xi_2}{\partial\hat{\xi}_2} \end{bmatrix}$$

is the Jacobian of the affine mapping. For this 2D problem, the matrices $\mathbf{G}^e$ and $\mathbf{K}_{i,j}$ have $s = 2$ rows and columns. For a 3D problem, these matrices would have $s = 3$ rows and columns. The number of basis functions needed for the 2D problem when using order p basis functions is $q = (p+1)(p+2)/2$, which means the tensor $\mathbf{K}$ has $((p+1)(p+2)/2)^2$ entries [4]. The number of basis functions needed for the 3D problem when using order p basis functions is $q = (p+1)(p+2)(p+3)/6$, which means the tensor $\mathbf{K}$ has $((p+1)(p+2)(p+3)/6)^2$ entries [4].

1.3 Matrix-Vector Multiplication

We can write the above local stiffness matrix $\mathbf{S}^e$ as a vector $\mathbf{y}$ (such that $y_{qi+j} = \mathbf{S}^e_{i,j}$). It follows that we can rewrite the above tensor stiffness matrix formulation as a matrix-vector multiplication operation $\mathbf{y} = \mathbf{A}\mathbf{x}^e$, where $\mathbf{A}$ is an element-independent matrix such that $\mathbf{A}_{qi+j,sk+l} = \mathbf{K}_{i,j,k,l}$ and $\mathbf{x}^e$ is an element-dependent vector such that $x^e_{sk+l} = \mathbf{G}^e_{kl}$. Throughout this paper, we use the convention that $\mathbf{r}_i$ is the vector whose transpose is the ith row of the matrix $\mathbf{A}$, $\mathbf{x}$ is the vector to be multiplied, and $\mathbf{y}$ is the vector resulting from matrix-vector multiplication. Thus, each entry in the vector $\mathbf{y}$ is the result of the inner product

$$y_{qi+j} = \mathbf{S}^e_{i,j} = \mathbf{A}_{(qi+j,*)}\mathbf{x}^e = \mathbf{r}^T_{qi+j}\mathbf{x}^e.$$

For the 2D Laplace equation, for example, each entry y_{qi+j} is the inner product of the vectors

$$\mathbf{A}^T_{(qi+j,*)} = \begin{bmatrix} \left(\frac{\partial\phi_i}{\partial\hat{\xi}_1}, \frac{\partial\phi_j}{\partial\hat{\xi}_1}\right)_{\hat{e}} \\ \left(\frac{\partial\phi_i}{\partial\hat{\xi}_1}, \frac{\partial\phi_j}{\partial\hat{\xi}_2}\right)_{\hat{e}} \\ \left(\frac{\partial\phi_i}{\partial\hat{\xi}_2}, \frac{\partial\phi_j}{\partial\hat{\xi}_1}\right)_{\hat{e}} \\ \left(\frac{\partial\phi_i}{\partial\hat{\xi}_2}, \frac{\partial\phi_j}{\partial\hat{\xi}_2}\right)_{\hat{e}} \end{bmatrix}, \quad \mathbf{x}^e = \det(\mathbf{J}) \begin{bmatrix} \frac{\partial\hat{\xi}_1}{\partial\xi_1}\frac{\partial\hat{\xi}_1}{\partial\xi_1} + \frac{\partial\hat{\xi}_1}{\partial\xi_2}\frac{\partial\hat{\xi}_1}{\partial\xi_2} \\ \frac{\partial\hat{\xi}_1}{\partial\xi_1}\frac{\partial\hat{\xi}_2}{\partial\xi_1} + \frac{\partial\hat{\xi}_1}{\partial\xi_2}\frac{\partial\hat{\xi}_2}{\partial\xi_2} \\ \frac{\partial\hat{\xi}_2}{\partial\xi_1}\frac{\partial\hat{\xi}_1}{\partial\xi_1} + \frac{\partial\hat{\xi}_2}{\partial\xi_2}\frac{\partial\hat{\xi}_1}{\partial\xi_2} \\ \frac{\partial\hat{\xi}_2}{\partial\xi_1}\frac{\partial\hat{\xi}_2}{\partial\xi_1} + \frac{\partial\hat{\xi}_2}{\partial\xi_2}\frac{\partial\hat{\xi}_2}{\partial\xi_2} \end{bmatrix}^e.$$

Thus, optimizing the construction of the finite element local stiffness matrices can be generalized to optimizing matrix-vector multiplication. For the 2D Laplace equation, we see above that the vector $\mathbf{x}$ has 4 entries (corresponding to the 4 elements in the 2-by-2 matrix $\mathbf{G}^e$) and matrix $\mathbf{A}$ has four columns (corresponding

to the 4 entries in the tensor blocks $\mathbf{K}_{i,j}$). For order p basis functions, matrix $\mathbf{A}$ has $((p+1)(p+2)/2)^2$ rows (corresponding to the number of matrices $\mathbf{K}_{i,j}$ in $\mathbf{K}$). However, for this problem, we can exploit symmetry to reduce the size of the matrix and vector, resulting in a matrix $\mathbf{A}$ with 3 columns and a vector $\mathbf{x}$ with 3 entries [6]. Symmetry can also be used to reduce the number of rows in $\mathbf{A}$ to $(p^4+6l^3+15p^2+18p+8)/8$. For the 3D Laplace equation, the vector $\mathbf{x}$ has 9 entries (corresponding to the 9 elements in the 3-by-3 matrix $\mathbf{G}^e$) and matrix $\mathbf{A}$ has 9 columns (corresponding to the 9 entries in the tensor blocks $\mathbf{K}_{i,j}$). For order p basis functions, matrix $\mathbf{A}$ has $((p+1)(p+2)(p+3)/6)^2$ rows (corresponding to the number of matrices $\mathbf{K}_{i,j}$ in $\mathbf{K}$. Again, we can exploit symmetry to reduce the number of columns in matrix $\mathbf{A}$ and entries in vector $\mathbf{x}$ to 6. We can similarly use symmetry to reduce the number of rows in $\mathbf{A}$ to $(p^6+12p^5+58p^4+150p^3+229p^2+198p+72)/72$.

1.4 Optimization Problem

We wish to minimize the number of operations needed to compute the vector resulting from matrix-vector multiplication. We use the number of *multiply-add pairs* (MAPs) as our metric for counting the number of operations required for the matrix-vector product. Since there is usually one more multiplication than addition operation needed to determine each entry of the resulting vector, we determine the number of MAPs by counting multiplications . For example, we would count $y_1 = 2.5y_2 + 2x_2$ as 2 MAPs. However, in order to be consistent with the FErari MAP counting, we assume a fused multiply-add operation that can perform multiply-add operations where only one multiply is required in one operation. Thus, we count $y_1 = y_2 + 2x_2$ or $y_1 = -y_2 + 2x_2$ as 1 MAP. Whether this is a good assumption for this type of operation is architecture and compiler dependent, and experimentally we find that it is more accurate to count this operation as 2 MAPs on some systems. For instance, we calculated this operation to cost 1.4 MAPs (Mac Intel Core 2 Duo, gcc), 1.5 MAPs (Dual-core AMD Opteron, gcc), and 1.6 MAPs (Apple Xserve G5 processor, gcc) on the three systems with which we experimented. For more flexibility and accuracy, we should assign a system-dependent fractional value between 1 MAP and 2 MAPs for this type of operation.

In order to minimize the number of MAPs needed to compute the matrix-vector product, we utilize relationships between rows in the matrix. These relationships allow us to replace full inner-product operations, which yield each resulting vector entry of the matrix-vector product, with less costly operations. We implement methods to generate operations based on the binary row relationships utilized in FErari and on additional binary row relationships not exploited in FErari. In addition, we also extend our methods to generate operations based on row relationships that relate more than two rows.

2 Graph Model

In our first attempt at minimizing the number of MAPs needed to calculate the vector resulting from a matrix-vector product, we used a weighted graph representation to model the problem, similar to that used in FErari [6, 8]. In this model, the resulting vector entries are represented by vertices in the graph, with some vector entries excluded. We exclude resulting vector entries corresponding to rows that contain only zeros in the graph, since the corresponding inner product for these rows is zero, and thus no operations are needed to compute this vector entry. This optimization saves 1 MAP for each column in the matrix over the naive matrix-vector multiplication algorithm for each zero row but saves nothing over an unoptimized matrix-vector multiplication algorithm that ignores the zero entries when calculating the matrix-vector product. Similarly, when identical rows occur in the matrix, we represent only one of the vector entries corresponding to the identical rows with a vertex in the graph since only one of the corresponding inner products needs to be calculated (see subsection 2.3).

We represent operations resulting from relationships between two rows by edges connecting the vertices corresponding to the two resulting vector entries involved. We store the operations for each edge needed to obtain each resulting vector entry given the other resulting vector entry, assigning weights to the edges equal to the number of MAPs needed for the corresponding operations. This is demonstrated in Figure 1 in which the resulting vector entries corresponding to rows 1 and 2 are represented by vertices 1 and 2, respectively. The operations relating y_1 and y_2 ($y_1 = -y_2/2$ and $y_2 = -2y_1$) are represented by the edge

connecting vertices 1 and 2. The direction of the edge determines which operation is used. However, in our graph model, we solve an undirected graph problem before assigning the edge directions since it reduces the number of edges built without affecting the optimality of the solution. Subsequently, we assign the undirected edges of the solution a direction based on a traversal of the solution graph.

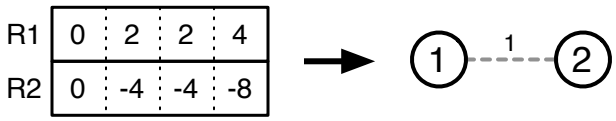

Figure 1: Two rows and subgraph that represents resulting vector entries and their relationship. Edge represents possible operations $y_1 = -y_2/2$ or $y_2 = -2y_1$, depending on direction.

The optimizations we implement (and represent as edges in our graph model) based on binary row relationships fall into one of three categories described in subsections 2.2-2.4. The implemented optimizations described in subsections 2.2 and 2.3 are similar to those in FErari, but the optimizations described in subsections 2.1 and 2.4 have no equivalents in FErari.

2.1 Inner-Product Operations

We include in our graph model a special *inner-product* vertex that does not correspond to any resulting vector entry. This inner-product vertex is connected to all vector entry vertices in the graph by edges that represent operations needed to calculate the inner product for the corresponding vector entry. The inner-product operations ignore zero entries in the row, using only nonzero entries when forming the inner product. Thus, the weight of an inner-product edge is equivalent to the number of nonzeros in the row corresponding to the vector-entry vertex of the edge (the vertex that is not the inner-product vertex).

FErari does not have an equivalent to the inner-product vertex and edges. This represents a difference in scope for the two respective optimization problems. The FErari optimization problem includes binary relationships between rows, and once the optimal set of operations is found from this problem, inner-product operations are added where necessary or when they are cheaper than the binary relationship operations. We include in our optimization problem inner-product relationships as well as binary relationships, so our solution includes the necessary inner products once the optimization problem is solved. We also view the inner-product vertex as a natural starting point for solution traversal, which FErari lacks. At least one inner-product operation must be calculated before any other optimizations based on components of the resulting vector can be utilized. This corresponds to traversing an inner-product edge. Any graph traversal starting from the inner-product vertex that spans the graph vertices will result in operations that compute the matrix-vector product correctly. Such a traversal is guaranteed to exist, since it can be formed by inner-product edges alone.

2.2 Colinear Row Relationship

One simple binary row relationship is when two rows are scalar multiples of each other, so the vectors formed from these rows are colinear (e.g., $\mathbf{r}_i = \alpha \mathbf{r}_j$). This relationship yields an optimization for calculating the corresponding resulting vector entries in which the resulting vector entry for one of the rows can be determined by scaling the vector entry corresponding to the colinear row at a cost of 1 MAP (e.g., $\mathbf{r}_i = \alpha \mathbf{r}_j \Rightarrow y_i = \alpha y_j$). For instance, in Figure 2, we see that $\mathbf{r}_2 = 1.5\mathbf{r}_1$, and thus it follows that the resulting vector entry $y_2 = 1.5y_1$. This colinear relationship is always optimal, and in theory all but one of the resulting vector entry vertices resulting from a colinear set of rows can be removed from the graph. This may not be the case in practice, however, since one vertex may yield a less costly solution when *partial* colinear row edges (see subsection 2.4) are considered, based on whether the partial colinear operations contain coefficients of unit magnitude. Since we cannot determine *a priori* which vector entry should not be calculated using the colinear optimization

(corresponding to which vertex should remain in the graph), we cannot remove any of these vertices from the graph.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 0 \\ 3 & 3 & 3 & 0 \\ 2 & 2 & 2 & 0 \\ 5 & 5 & 5 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Optimization	Resulting Operations	Cost
identical rows	$y_3 = y_1$	0 MAPs
colinear rows	$y_2 = 1.5y_1$	1 MAP
partial colinear rows	$y_4 = 2.5y_1 + 8x_4$	2 MAPs

Figure 2: Optimizations based on colinear matrix row relationships.

2.2.1 Searching for Colinear Row Relationships

An important issue is how to find these colinear row relationships efficiently. For this and future analyses of the search algorithms, we assume we have a matrix with n rows and d columns. The brute force search algorithm compares all $O(n^2)$ row vector pairs to detect whether they are colinear. We can determine whether a pair of vectors is colinear in $O(d)$ time. Thus, the brute force algorithm has complexity $O(dn^2)$. Faster methods can be utilized, however. We have implemented an $O(dn \log(n))$ search method that inserts slightly modified row vectors into a binary search tree that produces a lexicographically sorted set of vectors. This is similar to a method mentioned in [8]. Before inserting the row vectors into the tree, they are normalized and scaled by -1 (if necessary) so that $-v$ and v are lexicographically equivalent. These modified row vectors are inserted into the binary tree so that they are sorted lexicographically in $O(dn \log(n))$ time. A traversal of this sorted tree allows us to find the lexicographically equivalent (within some tolerance) modified vectors that correspond to originally colinear row vectors and are now consecutively arranged in the sorted tree. We also found that presorting the vectors by their angle relative to some reference vector (a necessary but not sufficient condition for colinearity) further reduced the runtime of this search algorithm. Kirby, et al. used hash tables to determine the set of colinear vectors [8]. By looping through each row once and inserting the row into the hash table, they determine in constant time whether the row is collinear to a previously inserted row, for a total cost of $O(dn)$. This would be a more effective method for sufficiently large d or n.

2.3 Identical Row Relationship

The identical row relationship is a special case of the colinear row relationship in which the scaling factor $\alpha = 1$. This relationship yields an optimization in which the resulting vector entries corresponding to the two identical rows will be equivalent. Thus, with this optimization we need to compute only the resulting vector entry for one of a set of identical rows, and the remaining resulting vector entries need only assignment, as demonstrated in Figure 2 ($\mathbf{r}_3 = \mathbf{r}_1 \Rightarrow y_3 = y_1$). Since only assignment is needed to compute all but one of the resulting vector entries for a set of identical rows, we assign a cost of 0 MAPs for the remaining vector entries, a savings of 1 MAP per column over naive matrix-vector multiplication algorithms and 1 MAP for every nonzero in the row for an unoptimized algorithm that is clever enough to ignore the matrix zeros in the computation. This assignment of 0 MAPs is consistent with [6, 8, 9]. As with colinear rows, the identical row relationship operation is always optimal. Unlike colinear row optimization, however, the choice of the vector entry to be represented in the graph is unimportant, since all rows are calculating the same inner product. Thus, in order to simplify the graph problem, we remove all but one of the vertices corresponding to a set of identical vector entries. In reality, the relative cost of this assignment operation should be between 0 and 1 MAPs. However, since this operation should always be in the optimal solution (and thus the corresponding vertex is removed from the graph), the exact cost of this operation does not affect the solution.

2.4 Partial Colinear Row Relationship

The final binary row relationship we utilize in our graph model is the partial colinear row relationship. The partial colinear row relationship is similar to the colinear row relationship except that only a proper subset

of the entries in a row are scalar multiples of the corresponding entries in the other row (the vectors formed from the subset of entries of these two rows are colinear). For each pair of rows, there may be many different subsets of entries that form colinear vectors, but we represent only one relationship having the largest subset of entries (ignoring trivial subsets of one entry) that are colinear. We also only consider partial colinear row relationships that yield potentially beneficial operations, ignoring partial colinear row relationships that yield operations more expensive than corresponding inner-product operations, for instance. It is important to note that there is no equivalent binary row relationship implemented in FErari. FErari implements a Hamming distance optimization in which two rows have a subset of entries that are identical or negative of each other, a special case of the partial colinear row relationship [8].

We can naturally write the expression of this partial colinear relationship between two rows i and j as

$$\mathbf{r}_i = \alpha \sum_{k \in C_{col}} r_{jk}\mathbf{e}_k + \sum_{k \notin C_{col}} \beta_k \mathbf{e}_k,$$

where C_{col} is the subset of column indices for the entries in the two rows that are part of the colinear subset, α is the factor relating the colinear subsets, $\mathbf{e}_k$ is the kth column of the identity matrix, and β_k is a correction factor for the non-colinear entry k. This yields the possible optimization

$$\mathbf{r}_i^T\mathbf{x} = \alpha \sum_{k \in C_{col}} r_{jk}\mathbf{e}_k^T\mathbf{x} + \sum_{k \notin C_{col}} \beta_k \mathbf{e}_k^T\mathbf{x} \Rightarrow y_i = \alpha \sum_{k \in C_{col}} r_{jk}x_k + \sum_{k \notin C_{col}} \beta_k x_k.$$

The first term in this resulting operation is part of the inner-product operation for y_j. We could represent this partial inner product as an optional vertex in our graph model, which may or may not be included in the graph solution. However, in our implementation, we did not implement partial colinear operations in this manner. In order to simplify the graph model by removing the need for these optional vertices in our graph solution, we introduce a different expression for the row relationship that yields a slightly less optimal resulting optimization based on the same partial colinear relationship.

Unlike the original expression for the partial colinear row relationship, we write row i in terms of the entire row j,

$$\mathbf{r}_i = \alpha\mathbf{r}_j + \sum_{k \in \overline{C_{col}}} \gamma_k \mathbf{e}_k,$$

where $\overline{C_{col}}$ is the set of column indices for the entries in the two rows that are not a part of the colinear subset, and γ_k is a correction factor for the entry k that is not part of the colinear subset. This yields a possibly less optimal optimization in which the resulting vector entry y_i can be written in terms of the resulting vector entry y_j,

$$\mathbf{r}_i^T\mathbf{x} = \alpha\mathbf{r}_j^T\mathbf{x} + \sum_{k \in \overline{C_{col}}} \gamma_k \mathbf{e}_k^T\mathbf{x} \Rightarrow y_i = \alpha y_j + \sum_{k \in \overline{C_{col}}} \gamma_k x_k.$$

This new optimization relates two resulting vector entries and thus can be represented in the graph model with normal (not optional) vertices. However, this operation is not optimal for all partial colinear rows, in particular when $\beta_k = 0$ for some k in the first expression. For instance, if $\mathbf{r_i}^T = [4, 4, 4, 0, 4]$ and $\mathbf{r_j}^T = [1, 1, 0, 1, 1]$, the first partial colinear approach for calculating y_i would use the operation $y_i = 4t_j + 4x_3$, where $t_j = x_1 + x_2 + x_5$ is a partial inner product of y_j previously stored during the calculation of the full inner product, at a cost of 2 MAPs. The second partial colinear approach for calculating y_i, however, would use the operation $y_i = 4y_j + 4x_3 - 4x_4$ at a cost of 3 MAPs. In practice, we believe the partial colinear row relationships that lead to this lack of optimality in the second approach to be relatively uncommon, and thus we are willing to sacrifice some optimality for the simpler graph model.

This partial colinear optimization is more expensive than the fully colinear optimization but still potentially useful, costing $1 + \left|\overline{C_{col}}\right|$ MAPS ($\left|\overline{C_{col}}\right|$ MAPS if $\alpha = \pm 1$). For example, we see in Figure 2, that rows 1 and 4 form vectors that are partially colinear. The vector formed from the first three columns of row 4 is a scalar multiple of the vector formed from the first three columns of row 1, with $\alpha = 2.5$ as the scaling factor. Thus, the resulting vector entry corresponding to row 4 can be computed as $y_4 = 2.5y_1 + 8x_4$ at a cost of 2 MAPs, a savings of 2 MAPs when compared to the 4 MAPs required to do the complete inner product.

2.4.1 Searching for Partial Colinear Row Relationships

As with the colinear row relationships, an important issue is how to find the partial colinear row relationships in an efficient manner. We use a partial brute force method in which we loop over all $O(n^2)$ pairs of row vectors. For each row vector pair, we find the largest proper subset of entries that are colinear. We can do this by looping through the d entries in the two vectors, calculating the ratio between the two vector entries, and inserting these d ratios into a sorted binary search tree (in $O(d\log(d))$) time. We can traverse the tree to find the largest set of ratios that are the same (within some tolerance). The corresponding entries indicate the largest proper subset of entries that are colinear. In this manner, we find the partial colinear row relationships in $O(dn^2\log(d))$ time. It is possible that there is a method for reducing the $O(n^2)$ part of the complexity (as seen in subsubsection 2.2.1) but we leave this for future work.

2.5 Graph Model Minimization Problem

Finding the set of operations with a minimum number of MAPs (for the binary row relationships utilized) is equivalent to finding a set of edges in the graph of minimal weight with certain properties. The resulting edges must form a subgraph spanning all vertices, since each entry in the resulting vector must be calculated by either an inner product or by an operation resulting from a binary row relationship with a row corresponding to a previously calculated vector entry. This spanning subgraph must also be connected, so that there is at least one path from the inner-product vertex to every other vertex, and each resulting vector entry can be calculated by following such a path to its corresponding vertex. An optimal solution subgraph must also be acyclic, since for any solution subgraph with a cycle an edge could be removed, leaving a less expensive but valid solution. Thus, a resulting optimal solution must be a *minimum spanning tree* (MST) for the graph. We can find an optimal solution to this optimization problem easily, since greedy algorithms such as Prim's algorithm find the MST of a graph optimally in polynomial time. The cost of a minimal MAP set of operations for matrix-vector multiplication (for the relationships utilized) is equivalent to the sum of the edge weights in the resulting MST. In Figure 3, we see a simple graph and solution resulting from a two-row matrix to demonstrate how the MST yields a solution to the optimization problem. The matrix on the left yields the middle graph with two resulting vector entry vertices and the inner-product vertex. The two edges from the inner-product vertex correspond to computing the inner product (ignoring zero entries) for each row. The edge connecting the two row vertices corresponds to an operation that arises from a partial colinear relationship between the two rows ($y_1 = -y_2 + 8x_1$ or $y_2 = -y_1 + 8x_1$). We see an MST of the graph on the right with the edges of the MST corresponding to a set of operations for computing the matrix-vector multiplication with a minimal number of MAPs (4) for the relationships utilized,

$$\begin{aligned} y_2 &= -4x_2 - 4x_3 - 8x_4, \\ y_1 &= -y_2 + 8x_1. \end{aligned}$$

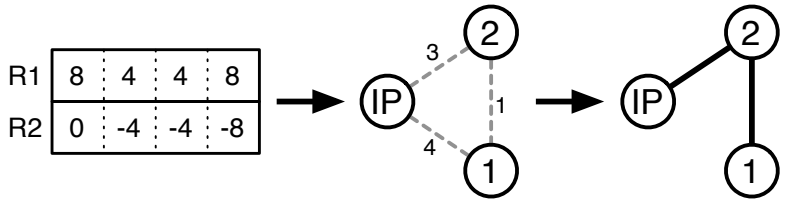

Figure 3: Simple two row matrix, corresponding graph problem, and resulting MST (4 MAPs) solution to graph problem.

It is important to note that if we had included the optional vertices described in subsection 2.4 for the partial colinear row relationships, the MST problem would be the wrong combinatorial optimization problem.

We believe the problem could instead be formulated as a weighted Steiner tree problem, another well known combinatorial optimization problem. The Steiner tree problem in graphs is to find the minimal cost tree that spans all the vertices in a given vertex set (required vertices) and may or may not span the remaining vertices (optional vertices)[11].

2.6 Graph Example

Figure 4 shows an example formulation and solution of a graph problem to generate less costly code for a matrix-vector product. We see in Figure 4(a) a simplified version of the matrix for building the finite element local stiffness matrices for the 2D Laplace equation (triangles and 2nd order Lagrange polynomial basis) with the zero rows, identical rows, and several additional rows removed in order to display the graphs more easily. Figure 4(b) shows the graph vertices corresponding to the resulting vector entries of the matrix-vector product as well as the inner-product vertex and inner-product edges. The colinear edges are added in Figure 4(c) corresponding to optimizations resulting from rows {4,7} being colinear and rows {5,6,8} also being colinear. In Figure 4(d) the edges for the partial colinear operations are added. Finally, Figure 4(e) shows an MST solution for this graph problem with the edges of the MST highlighted. A traversal of this MST generates the following operations to compute the matrix-vector product

$$\begin{aligned} y_3 &= 0.5x_1, \\ y_2 &= 0.5x_3, \\ y_1 &= (4/3)x_2, \\ y_8 &= -y_1 - (4/3)x_3, \\ y_7 &= -y_1 - (4/3)x_1, \\ y_9 &= -y_8 + (4/3)x_1, \\ y_6 &= 0.5y_8, \\ y_5 &= (-1/8)y_8, \\ y_4 &= (-1/8)y_7 \end{aligned}$$

at a cost of 9 MAPs.

2.7 Graph Model Results

We implemented the graph model with the inner-product vertices/edges and the binary row relationship edges described above (subsections 2.1-2.4) and used Prim's algorithm to find the MST for this graph. Then we traversed the MST starting from the inner-product vertex to obtain an optimal solution (for the row relationships used) and computed the cost of the solution (in MAPs) from the weights of the MST edges. The implementation was written in C++.

We used matrices from the local stiffness matrix formulation (described in subsections 1.2 and 1.3) obtained from FErari so that our results could be compared directly with previously reported results. These FErari matrices were obtained for the 2D and 3D Laplace equation (using triangles and tetrahedrons) and Lagrange polynomial basis functions of orders 1 to 6. Information about these matrices is shown in Table 1.

Table 2 gives a summary of the edges produced in the resulting graph for each matrix. Only the inner-product (IP), colinear (CL), and partial colinear (PCL) are applicable to the graph model. The coplanar hyperedges (CP) are given for the hypergraph model in the next section. As previously mentioned, we have not included the edges that cannot be in the optimal solution. We see that the partial colinear edges are the most plentiful. However, even with these edges, the edge count is fairly modest and does not pose a problem for finding the MST. In fact, building the graph is usually significantly more expensive than finding the MST for the graph.

Tables 3 and 4 show the results of the FErari graph algorithm and our improved graph algorithm that includes partial colinear row relationships. Each row in these tables reports results for a matrix resulting

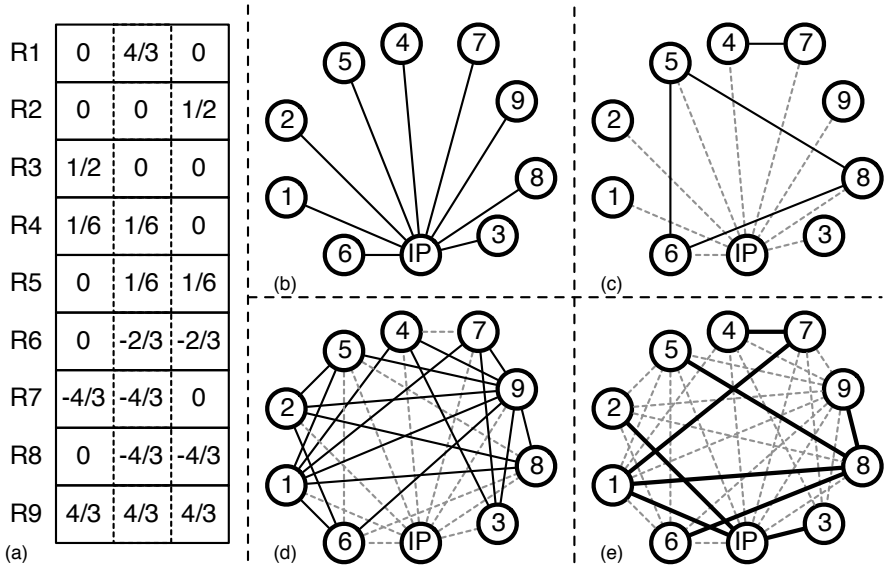

Figure 4: Graph example with (a) matrix, (b) row vertices and inner-product vertex/edges, (c) colinear edges, (d) partial colinear edges, and (e) MST solution. Edges weights are suppressed in figure for graphical clarity.

from a different order polynomial basis. The first column of these tables gives the order of the Lagrange polynomials. The number of MAPs required to compute the full matrix-vector product with and without unnecessary multiplications by zero matrix entries is shown in columns two and three, respectively. The cost of the FErari algorithm (best FErari graph method results in MAPs reported from [6, 8, 9]) for each matrix is given in the fourth column. We report the cost of our improved algorithm in the final column.

From these results (Tables 3 and 4), we see that both FErari's graph implementation and our improved graph implementation result in a substantial reduction in MAPs for the matrix-vector multiplication over the unoptimized algorithms. Our implementation reduces the number of MAPs by up to 60% for the 2D matrices (order 3) and up to 59% for the 3D matrices (order 4) over the unoptimized matrix-vector product without the multiplications by the zero matrix entries (column 3). We also see a significant improvement in the results from our graph implementation over the FErari results, especially for the higher order problems. For example, our implementation showed a 17% and 21% reduction in the number of MAPs over FErari for

Table 1: FErari 2D and 3D Laplace matrices. For each matrix, the number of rows (n), number of columns (d), number of elements (nd), and the number of nonzeros (nnz) are given.

	2D Laplace				3D Laplace			
Order	n	d	nd	nnz	n	d	nd	nnz
1	6	3	18	10	10	6	60	21
2	21	3	63	34	55	6	330	177
3	55	3	165	108	210	6	1260	789
4	120	3	360	292	630	6	3780	2586
5	231	3	693	589	1596	6	9576	7125
6	406	3	1218	1070	3570	6	21420	16749

Table 2: Number of hyperedges in resulting hypergraph for FErari matrices. Hyperedges are divided into four categories: inner-product edges (IP), colinear edges (CL), partial colinear edges (PCL), and coplanar hyperedges (CP).

	2D Laplace				3D Laplace			
Order	IP	CL	PCL	CP	IP	CL	PCL	CP
1	6	0	9	3	10	0	18	0
2	13	9	41	24	49	0	507	84
3	35	22	269	420	168	22	4802	732
4	97	12	1001	2184	480	75	31995	5380
5	213	48	3120	9213	1291	127	114494	23276
6	379	87	7112	25948	2949	283	419303	75985

Table 3: Graph algorithm: 2D Laplace FErari matrices, matrix-vector multiplication costs (in MAPs) for several different order Lagrange polynomials and algorithms. GPCR is our improved graph algorithm with partial colinear row relationship optimizations.

Order	Unoptimized MAPs	Unoptimized Nonzero MAPs	FErari MAPs	GPCR MAPs
1	18	10	**7**	**7**
2	63	34	15	**14**
3	165	108	45	**43**
4	360	292	176	**152**
5	693	589	443	**366**
6	1218	1070	867	**686**

the 2D order 5 and order 6 matrices, respectively.

3 Hypergraph Extension

A major limitation of the graph model described above is that more complex relationships relating more than two rows (and thus more than two resulting vector entries) cannot be expressed, since an edge contains only two vertices. A natural extension to the graph model that addresses this limitation is a hypergraph model with hyperedges. In this hypergraph model, each hyperedge can contain two or more vertices and represents operations relating two or more resulting vector entries [2]. In our hypergraph model implementation, we build edges (2-vertex hyperedges) for the same inner-product operations and operations resulting from binary row relationships described in section 2 and augment these edges with additional hyperedges of cardinality greater than two. For the present implementation, we limit our hypergraph model to additional hyperedges with cardinality three.

3.1 Coplanar Row Relationships

One row relationship that results in optimizations that we can express with the higher cardinality hyperedges is a generalization of the colinear row relationship in which three or more rows are linearly dependent. Limiting the model to three-vertex hyperedges, this relationship describes three rows in which the corresponding vectors are linearly dependent or coplanar, so that $\alpha_1 \mathbf{r}_1 + \alpha_2 \mathbf{r}_2 + \alpha_3 \mathbf{r}_3 = 0$ for some nonzero scalars α_1, α_2, and α_3. Thus, the resulting vector entry for a given row (e.g., y_1) can be written in terms of the resulting

Table 4: Graph algorithm: 3D Laplace FErari matrices, matrix-vector multiplication costs (in MAPs) for several different order Lagrange polynomials and algorithms. GPCR is our improved graph algorithm with partial colinear row relationship optimizations.

Order	Unoptimized MAPs	Unoptimized Nonzero MAPs	FErari MAPs	GPCR MAPs
1	60	21	–	**17**
2	330	177	101	**79**
3	1260	789	370	**342**
4	3780	2586	1118	**1049**
5	9576	7125	–	**3592**
6	21420	16749	–	**8835**

vectors for the other two rows in the following optimization

$$y_1 = \mathbf{r}_1 x = \beta_1 \mathbf{r}_2 x + \beta_2 \mathbf{r}_3 x = \beta_1 y_2 + \beta_2 y_3,$$

where $\beta_1 = -\alpha_2/\alpha_1$ and $\beta_2 = -\alpha_3/\alpha_1$. The cost of this relationship is 2 MAPs (1 MAP if $\beta_1 = \pm 1$ or $\beta_2 = \pm 1$). While this optimization requires more MAPs than the colinear optimization, it can yield a reduction in MAPs over the partial colinear optimizations or the full inner-product computation for a vector entry. Figure 5 shows a three-row example and the hyperedge representation of operations resulting from the three-row linear dependence for these rows in which the operations relate the resulting vector entries by $y_1 = 3y_2 + 3y_3$, $y_2 = \frac{y_1}{3} - y_3$, and $y_3 = \frac{y_1}{3} - y_2$.

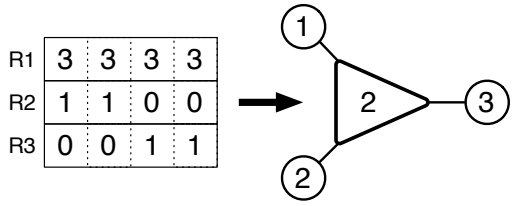

Figure 5: Hyperedge representation of operations resulting from three-row linear dependence. Connected triangle represents the hyperedge, with weight (2) placed inside triangle.

3.2 Searching for Coplanar Row Relationships

An important issue is how to find these coplanar row relationships in an efficient manner. The brute force method of looping over all row triples has complexity $O(\alpha n^3)$, where α is the cost of determining whether a vector triple is coplanar (e.g., α is $O(d^3)$ if SVD is used). We can do better by applying a necessary condition for coplanarity to all pairs of row vectors [9]. In particular, we project each row vector from $\mathbb{R}^d$ to $\mathbb{R}^3$ and calculate the normal to the plane formed by each pair of projected vectors. If two pair of vectors are coplanar, then the corresponding normal vectors must be colinear (for $d = 3$, this is also a sufficient condition). We can apply the colinear searching method described in subsubsection 2.2.1 to the vectors normal to the planes formed by the n^2 projected row vector pairs. In this manner, we can find sets of potentially coplanar row vectors in $O(dn^2 \log(n))$. For $d > 3$, we then use brute force to determine which of the potentially coplanar row vectors are in fact coplanar. This is effective since the number of potentially coplanar row vector triples is much smaller than the total number of row vector triples.

3.3 Hypergraph Model Minimization Problem

Defining a valid hypergraph solution that corresponds to a set of operations with a minimum number of MAPs is more complicated than for the graph formulation. The optimization problem is still equivalent to finding a set of hyperedges of minimum weight with certain properties. The resulting hyperedges also still form a connected subhypergraph spanning all vertices, since each entry in the resulting vector must be calculated either by an inner product or from a row relationship utilizing one or more previously calculated vector entries. With only these restrictions on the solution, finding a hypergraph MST (follows naturally from the generalization of a cycle [1]) would yield a solution with the minimum number of MAPs, as with the graph formulation. However, the MST would allow infeasible solutions with three-row linear dependence optimizations in which two of the resulting vector entries are calculated from one previously calculated resulting vector entry. Thus, we must impose the restriction that all but one of the resulting vector entries corresponding to the rows in the relationship are determined before the final vector entry is calculated using this linear dependency operation. This translates to the restriction on the hypergraph model that a hyperedge may be in the hypergraph solution only if there exist valid traversals in the solution subhypergraph (without this hyperedge) from the inner-product vertex to all but one of the vertices in the hyperedge. A valid traversal is a traversal of a series of hyperedges in which each hyperedge cannot be traversed until all but one of its vertices have been visited in the traversal. It is important to note that this constraint applies to the MST solution of the graph model as well, since each edge in the solution has two vertices and there must be a traversal (path not containing this edge) from the inner-product vertex to one of the two vertices. With this important restriction, it is clear that any feasible solution that contains a hyperedge of cardinality three or greater contains a cycle and thus is not a tree. We see an example of a hypergraph solution in Figure 6 that is not a tree (thus not an MST). The solution graph contains two different paths from the inner-product vertex to vertex 5 ({IP,1,3,5} and {IP,2,4,5}) and thus contains a cycle and is not a tree. The solution in which we are interested is a minimum spanning connected subhypergraph of the original hypergraph with the hyperedge restriction. As with the graph problem, the cost of the matrix-vector product resulting from the hypergraph solution is equivalent to the sum of the hyperedge weights in the resulting solution subhypergraph.

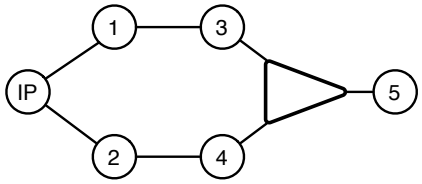

Figure 6: Example of hypergraph solution that is not a tree.

Although the solution to the hypergraph minimization problem is not an MST (if it has a hyperedge of cardinality greater than two), much research has been done on the complexity analysis of the hypergraph MST problem (MST-H) and the related hypergraph Steiner tree problem [12]. We summarize a couple of the MST-H complexity results since we believe our hypergraph problem to be of similar hardness. In particular, finding the MST of a weighted r-bounded (all hyperedges have cardinality of at most r) hypergraph for $r \geq 4$ is NP-hard [11]. The complexity for $r = 3$ is still an open problem. The best known polynomial time approximation algorithm for $r = 3$ yields a solutions within a factor of $3/2$ of the optimal solution [11, 10].

3.4 Modified Prim's Algorithm

The hypergraph model minimization problem is significantly more difficult than the graph optimization problem. We suspect that finding an optimal solution is NP-hard, and thus we consider heuristic methods. Our first attempt at solving this optimization problem was to implement a version of Prim's algorithm

modified for hypergraphs. We enforce the previously described hyperedge restriction in the algorithm by allowing hyperedges to be added to the solution only if all but one of the vertices for that hyperedge are covered by previously added hyperedges. Figure 7 shows an example of the modified Prim's algorithm on a small hypergraph with the MAP count shown inside the box at each stage of the algorithm. The three-vertex hyperedge {1, 2, 5} is not available to be added to the solution until two of the three vertices (1 and 2) are both present in the solution after stage (d). Again we see that the solution shown in Figure 7(f) is not a tree.

Unfortunately, this greedy algorithm does not necessarily yield an optimal solution for the row relationships utilized. Figure 8 shows a simple example where the modified Prim's algorithm yields a suboptimal result. For the hypergraph on the left, the optimal solution costs 8 MAPs but the modified Prim's algorithm solution costs 9 MAPs. However, the modified Prim's algorithm solution of the hypergraph problem is guaranteed to require no more (and often fewer) MAPs than the solution of the graph formulation of section 2 and can be found in polynomial time. Thus, this greedy solution of the hypergraph problem can still potentially yield useful results.

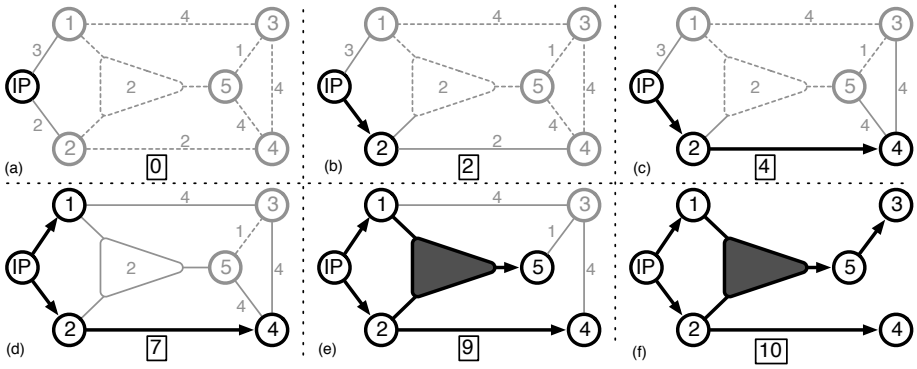

Figure 7: Example of hypergraph extension of Prim's algorithm. Non-solution inner-product edges suppressed for graphical clarity.

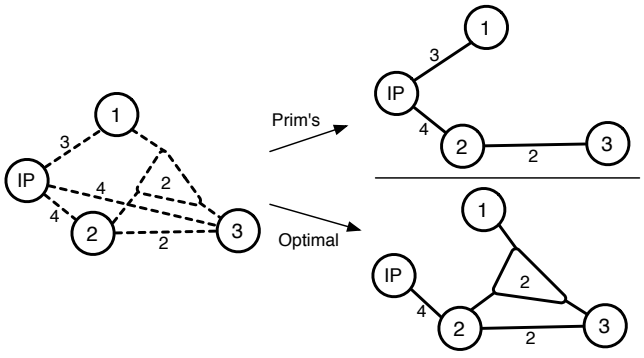

Figure 8: Suboptimal solution obtained by modified Prim's algorithm for hypergraphs. Optimal solution requires 8 MAPs and modified Prim's algorithm solution requires 9 MAPs.

3.5 Related Work

Kirby, et al. addressed these more complex relationships that relate more than two rows in a different manner [9], focusing on the geometric relationships between three or more row vectors. After removing the colinear and identical rows, they have a two-step approach for addressing coplanar row relationships. First, they find all row triples that form coplanar vectors and connect the triples that have two rows in common. The idea is that computing the resulting inner-products for two of the rows in this connected set of row triples allows replacement of the inner-product for each remaining row using a 2 MAP operation. This step is similar to our construction of the hypergraph. The second step is to find the cheapest set of rows to calculate full inner-products (referred to as a minimal generator) such that two rows for each connected set of row triples are in this set. This step is akin to our solution of the hypergraph problem. Kirby, et al. give a greedy algorithm for finding a minimal generator [9]. This process for coplanar relationships is generalized for relationships of higher numbers of rows. With this generalization, an iterative process can be used to find a series of minimal generators such that the minimal generator for i-row linear relationships is used as the input set of rows to be enumerated when searching for the $(i+1)$-row linear dependency relationships in the next iteration. This gives a very efficient algorithm for calculating the minimal cost matrix-vector product, using only dot product and linear dependency (including colinear) relationships. We report the results from this approach in the next subsection.

The drawback to this approach is that it is less natural to incorporate the non-linear dependency operations such as the Hamming distance optimization [6, 8] or our optimizations resulting from partial colinear row relationships. We saw in section 2 that these partial colinear row relationships are very important in minimizing the MAPs. Kirby, et al. outline a combinatorial structure that would incorporate the inner-product, Hamming distance, and linear-dependency operations and a heuristic modification to Prim's algorithm to solve this problem [9]. However, they they conclude that their implementation of this is impractical for all but the simplest problems and do not report results from this approach.

3.6 Hypergraph Model Results

We implemented the hypergraph model of the optimization problem with the inner-product vertices/edges, the binary row relationship edges (subsections 2.2-2.4), and the three-vertex linear dependence hyperedges (subsection 3.1) and used the modified hypergraph version of Prim's algorithm to find the minimum spanning connected subhypergraph solution. Our code traverses the subhypergraph solution starting from the inner-product vertex to determine a reduced set of operations to calculate the matrix-vector product and determines the cost of the solution (in MAPs) from the weights of the solution hyperedges. As with the graph implementation, the hypergraph implementation was written in C++. We used the GNU Scientific Library (GSL) singular value decomposition routine (gsl_linalg_SV_decomp) to verify that three row vectors were coplanar [3].

We use the same matrices for the hypergraph results as for the previous graph results, shown in Table 1. Table 2 gives a summary of the edges produced in the resulting hypergraph. The inner-product (IP), colinear (CL), and partial colinear (PCL) are the same as in the graph model. The coplanar hyperedges (CP) are also shown for the resulting hypergraphs. Again, the hyperedge count is fairly modest and building the hypergraph is significantly more expensive than finding the solution to the hypergraph minimization problem.

Table 5 shows a comparison between our graph model implementation and our hypergraph model for the 2D Laplace equation matrices. For these 2D matrices, the hypergraph model results show modest improvement over the graph model results (at most 14%, which was obtained for the fourth order polynomial basis). However, it is important to note that these 2D matrices have only three columns. Thus, we gain little by using the additional three-vertex linear dependency hyperedges in the hypergraph since it will save at most 1 MAP over a full inner product (2 MAPs if one of the coefficients in the operation happens to be ± 1). Another possible explanation for the modest improvement is that the modified Prim's hypergraph implementation may yield significantly suboptimal results, and if the problem were solved optimally, the hypergraph implementation would provide a further reduction in MAPs. We believe the former is the more

likely explanation, however, and that the graph model solution is nearly optimal for these 2D matrices, with little further reduction possible.

Table 5: Hypergraph algorithm: 2D Laplace FErari matrices, matrix-vector multiplication costs (in MAPs) for several different order Lagrange polynomials and algorithms. GPCR is our improved graph model with partial colinear row relationships. HGraph is our hypergraph extension to the improved graph model.

Order	Unoptimized MAPs	Unoptimized Nonzero MAPs	GPCR MAPs	HGraph MAPs
1	18	10	7	**6**
2	63	34	**14**	**14**
3	165	108	43	**38**
4	360	292	152	**131**
5	693	589	366	**352**
6	1218	1070	686	**677**

Table 6 shows a comparison between our graph model implementation and our hypergraph model implementation for the 3D Laplace equation matrices (columns 5 and 6). The table also shows the FErari geometric results (for polynomial basis orders 2-4), the approach outlined in the previous subsubsection [9]. For order 3 and 4 polynomial bases, the FErari geometric results are an improvement over the FErari graph results, so we have included the geometric rather than the graph FErari results. For these 3D matrices, the code generated by our hypergraph model showed a significant reduction in MAPs in comparison with the code generated by our graph model, especially for the fourth order polynomial basis in which we saw an additional reduction in MAPs by 31% for a total reduction of 72% over the unoptimized matrix-vector product code. We speculate that solving this problem optimally for the higher order polynomial bases would further reduce the MAPs, although the reduction might be modest. Similarly, our hypergraph implementation yielded less costly results than the FErari geometric implementation (e.g., 31% for the fourth order polynomial basis). Much of this may be attributed to our improved graph model implementation and that we take advantage of the partial colinear relationships that the FErari geometric implementation neglects.

3.6.1 Runtimes for Implementation

In this subsubsection, we present the runtimes for our implementation for the 3D Laplace matrices. In particular, Table 7 shows the runtimes for building the graph/hypergraph and solving the resulting minimization problem for both our graph and hypergraph implementations. For the most part, we see that the time required to build the graph/hypergraph is more significant than the time to solve the resulting optimization problem. For the larger problems, the runtime for building the graph is dominated by the $O(dn^2 \log(d))$ complexity for detecting the partial colinear row relationships. The coplanar row detection dominates the time needed to build the hypergraphs. Admittedly, the runtimes for the largest problems were sizable. Clearly one would not want to use the optimizations we implemented for one matrix-vector multiplication, but in the context of a "finite-element compiler optimization" where the optimized code could be used billions of times, we feel the runtimes are reasonable and justifiable. These runtimes were obtained on a Mac with a 2.16 GHz Intel Core 2 Duo processor, using the gcc compiler.

4 Accuracy

We have shown that our implementations generate operations for computing matrix-vector products with a significant reduction in MAPs in comparison to the unoptimized matrix-vector multiplication algorithms for the Laplace finite element FErari matrices. However, the relationships between resulting vector entries are determined numerically (based on a tolerance) and are not exact. Thus, we are naturally concerned with

Table 6: Hypergraph algorithm: 3D Laplace FErari matrices, matrix-vector multiplication costs (in MAPs) for several different order Lagrange polynomials and algorithms. FErari Geom are results from the new method in FErari geometric optimization paper [9] that utilizes row relationships of more than two rows. GPCR is our improved graph model with partial colinear row relationships. HGraph is our hypergraph extension to the improved graph model.

Order	Unoptimized MAPs	Unoptimized Nonzero MAPs	FErari Geom MAPs	GPCR MAPs	HGraph MAPs
1	60	21	–	**17**	**17**
2	330	177	105	79	**65**
3	1260	789	327	342	**262**
4	3780	2586	1045	1049	**726**
5	9576	7125	–	3592	**3098**
6	21420	16749	–	8835	**8199**

Table 7: Time (in seconds) for building the graph/hypergraph and solving the resulting optimization problem. Runtimes for 3D Laplace FErari matrices of several different polynomial orders.

Order	Graph		Hypergraph	
	Build	Solve	Build	Solve
1	6.97e-4	3.36e-4	5.14e-3	3.22e-4
2	4.71e-3	1.29e-3	1.52e-2	1.57e-3
3	3.78e-2	9.62e-3	1.26e-1	1.11e-2
4	2.79e-1	1.08e-1	1.39e+0	1.29e-1
5	1.71e+0	1.19e+0	1.76e+1	1.58e+0
6	9.08e+0	1.17e+1	1.51e+2	1.47e+1

Table 8: 2D and 3D Laplace FErari matrices, matrix-vector multiplication accuracy for several different polynomial order basis functions. GPCR is our improved graph model with partial colinear row relationships. HGraph is our hypergraph extension to the improved graph model.

Order	2D Laplace		3D Laplace	
	GPCR	HGraph	GPCR	HGraph
1	0	0	0	0
2	3.1123e-16	3.1294e-16	1.2125e-16	1.4664e-16
3	4.1471e-16	2.8602e-16	7.5860e-16	1.7935e-15
4	1.9063e-15	1.0295e-15	2.6234e-16	7.1006e-16
5	4.0136e-16	6.3698e-16	1.6623e-15	1.4481e-15
6	2.2813e-16	2.4842e-15	6.8123e-15	3.4148e-15

what price is paid in terms of accuracy for the reduction in MAPs. Table 8 shows the accuracy for the matrix-vector multiplication code for the 2D and 3D Laplace FErari matrices, using the operations generated by our graph and hypergraph implementations. We measured the relative error for each matrix-vector product by dividing the norm of the difference between the resulting vectors for the full matrix-vector multiplication algorithm and our generated optimized matrix-vector multiplication operations (for both the graph and hypergraph models) by the norm of the resulting vector for the full matrix-vector multiplication algorithm, $\|\hat{\mathbf{y}} - \mathbf{A}\mathbf{x}\|_2 / \|\mathbf{A}\mathbf{x}\|_2$ (where $\hat{\mathbf{y}}$ is the result vector calculated by our optimized code). The relative accuracy of the resulting vector is generally satisfactory for the optimized instructions generated from both the graph and hypergraph algorithms. This accuracy is about best we can expect for double precision arithmetic.

5 Conclusions

We have shown that our graph implementation greatly reduces the number of MAPs for the matrices obtained by FErari for the 2D and 3D Laplace equation finite element discretizations with several orders of basis functions. Our implementation produced significantly improved results over the FErari graph implementation, demonstrating the importance of utilizing partial colinear row relationships in generating optimal code to perform the matrix-vector multiplication. Although we limited our model to three-vertex linear dependence hyperedges and solved the resulting hypergraph problem in a greedy fashion, the hypergraph extension to the graph model showed additional improvement in the reduction of MAPs, in particular for the 3D matrices, which had more columns than the 2D matrices. Most likely, we would see a further reduction in MAPs for the 3D matrices with the implementation of higher cardinality hyperedges (e.g., those representing 4 row linear dependence operations) and additional hyperedge relationships (e.g., partial coplanar row relationships).

Although we presented results only for the FErari matrices used in the construction of the finite element local stiffness matrices, our implementation is general and can attempt to generate a reduced operation matrix-vector multiplication set of instructions for any matrix. Realistically, however, the effectiveness of the optimization will depend on the particular application and properties of the matrices involved. In order to make this optimization useful, the matrices must have row relationships that can be efficiently exploited.

It is important to note that the setup and solution of the optimization problem requires much more time than the actual matrix-vector multiplication algorithm. Thus, the optimization of matrix-vector multiplication is useful only when a matrix is multiplied many times by different vectors. For the matrices presented that are used in the evaluation of local stiffness matrices, the matrix is multiplied by a vector for every element in the finite element discretization (assuming all elements use the same order basis functions), and thus the generated code can be reused for every finite element. Furthermore, the generated matrix-vector multiplication code for a particular equation and particular order of basis functions can be stored and reused for all problems using the same equation and basis functions. Thus, paying the relatively

high cost of optimization is reasonable for these matrices used in finite element matrix evaluation, since the optimized matrix-vector product operations are reused repeatedly. This is analogous to spending time optimizing numerical kernels in libraries that are to be used many times by library users. For example, autotuning projects such as ATLAS [14] and OSKI [13] optimize simple numerical kernels to run efficiently on particular systems. This optimization procedure will be more costly than the numerical kernels but the numerical kernels will be reused repeatedly.

As for the types of matrices for which this combinatorial optimization problem will be useful in general, this approach is most useful for matrices with many row relationships between a small set of rows and with a moderate total number of rows. If a matrix has only very complex row relationships relating several rows, the time to generate the high cardinality hyperedges will become too expensive and the potential savings will be moderate at best. Similarly, if a matrix has too many rows, the cost to find these relationships may also become prohibitive.

6 Future Work

We believe that our current hypergraph model generates nearly optimal instructions for the 2D Laplace matrices used in this paper. However, we believe that significant further improvement can be made for the 3D Laplace matrices and other matrices in general. In this paper, we focused on hyperedges with cardinality of two or three. More recently, we have begun implementing higher cardinality hyperedges with 4, 5, and 6 vertices for the linear dependence row relationships. There are many challenges for generating these higher cardinality linear dependency hyperedges, including efficiently detecting the linear dependencies of the rows, generating the resulting hypereges, and pruning the suboptimal hyperedges. So far we have found these higher cardinality hyperedges too computationally intensive to utilize in our hypergraph implementation for the larger test problems. More efficient detection algorithms and more aggressive pruning techniques are clearly needed. Similar to the partial colinear binary row relationships, we plan to implement more complex hyperedge relationships that may prove useful in finding an optimal solution. It may also become important to solve the hypergraph problem optimally. Most likely this problem is NP-hard, but we still hope to solve the problem in a near optimal and efficient manner.

Appendix

In this appendix, we derive the local stiffness matrix for the 2D Laplace equation. We first start with the bilinear form $(\nabla u, \nabla v)_e = \det(\mathbf{J})(\nabla u, \nabla v)_{\hat{e}}$. It follows that

$$\begin{aligned}
(\nabla u, \nabla v)_e &= \det(\mathbf{J})(\nabla u, \nabla v)_{\hat{e}} \\
&= \det(\mathbf{J}) \left(\sum_{l,m,n=1}^{2} \left(\frac{\partial u}{\partial \hat{\xi}_m} \frac{\partial \hat{\xi}_m}{\partial \xi_l}, \frac{\partial v}{\partial \hat{\xi}_n} \frac{\partial \hat{\xi}_n}{\partial \xi_l} \right)_{\hat{e}} \right) \\
&= \det(\mathbf{J}) \left(\sum_{l,m,n=1}^{2} \frac{\partial \hat{\xi}_m}{\partial \xi_l} \left(\frac{\partial u}{\partial \hat{\xi}_m}, \frac{\partial v}{\partial \hat{\xi}_n} \right)_{\hat{e}} \frac{\partial \hat{\xi}_n}{\partial \xi_l} \right) \\
&= \det(\mathbf{J}) \left(\sum_{l,m,n=1}^{2} \frac{\partial \hat{\xi}_m}{\partial \xi_l} \left(\mathbf{u^T D_{mn} v} \right) \frac{\partial \hat{\xi}_n}{\partial \xi_l} \right),
\end{aligned}$$

where $\mathbf{D_{mn}}(i,j) = \left(\frac{\partial \phi_i}{\partial \hat{\xi}_m}, \frac{\partial \phi_j}{\partial \hat{\xi}_n} \right)_{\hat{e}}$. Factoring the vectors $\mathbf{u}$ and $\mathbf{v}$ out, we get $(\nabla u, \nabla v)_e = \mathbf{u^T S}^e \mathbf{v}$, where the local stiffness matrix is

$$S^e = \det(\mathbf{J}) \left(\sum_{l,m,n=1}^{2} \frac{\partial \hat{\xi}_m}{\partial \xi_l} \mathbf{D_{mn}} \frac{\partial \hat{\xi}_n}{\partial \xi_l} \right).$$

However, in this form of the stiffness matrix, the terms dependent on the individual elements are mixed with the terms only dependent on the reference elements. Rearranging the terms, we can achieve more separation

$$S^e = \sum_{m,n=1}^{2} \mathbf{D_{mn}} \left[\det(\mathbf{J}) \left(\frac{\partial \hat{\xi}_m}{\partial \xi_1} \frac{\partial \hat{\xi}_n}{\partial \xi_1} + \frac{\partial \hat{\xi}_m}{\partial \xi_2} \frac{\partial \hat{\xi}_n}{\partial \xi_2} \right) \right].$$

Terms inside the bracket are dependent on the individual elements, while the matrices outside the bracket only depend only on the reference element. The entries in this stiffness matrix can be written as the Frobenius product of a tensor and a matrix

$$\mathbf{S}^e_{i,j} = \sum_{m}^{2} \sum_{n}^{2} \mathbf{G}^e_{m,n} \mathbf{K}_{i,j,m,n} = \mathbf{K}_{i,j} : \mathbf{G}^e,$$

where

$$\mathbf{G^e} = \begin{bmatrix} \det(\mathbf{J}) \left(\frac{\partial \hat{\xi}_1}{\partial \xi_1} \frac{\partial \hat{\xi}_1}{\partial \xi_1} + \frac{\partial \hat{\xi}_1}{\partial \xi_2} \frac{\partial \hat{\xi}_1}{\partial \xi_2} \right) & \det(\mathbf{J}) \left(\frac{\partial \hat{\xi}_1}{\partial \xi_1} \frac{\partial \hat{\xi}_2}{\partial \xi_1} + \frac{\partial \hat{\xi}_1}{\partial \xi_2} \frac{\partial \hat{\xi}_2}{\partial \xi_2} \right) \\ \det(\mathbf{J}) \left(\frac{\partial \hat{\xi}_2}{\partial \xi_1} \frac{\partial \hat{\xi}_1}{\partial \xi_1} + \frac{\partial \hat{\xi}_2}{\partial \xi_2} \frac{\partial \hat{\xi}_1}{\partial \xi_2} \right) & \det(\mathbf{J}) \left(\frac{\partial \hat{\xi}_2}{\partial \xi_1} \frac{\partial \hat{\xi}_2}{\partial \xi_1} + \frac{\partial \hat{\xi}_2}{\partial \xi_2} \frac{\partial \hat{\xi}_2}{\partial \xi_2} \right) \end{bmatrix}$$

is a matrix dependent on the individual element and K is a tensor only dependent on the reference element such that

$$\mathbf{K}_{i,j} = \begin{bmatrix} \mathbf{D}_{11}(i,j) & \mathbf{D}_{12}(i,j) \\ \mathbf{D}_{21}(i,j) & \mathbf{D}_{22}(i,j) \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial \phi_i}{\partial \hat{\xi}_1}, \frac{\partial \phi_j}{\partial \hat{\xi}_1} \right)_{\hat{e}} & \left(\frac{\partial \phi_i}{\partial \hat{\xi}_1}, \frac{\partial \phi_j}{\partial \hat{\xi}_2} \right)_{\hat{e}} \\ \left(\frac{\partial \phi_i}{\partial \hat{\xi}_2}, \frac{\partial \phi_j}{\partial \hat{\xi}_1} \right)_{\hat{e}} & \left(\frac{\partial \phi_i}{\partial \hat{\xi}_2}, \frac{\partial \phi_j}{\partial \hat{\xi}_2} \right)_{\hat{e}} \end{bmatrix}.$$

Acknowledgements

We thank Robert C. Kirby for assistance with FErari and for discussions about the problem in general. We also thank Erik G. Boman and Bruce A. Hendrickson for bringing the problem to our attention, suggesting we look at a hypergraph model, and for many related discussions. This research was made possible by DOE CSGF fellowship support, DE-FG02-97ER25308.

References

[1] C. Berge. *Graphs and Hypergraphs*, volume 6 of *North-Holland Mathematical Library*. Elsevier Science Publishing Company, 1973.

[2] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. Elsevier Science Publishing Company, 1989.

[3] M. Galassi, et al. *GNU Scientific Library Reference Manual*. 2nd edition, 2007.

[4] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics (Numerical Mathematics and Scientific Computation)*. Oxford University Press, USA, August 2005.

[5] R. C. Kirby. Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, December 2004.

[6] R. C. Kirby, M. G. Knepley, and L. R. Scott. Optimizing the evaluation of finite element matrices. *SIAM Journal on Scientific Computing*, 27(3):741–758, 2005.

[7] R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Trans. Math. Softw.*, 32(3):417–444, 2006.

[8] R. C. Kirby, A. Logg, L. R. Scott, and A. R. Terrel. Topological optimization of the evaluation of finite element matrices. *SIAM Journal on Scientific Computing*, 28(1):224–240, 2006.

[9] R. C. Kirby and L. R. Scott. Geometric optimization of the evaluation of finite element matrices. *SIAM Journal on Scientific Computing*, 29(2):827–841, 2007.

[10] H. J. Prömel and A. Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36:89–101, 2000.

[11] H. J. Prömel and A. Steger. *The Steiner Tree Problem: A Tour Through Graphs, Algorithms, and Complexity.* Vieweg+Teubner Verlag, 2002.

[12] V. V. Vazirani. Recent results on approximating the steiner tree problem and its generalizations. *Theoretical Computer Science*, 235:205–216, 2000.

[13] R. Vuduc, J. W. Demmel, and K. A. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16:521–530, 2005.

[14] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–25, 2001.