Noname manuscript No.
(will be inserted by the editor)

Improved Cache Utilization and Preconditioner Efficiency Through Use of a Space-Filling Curve Mesh Element- and Vertex-Reordering Technique

Shankar P. Sastry · Emre Kultursay · Suzanne M. Shontz · Mahmut T. Kandemir

the date of receipt and acceptance should be inserted later

Abstract Solving partial differential equations using finite element (FE) methods for unstructured meshes that contain billions of elements is computationally a very challenging task. While parallel implementations can deliver a solution in a reasonable amount of time, they suffer from low cache utilization due to unstructured data access patterns. In this work, we reorder the way the mesh vertices and elements are stored in memory using Hilbert space-filling curves in order to improve cache utilization in FE methods for unstructured meshes. This reordering technique enumerates the mesh elements such that parallel threads access shared vertices at different time intervals, reducing the time wasted waiting to acquire locks guarding atomic regions. Further, when the linear system resulting from the FE analysis is solved using the preconditioned conjugate gradient method, the performance of the block-Jacobi preconditioner also improves, as more nonzeros are present near the stiffness matrix diagonal. Our results show that our reordering reduces the L1 and L2 cache miss-rates in stiffness matrix assembly step by about 50% and 10%, respectively, on a single-core processor. We also reduce the number of iterations required to solve the linear system about 5%. Overall, our reordering reduces the time to assemble the stiffness matrix and to solve the linear system on a on a 4-socket, 48-core multi-processor by about 20%.

Shankar P. Sastry
Scientific Computing and Imaging Institute,
The University of Utah,
Salt Lake City, UT, 84112, U.S.A.
E-mail: sastry@sci.utah.edu

Emre Kultursay
Department of Computer Science and Engineering,
The Pennsylvania State University,
University Park, PA 16802, U.S.A.
E-mail: euk139@cse.psu.edu

Suzanne M. Shontz
Department of Mathematics and Statistics,
Department of Computer Science and Engineering,
Center for Computational Sciences,
Graduate Program in Computational Engineering,
Mississippi State University,
Mississippi State, MS 39762, U.S.A.
E-mail: sshontz@math.msstate.edu

Mahmut T. Kandemir
Department of Computer Science and Engineering,
The Pennsylvania State University,
University Park, PA 16802, U.S.A.
E-mail: kandemir@cse.psu.edu

1 Introduction

Meshes play a vital role in the numerical solution of partial differential equations (PDEs) on a given geometric domain. Accuracy of the solution depends on parameters such as the shape and size of the mesh elements [1]. With the advent of multicore processors, larger meshes can be used to solve the PDEs within a prescribed amount of time by solving the resulting large system of linear equations in parallel. In this paper, we study a mesh vertex- and element-reordering technique to exploit the memory architecture in shared-memory, multicore processors. We propose using a Hilbert space-filling curve (SFC) reordering technique [2] to order the mesh elements and vertices to improve the temporal and spatial locality in the data access patterns. The reordering also improves the efficiency of the block-Jacobi preconditioners used to solve the system of linear equations arising from a finite-element (FE) formulation to solve the PDE associated with mesh warping [3].

A mesh may be used several times to solve a PDE with different boundary conditions, and its elements and vertices may be accessed many times for mesh quality improvement. In some applications, a mesh may need to be warped several times [4–6]. In such cases, reordering the elements and vertices of the mesh can bring overall performance improvement, as the overhead of reordering is typically less. In applications involving large deformations, several time steps may be necessary to obtain an accurate solution to the corresponding PDEs [7]. In such applications, involving multiple passes through mesh vertices and elements, reordering techniques can help in improving the cache utilization. In this paper, we demonstrate the improvement in performance that is observed for one such application, namely the finite element-based mesh warping (FEMWARP) technique [3].

In order to numerically solve a PDE, a stiffness matrix is constructed from the discretize form of the PDE and the mesh. The boundary conditions are then applied, and the resulting linear system is solved. In this paper, we focus on parallel algorithms for solving a PDE using the FE method.

Parallel techniques have been developed in order to assemble the stiffness matrices associated with PDEs [8–11]. To improve the cache utilization for solving linear systems, vertex reordering techniques have also been developed [12, 13]. For improving cache performance for both constructing the stiffness matrix and solving the linear equations, Zhou *et al.* [14] developed a breadth first search (BFS)-based vertex- *and* element-reordering technique. Run-time techniques have also been developed [15, 16] to improve cache utilization in unstructured data access. These techniques are briefly described in Section 2.

Oliker *et al.* [17, 18] studied the reordering techniques mentioned above for two-dimensional meshes and found that a geometry-aware reordering technique, namely the self-avoiding walk (SAW) [13] technique, performs better than graph partitioning and BFS-based techniques. As the SAW technique has not been developed for three-dimensional geometry, we use a geometric, SFC reordering technique to reorder our mesh vertices and elements.

In order to assess the performance improvement due to the reordering technique, we warp our meshes using a parallel implementation of FEMWARP [3] on shared-memory processors. FEMWARP solves Laplace's equations to warp the mesh. The linear equations arising from the discretized Laplace's equations using our meshes are solved using a parallel implementation of the block-Jacobi preconditioned conjugate gradient (PCG) method. The SFC reordering algorithm, the FEMWARP technique, and the PCG method are described in detail in Section 3, and their implementations are described in Section 4.

We carry out numerical experiments for three different types of geometric domains using meshes that each have more than 10 million elements. In our experiments, we use a 48-core shared memory multi-processor. In this system, each core is assigned around 20,000 degrees of freedom to compute, which is sufficient to show the effectiveness of our method. We also perform simulations to demonstrate hit rate improvement in the L1 and L2 caches. The experimental setup and our results are described in Section 5. The reordering shows about 20% improvement in the run time of FEMWARP implementation on a 4-socket, 48-core multiprocessor. We strongly believe that our method can improve the performance of many real-world applications, such as the one described in [7] which assigns nearly 40,000 degrees of freedom to each of the 960 target processors.

Our conclusions and future work directions are given in Section 6.

2 Related Work

Several parallel algorithms have been proposed to assemble the global stiffness matrix associated with finite element methods. Since the global stiffness matrix is shared by many processors, graph-partitioning techniques have been used [8, 9] to partition mesh elements among all the threads. The corresponding rows of the global stiffness matrix are assembled in parallel by the threads. Synchronization constructs (i.e., semaphores and monitors) are used to obtain the correct global stiffness matrix when race conditions arise. An alternate method is to store the shared data of the global stiffness matrix in a distributed manner and to update the corresponding rows separately after those rows are assigned to each of the threads. The latter method avoids synchronization constructs. Rezende [10] proposed a different technique in which the mesh vertices, instead of the mesh elements, were assigned to each thread. This enables the corresponding rows to be assigned to each thread, and since each row is updated by a single thread, no synchronization steps are necessary. Note that a separate data structure must be maintained to store all the elements associated with every vertex.

Chien and Sun [11] proposed an element reordering algorithm in which mesh elements are renumbered and assigned to processors such that the last element of the previous thread and the first element of the next thread have common vertices. This ensures that the threads access the shared data at different times, and, thus, synchronization is theoretically not necessary. However, in practice, when many threads are operating on a mesh, synchronization constructs are necessary to obtain an accurate global stiffness matrix. Cuthill and McKee [12] developed a BFS-based vertex-renumbering algorithm that reduces the *bandwidth* of the adjacency matrix of a graph. In their algorithm, a BFS determines the order of vertices, but this ordering also depends on the degree of each of the vertices. The newly-discovered vertices at each level of the BFS are listed in increasing order of their degrees. Heber *et al.* developed a self-avoiding walk (SAW) technique [13] to reorder mesh elements and vertices based on the element and edge connectivity of a mesh. These algorithms have been used for accelerating sparse matrix-vector multiplications and other sparse matrix computations [17, 18]. Oliker *et al.* [17] observed that the SAW reordering technique performs better than the Cuthill and McKee's reordering technique [12] for sparse matrix computations. Zhou *et al.* developed a reordering technique [14] that reorders both mesh elements and vertices. This technique exploits the cache memory hierarchy while assembling the global stiffness matrix and also while solving the linear system that results from the FE formulation. Han *et al.* developed an algorithm [15] that constructs a hypergraph from the data-access pattern and applies hierarchical clustering. Strout and Hovland [16] developed a data- and iteration-reordering technique that uses spatial- and temporal-locality hypergraphs to improve the cache performance. These algorithms can be very expensive, as the underlying graph structures are computed at run time. There have been no studies on the effectiveness of the algorithms when they are applied before the execution of the program.

Mesh vertex reordering techniques have also been used in the context of mesh quality improvement [19, 20]. The objective of the vertex reordering technique is to reduce the total time required to improve the mesh quality. Shontz et al. [19] considered both a priori and dynamic reordering, whereas Park et al. [20] considered a priori vertex reordering

We use a Hilbert space-filling curve (SFC) element- and vertex-reordering technique to ensure that both temporal and spatial locality can be exploited while mesh elements and vertices are being accessed to compute the global stiffness matrix. An SFC naturally partitions a mesh geometrically and orders the elements such that the threads access elements with shared vertices at the same time very rarely. Therefore, synchronization does not significantly reduce the performance of the algorithm when scaled to a large number of processors.

SFCs have been used previously before in the context of structured data access [21] and unstructured meshes. Cache-oblivious layouts have been proposed [22] for volume rendering, surface mesh simplification, and isovalue extraction based on the Morton SFC. SFCs have also been used to perform parallel adaptive mesh generations [23] and parallel anisotropic mesh adaptation [24]. SFCs have also been proposed for sparse matrix-vector multiplications [25]. Mellor-Crummey *et al.* [26] also studied the effect of data reordering using SFCs on the data locality in multiple levels of cache memory hierarchy.

With the exception of [14], prior work has focused just on cache utilization for solving linear equations, i.e., the matrix is given as an input to the algorithms using a suitable data structure, but the construction of the matrix or the data structure has not been taken into account. Gerhold *et al.* [27] proposed a parallel mesh warping method that uses a spring analogy to move vertices to their new locations, and Tsai *et al.* [28] have proposed a multiblock spring- and transfinite interpolation-based parallel mesh warping technique, but they have not studied the cache utilization of their techniques. In this paper, we study the effect of the Hilbert SFC reordering technique on *both* the stiffness matrix assembly and the linear solution process for three-dimensional domains. This paper also presents the effect of the reordering on the efficiency of block Jacobi preconditioner.

3 Background

In this section, we provide background on some of the fundamental concepts and methods we refer to throughout the rest of this paper.

3.1 Data Structure to Store Meshes

Mesh vertices and elements are stored in two separate arrays. The vertices are indexed, and their $x-$, $y-$, and $z-$coordinates are stored in an array. The indices of four vertices in each of the tetrahedral elements are stored in another array. As a vertex can be shared by many elements, its index may be found in many entries of the array.

3.2 Space-Filling Curves

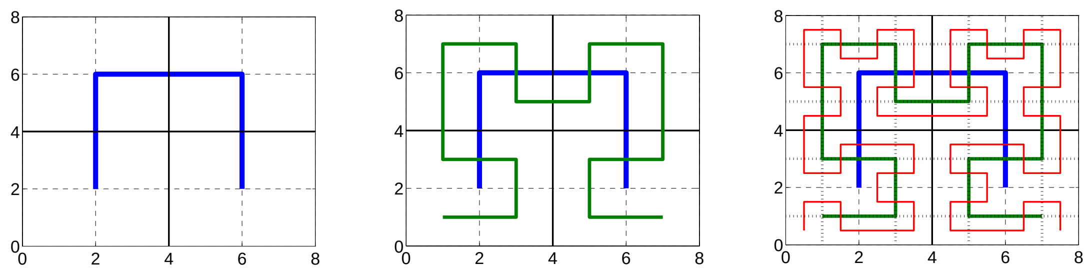
Space-filling curves are mappings from a one-dimensional interval to a region in an n-dimensional space. Space-filling curves pass though every point in the region in the n-dimensional space [2]. Since a space-filling curve does not intersect itself, an ordering of the points in the n-dimensional space can be obtained through its use. Fig. 1 shows the evolution of a space-filling curve for a two-dimensional region. Such curves are usually constructed through recursion, but infinite recursive calls are necessary to construct the ideal curve (that contains every point in the domain) to find the ordering of all the points in an n-dimensional space. Since only a finite number of elements are present in a mesh, it is possible to determine an ordering of the mesh elements through a finite number of steps. As space-filling curves order the vertices based on their spatial proximity, it is possible to extract spatial locality in their access pattern by ordering mesh elements and vertices appropriately. Since adjacent elements share one or more vertices, traversing mesh elements in the SFC order can also result in a higher temporal locality.

3.3 Reverse Cuthill-McKee Ordering

Cuthill and McKee [12] developed a reordering algorithm in order to reduce the bandwidth of sparse symmetric matrices. Their algorithm constructs a graph from a symmetric matrix, and BFS is used to determine the ordering of the matrix rows and columns. In each level of the BFS, graph nodes are arranged in increasing order of their degrees before proceeding to the next level. The reverse Cuthill-McKee (RCM) ordering [29] reverses the ordering obtained from Cuthill and McKee's BFS.

3.4 FEMWARP

FEMWARP is a finite element-based mesh warping algorithm [3], which solves Laplace's equations, $\nabla^2 u = 0$, (i.e., there is one equation in each of the three dimensions), in order to warp a mesh from an initial domain to a target domain. The algorithm takes the initial mesh and the boundary deformation (target vertex positions) as the input, and deforms



(a) first iteration (b) second iteration (c) third iteration

Fig. 1 Evolution of a space-filling curve (SFC) for a two-dimensional, square domain.

the mesh in three steps. First, the stiffness matrix obtained from the source mesh is used to compute the weights for each interior vertex such that the weighted sum of the positions of all of its neighboring vertices gives the position of the interior vertex. Second, the boundary elements are then transformed using the target vertex positions. Third, the new positions of the interior mesh vertices are computed by solving the system of linear equations obtained from the stiffness matrix and target vertex positions. The solution of the system of equations corresponds to the solution of Laplace's equations with Dirichlet boundary conditions given by the target vertex positions of the boundary vertices. A mesh may need to be deformed multiples times during the course of a simulation. Each time the mesh is deformed, the stiffness matrix remains the same, but the right-hand side, b, of the system of linear equations, $Ax = b$, is different. For a 3-dimensional mesh, FEMWARP solves Laplace's equations for each of the three dimensions in an uncoupled manner using the finite element method. The portion of the stiffness matrix that describes the connections of interior vertices to their interior neighbors is symmetric and positive definite. This submatrix and the boundary conditions form a set of linear equations, which can be solved using the preconditioned conjugate gradient (PCG) method described below.

3.5 Stiffness Matrix Assembly

The FE stiffness matrix contains the same number of rows and columns as the number of vertices in the corresponding FE mesh. Each row and column in the matrix corresponds to a vertex in the mesh. If vertex i and vertex j are connected by an edge in a tetrahedral element, the tetrahedral element contributes to the value of the stiffness matrix element in row i and column j of the matrix. The value depends upon the geometry of the element as well as the PDE being solved. The tetrahedral element also contributes to the value of the diagonal elements (in row i and column i, for instance) in the matrix. The contributions from all the elements are added to assemble the stiffness matrix. More information on FE techniques to solve Laplace's equations can be found in [30].

3.6 Preconditioned Conjugate Gradient Method

In order to solve a set of linear equations denoted by $Ax = b$, where A is a symmetric positive definite matrix and b is a vector, the CG method may be used. The CG method theoretically converges in n iterations, where n is the number of interior vertices in the mesh, and A is $n \times n$. However, if A is sparse, the CG method needs fewer iterations to converges. If A is ill-conditioned, a preconditioner P may also be used to improve the conditioning and to solve the equivalent system:

$$P^{-1}Ax = P^{-1}b, \tag{1}$$

which has a lower condition number if P is well chosen. The preconditioned conjugate gradient (PCG) algorithm is shown in Algorithm 1.

Algorithm 1 Preconditioned conjugate gradient (PCG) algorithm for solving a system of linear equations, $Ax = b$, using a preconditioner P.

```
$r_0 = b - Ax_0$
Solve $Pz_0 = r_0$
$d_0 = z_0$
**for** $k = 0, 1, 2, \ldots$ **do**
    $\alpha_k = \frac{z_k^T r_k}{d_k^T A d_k}$
    $x_{k+1} = x_k + \alpha_k d_k$
    $r_{k+1} = r_k - \alpha_k A d_k$
    Solve $Pz_{k+1} = r_{k+1}$
    $\beta_{k+1} = \frac{z_{k+1}^T r_{k+1}}{z_k^T r_k}$
    $d_{k+1} = z_{k+1} + \beta_{k+1} d_k$
**end for**
```

4 Implementation

Mesh vertex- and element-reordering techniques are effective only when the mesh is used several times. We incur cache misses in the process of determining an efficient ordering. If mesh vertices and elements are accessed only once, the same cache misses are incurred. The overhead cost of reordering elements is low when the mesh elements and vertices are accessed several times. Thus, the reordering techniques should be used only when the mesh vertices and elements are accessed several times. In this paper, we focus on the improvement in running times for the mesh warping application *after* the reordering technique is used. Since the overhead cost of reordering is low, a naive, serial implementation of the SFC reordering technique is used. An efficient parallel implementation of the mesh warping algorithm is used to report the improvement in run times. The implementation techniques are described below.

4.1 Hilbert Space-Filling Curve Reordering

Our serial implementation of the Hilbert SFC reordering algorithm reorders the mesh elements recursively. The location of a tetrahedral element is defined to be its centroid. The centroids are recursively subdivided into eight octants until at most one centroid point is present in every octant. These octants are ordered according to the Hilbert SFC [2]. After obtaining the order of the mesh elements, the vertices are reordered such that they follow the ordering of the elements, i.e., if an element, A, is ordered to appear before an element, B, the vertices of the element A would appear before the vertices of the element B.

4.2 Reverse Cuthill-McKee Reordering

We construct a graph with mesh elements as its nodes, and a pair of nodes are connected by an edge if the two corresponding elements share a face. The RCM ordering is obtained for this graph, and the elements are ordered accordingly. The vertices are ordered in the same way as above, i.e., if an element, A, is ordered to appear before an element, B, the vertices of the element A would appear before the vertices of the element B.

4.3 Stiffness Matrix Assembly

After reordering the mesh elements and vertices, we compute the finite element stiffness matrix that is used to solve the discretized form of Laplace's equations. The stiffness matrix is sparse, and the nonzero matrix elements correspond to the edge-connected vertices in the mesh. A local stiffness matrix is computed in parallel for every mesh element, and this matrix is added in parallel to the global stiffness matrix. Each thread is assigned a set of elements (i.e., contiguous blocks in the reordered array) for which the local stiffness matrix is computed. The global stiffness matrix is stored in an array of arrays. We construct an array of pointers in which each element points to an array, which corresponds to a row in the global stiffness matrix. Each of those row-assigned arrays stores the nonzero elements in their respective rows. The row-assigned arrays store the column data and the value of the element at the respective row-and-column location in the global stiffness matrix.

Computation of the local stiffness matrix for each element is distributed across the threads. As two elements may share an edge, and the vertices connected by the edge may be accessed by two threads in parallel, race conditions will arise when the global stiffness matrix is being updated in parallel. To avoid incorrect assembly of the global stiffness matrix due to race conditions, locks are employed to guarantee mutual exclusion. Our parallel global stiffness matrix assembly algorithm is shown in Algorithm 2.

When a matrix element in the global stiffness matrix is to be updated by a thread, the corresponding column entry in the row-assigned array is searched. If an entry for the column is found, a lock is acquired for that element, the entry is updated, and the lock is released. If the column entry is not present in the array, then a lock for the entire row-assigned array is acquired. Other threads may have added a matrix element corresponding to the row and the column after the search is complete but before the lock is acquired by our thread. Therefore, the row-assigned array is searched again to ensure that no other thread has added an element corresponding to the row and the column. If the element was added, the array is accordingly updated. If the element is still not present, a new entry is created, and the lock is released. This technique is efficient because the matrix is very sparse. The maximum number of nonzero elements in a row is typically only 20 or 30 even for meshes containing a billion vertices, and very few tetrahedral elements are updated by more than one thread.

4.4 Parallel Preconditioned Conjugate Gradient Method

In order to solve the symmetric positive definite linear system (1), we employ the PCG method as described in the previous section. The method was parallelized using OpenMP constructs as shared-memory processors were used for our numerical experiments. The implementation of most of the steps in the CG method is embarrassingly parallel. The PCG method involves sparse matrix-vector multiplication, which is parallelized by assigning to each processor a block of rows upon which to operate. The computation of the norm and dot product both require addition of elements in an array. They are both parallelized using the reduction operator construct in our OpenMP implementation.

Most preconditioners are parallelized efficiently by partitioning the mesh or coloring the vertices using graph-theoretic techniques [31], and the rows and columns (corresponding to the vertices) are reordered accordingly. Our intuition

Algorithm 2 Algorithm for synchronization of global stiffness matrix assembly in parallel

```
Input: $myRow$, $myCol$, $myValue$, vector<vector<double>> $myMatrix$
Output: Update the stiffness matrix element ($myRow$, $myCol$) with $myValue$
$n$ = length of the row $myRow$
for $i = 1$ to $n$ do
    if $myMatrix[myRow][i].col == myCol$ then
        Obtain a lock for the element
        Update $myMatrix[myRow][i].value += myValue$
        Release the lock for the element
        Return
    end if
end for
Obtain a lock to add an element to the row
$n_{new}$ = length of the row $myRow$.
for $i = n+1$ to $n_{new}$ do
    if $myMatrix[myrow][i].col == myCol$ then
        Update $myMatrix[myRow][i].value += myValue$
        Release lock for the row
        Return
    end if
end for
Add a new element to $myRow$
Release lock for the row
Return
```

is that such reordering techniques are not efficient with respect to cache utilization for stiffness matrix assembly. For this reason, we use the block-Jacobi preconditioner in our implementation. It is a common, effective preconditioner that can be easily implemented in parallel. We will show that our reordering technique lowers the number of iterations required to converge to a solution if the block-Jacobi preconditioner is used, i.e., when compared to the mesh with the default ordering of vertices and elements, the reordered mesh converges in fewer iterations. Note that a different preconditioner may be used for solving linear equations if it is found to be more efficient.

4.5 Parallel Block-Jacobi Preconditioner

The block-Jacobi preconditioner is constructed using square matrix blocks which form submatrices in our global stiffness matrix, and the union of the diagonals of the square submatrices form the diagonal of the stiffness matrix. We construct the submatrices using consecutive rows and columns and ensure that they do not overlap. This preconditioner results in a large number of independent linear systems, which can be solved in parallel using the Cholesky factorization technique followed by forward and backward substitution.

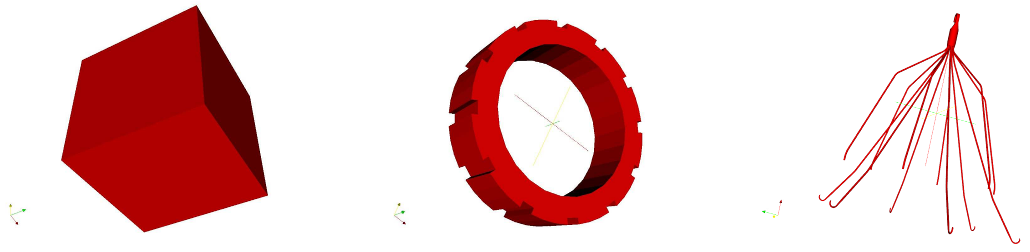
5 Numerical Experiments

We implemented the parallel FEMWARP algorithm using OpenMP constructs in C++ and executed it on the three domains shown in Fig. 2. The cube is a regular domain; the gear is topologically different, as there is a hole in the domain; the inferior vena cava (IVC) filter [32] is a medical device used to capture blood clots that are formed in deep veins in the human body; it is a domain with a small local feature size at any point in the domain. These diverse domains were chosen to characterize the performance of our SFC reordering algorithm on such domains. We generated tetrahedral meshes with more than 10 million elements using Tetgen [33]. As each core is assigned more then 20,000 degrees of freedom to compute, the meshes were sufficient to show the effectiveness of our technique. In order to simulate cache behavior, we also generated smaller meshes with more then a million elements and used the Simics [34] full-system simulator for our simulations. Smaller meshes containing a million elements were used because the simulation of the cache behavior for larger domains (with 10 million elements) takes an excessively long time (about 48 hours each).

Fig. 3 shows matrices with vertex connectivity information for the three domains with the default ordering obtained from Tetgen. Each row and column corresponds to a vertex. Fig. 4 shows the same information for the reordered meshes. The SFC reordering technique orders the vertices such that most of the elements move closer to the diagonal. Therefore, we expect better cache utilization and preconditioner efficiency while assembling the stiffness matrix and solving the linear equations, respectively.

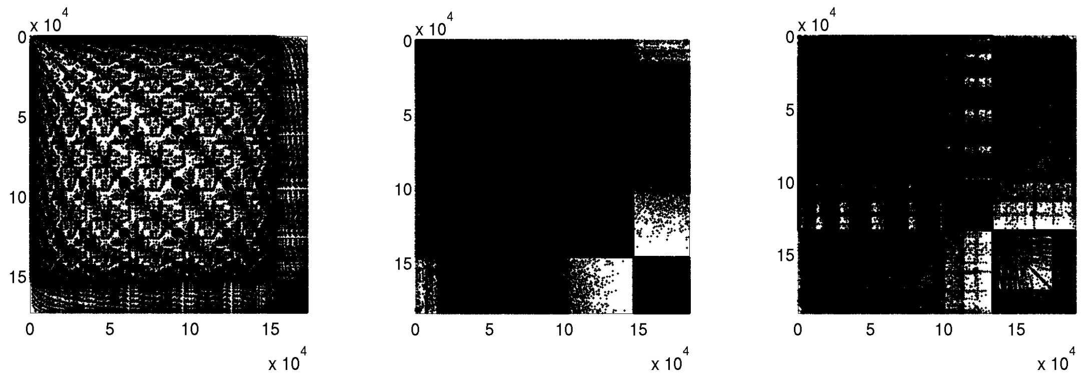
A set of preliminary experiments was carried out to determine an optimal block size for the block-Jacobi preconditioner. The results from these experiments were used for all the subsequent experiments, which includes cache utilization, performance improvement due to the SFC reordering technique, and scalability. For all our experiments, we perturbed the boundary vertices of our meshes by a unit distance in each of the three axes. Note that the last set of rows (and columns) correspond to boundary vertices. We chose the simple perturbation, as stiffness matrix assembly takes the same time regardless of the perturbation, and the PCG method takes sufficiently long to compare the results of our reordering technique with the results from the default ordering of vertices. The FEMWARP algorithm moves the interior vertices to warp the mesh from the initial domain to the new domain.

Table 1 reports the overhead time for reordering a mesh on a single core, Table 2 provides the number of vertices that are shared by multiple cores when the stiffness matrix is parallely assembled. Clearly, the overhead time is not significant when a mesh is used multiple times during the course



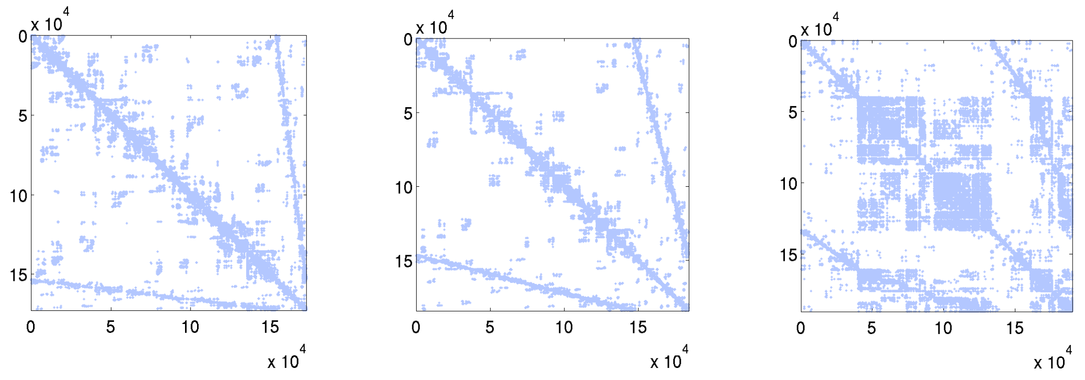
(a) cube (b) gear (c) IVC filter

Fig. 2 Domains on which the SFC reordering technique was applied and for which the meshes were warped.



(a) cube (b) gear (c) IVC filter

Fig. 3 Spy plots of the stiffness matrix for the three domains with the default ordering.



(a) cube (b) gear (c) IVC filter

Fig. 4 Spy plots of the stiffness matrix for the three domains with the SFC ordering of both vertices and elements.

of an iteration for the SFC reordering. The number of shared vertices increases with the number of cores as expected in most cases. For smaller number of partitions, the SFC ordering generally results in fewer shared vertices when compared to the RCM ordering. For larger number of partitions, the SFC ordering results in more shared vertices.

5.1 Block Size Experiment

A preliminary set of experiments are carried out on the default and reordered gear domains to determine the optimal block size to solve (1). We vary the block size for the preconditioner from one to eight, and the number of cores are varied from one to 48. Both the number of iterations and the time taken to solve the linear system are obtained for running the PCG algorithm.

Fig. 5 shows the number of CG iterations needed for the solution to converge as a function of the block size. For the default gear mesh, the block size does not play a significant role in reducing the number of iterations. However, for the reordered meshes, increasing the block size reduces the number of iterations needed to converge. Since more nonzeros are present near the diagonal for the reordered meshes, increasing the block size is effective.

Domain	# Vertices	# Elements	Ordering	Overhead Time
cube	1,718,389	10,777,036	default	-
			SFC	51.6
			RCM	419
gear	1,991,261	12,196,347	default	-
			SFC	55.7
			RCM	459
IVC filter	1,790,304	10,669,653	default	-
			SFC	55.2
			RCM	414

Table 1 The overhead time (in seconds) for reordering the vertices and elements of a mesh for both the SFC and RCM orderings.

Domain	Ordering	Number of Shared Vertices							
		Number of Cores							
		2	4	8	12	16	24	32	48
cube	default	1,557,890	1,698,416	1,714,838	1,716,623	1,717,509	1,718,003	1,718,137	1,718,265
	SFC	27,311	57,057	81,565	219,930	199,915	343,528	320,307	488,600
	RCM	35,482	87,352	186,534	283,767	381,414	574,044	766,376	1,123,533
gear	default	1,841,827	1,968,056	1,986,396	1,989,356	1,989,890	1,990,631	1,990,796	1,990,974
	SFC	6,948	15,776	70,237	466,122	427,617	687,256	914,044	1,099,137
	RCM	8,817	28,469	67,567	104,512	142,829	217,517	293,254	442,543
IVC filter	default	1,537,630	1,743,836	1,781,755	1,788,116	1,789,403	1,789,990	1,790,204	1,790,262
	SFC	96,352	467,400	1,055,949	1,235,017	1,244,286	1,379,797	1,427,270	1,493,447
	RCM	3,094	14,348	34,561	50,297	63,456	98,689	129,059	195,667

Table 2 The number of shared vertices when multiple cores are used to assemble the stiffness matrix in parallel.

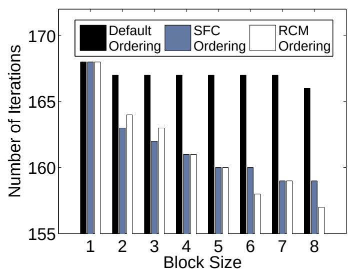


Fig. 5 The number of PCG iterations to solve the system of linear equations to determine the x-coordinates of the vertices in the reordered gear mesh as a function of the block size of the Jacobi preconditioner. The results are similar for the y- and z-coordinates.

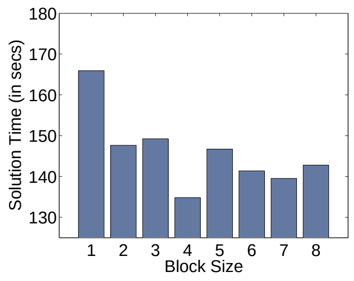


Fig. 6 Time taken to solve the system of linear equations to determine the x-coordinates of the vertices in the reordered gear mesh as a function of the block size of the Jacobi preconditioner. The results are similar for the y- and z-coordinates. Note that the y axis does not start from 0.

There are three factors that determine the total time to solve the system of linear equations when the block size is changed. First, the time taken to solve the system of linear equations depends upon the time per iteration. As block size increases, more time is required to solve each of the independent systems of linear equations. Second, the number of those independent systems of equations, and third, the number of PCG iterations both reduce with increasing block size. Fig. 6 shows the time required to solve the equations on eight cores for the SFC reordered mesh. Typically, a block size of four elements takes the least amount of time to solve the system of linear equations. For the rest of the experiments, we use a block size of four elements.

5.2 Cache Utilization Experiment

We simulate cache behavior using Simics [34] for a shared-memory processors with a shared L2 cache and private L1 caches for 1-, 2-, 4-, 8- and 16-core processors. In our simulations, 4 MB shared L2 caches and 64 KB L1 caches are used. For PCG, the sparse matrix-vector multiplication step is the only step in which cache utilization improvement can be seen due to the reordering technique. As there has also been prior work [35] showing such results, we simulate the cache behavior only for the stiffness matrix assembly.

Table 3 shows the number of cache misses incurred during each of our simulations. The results for the L1 cache hit rates for a 1-core processor are shown in Fig. 7. The data read miss rates for the L1 cache are reduced by almost 50%

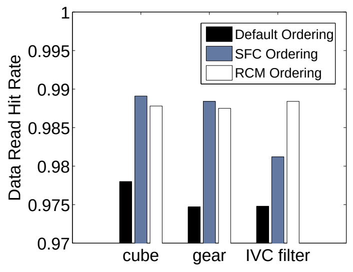


Fig. 7 Private L1 cache data read hit rate for the three domains for 1-processor execution. Note that the *y* axis does not start from 0.

in some cases due to the reordering. We find very similar results for 2- and 4-core simulations. The shared L2 cache performance also improves due to the reordering in most cases. The results for the L2 cache simulations are shown in Fig. 8. Note that the data transaction on the L2 cache happens only when an L1 cache miss happens. Therefore, where there is an L1 cache hit rate improvement, comparable hit rates of the L2 cache for the default meshes and the reordered meshes will result in improved performance for the reordered meshes.

We only find one case (the cube mesh on a 1-core processor) where the L2 cache hit rate slightly reduces due to the reordering. But since the number of data transactions on the L2 cache for the default mesh are twice as many as for the reordered mesh, the run time for the reordered cube mesh is lower than that for the default mesh. Between the two reordering techniques (SFC and RCM), we see that the SFC technique is more effective for the cube and gear meshes, whereas the RCM technique is more effective for the IVC filter mesh. This is because the IVC filter domain contains regions with a small feature size, and the space-filling curves lead to ordering in which elements that are geometrically far apart to be become next to each other.

5.3 Performance Improvement with the SFC Reordering Technique

We also measured the improvement in the run time due to the reordering on a real system. We ran our experiments on a 48-core multiprocessor system with four AMD Opteron 6174 processors connected using AMD Hypertransport links. Each processor has 12-cores running at 2.2 GHz. In each processor, each core has a 64KB private data cache, a 64KB private instruction cache, a 512KB private L2 cache, and all 12 cores share a 12MB L3 cache. The results for the relative time of execution for the reordered mesh with respect to the default mesh for stiffness matrix assembly are shown in Fig. 9. For a greater number of cores, the improvement obtained by reordering is nearly 15-20% for all three domains. The performance mostly improves with the increasing number of processors. Fig. 9 also shows that the SFC reordering algorithm is more effective for the cube and the gear mesh. For the IVC filter mesh, the RCM technique was found to be more effective.

When solving the system of linear equations, we see favorable results, i.e., a reduction in run time by about 5-15% in most cases due to the reordering. The results are shown in Fig. 10. The use of the SFC reordering technique takes less time than the RCM technique to solve the system of linear equations in most cases. The use of the SFC reordering technique improves the performance of the parallel implementation of the FE-based algorithms.

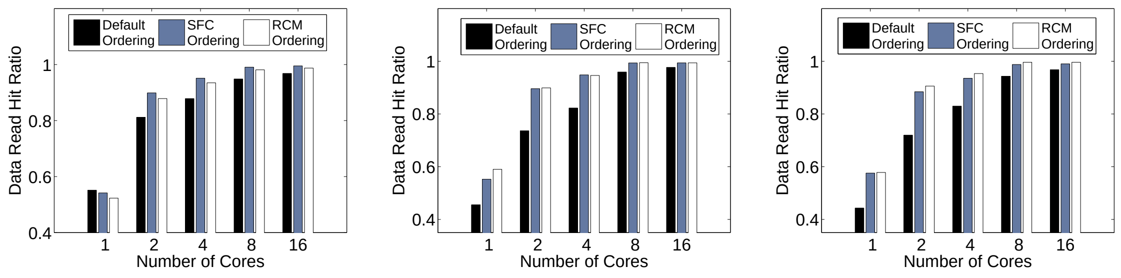
5.4 Scalability

We examined the strong scaling of the parallel implementation of the FEMWARP algorithm. Both the default and the reordered meshes are used in our experiments. The results are presented for the two stages of the algorithm, which are the a) stiffness matrix assembly and b) solution of the system of linear equations. For stiffness matrix assembly, the results for the gear mesh are presented in Fig. 11(a). The results for the other two domains are similar. For the default mesh, the efficiency of scaling is around 75% for 48 cores. Whereas, for the SFC reordered mesh, it is around 82%. For the RCM reordering, the efficiency is around 81% for 48 cores. Clearly, improvement in cache hit rates plays a major role in reducing the run time of the FEMWARP algorithm.

For solving the linear system of equations, the scaling is not as good as in the previous stage due to the serialization in computing quantities such as the vector norm and the dot product. The results are shown in Fig. 11(b). They are comparable to results in [36]. There is a slight improvement (about 2%) in the results for the SFC reordered meshes due to the reduction in the number of iterations as well as the improved cache utilization due to reordering. However, the RCM reordering technique does not help in scaling for the gear mesh. The efficiency is nearly identical to that of the default ordering.

6 Conclusions and Future Work

We demonstrated the effectiveness of the SFC reordering technique for FE methods in improving the cache utilization in parallel stiffness matrix assembly and in reducing the number of iterations via the use of the block-Jacobi preconditioner for solving the linear system using the PCG method. Due to reordering, synchronization constructs in the algorithm do not significantly affect the running time. About 20% improvement in the run time is obtained from reordering the mesh elements and vertices on shared-memory, multicore processors. We also showed that the strong scaling



(a) cube (b) gear (c) IVC filter

Fig. 8 L2 cache data read hit rate for the three domains for 1-, 2-, and 4-core execution.

Mesh	Ordering	Cache	Number of Cores				
			1	2	4	8	16
cube	Default	L1 L2	62,668,490 28,098,680	145,524,243 25,434,250	194,529,938 23,630,539	143,294,518 7,356,511	123,144,562 3,889,960
	SFC	L1 L2	31,105,850 14,258,469	127,395,303 11,152,992	174,836,885 8,504,900	115,138,039 1,072,793	104,446,498 476,748
	RCM	L1 L2	34,458,635 16,437,792	117,785,545 14,313,189	175,507,731 11,443,342	120,261,125 2,243,899	110,329,527 1,369,829
gear	Default	L1 L2	66,756,315 41,676,355	146,833,342 36,244,728	190,927,402 33,869,673	275,122,598 11,205,708	268,291,370 6,158,739
	SFC	L1 L2	31,302,744 1,725,6801	121,136,092 11,122,832	167,718,249 8,670,348	195,709,247 1,245,778	104,602,420 615,127
	RCM	L1 L2	33,557,100 13,753,467	114,427,214 11,569,333	168,859,365 9,101,523	197,644,124 1,096,128	109,980,945 646,180
IVC filter	Default	L1 L2	68,896,503 38,363,790	139,589,822 36,225,347	190,125,522 32,325,440	212,265,141 12,022,282	191,230,089 6,090,045
	SFC	L1 L2	37,077,173 19,184,474	124,940,594 13,040,197	171,336,297 11,029,769	213,125,900 2,598,272	106,670,298 1,030,516
	RCM	L1 L2	31,286,358 13,197,298	114,789,113 10,785,404	167,432,540 7,820,276	208,318,077 1,365,152	106,841,248 379,618

Table 3 The number of data read misses (NOT miss rate) for each of the domains for the L1 and L2 caches. The total number of misses are provided for all private L1 caches and for the shared L2 cache for each simulation. Note that the ratio of the total number of L1 misses to the number of L2 misses gives the miss rate for the L2 cache. For some cases, the miss rate for L1 and L2 caches can be also computed using Figs. 7 and 8. Those figures provide the hit rate, and thus, subtracting the hit rate from 1 gives the miss rate.

efficiency also improves due the reordering technique. The SFC reordering technique is most effective when multiple iterations of computations or several time steps are carried out (such as in [4] and [7]) for accurately solving PDEs.

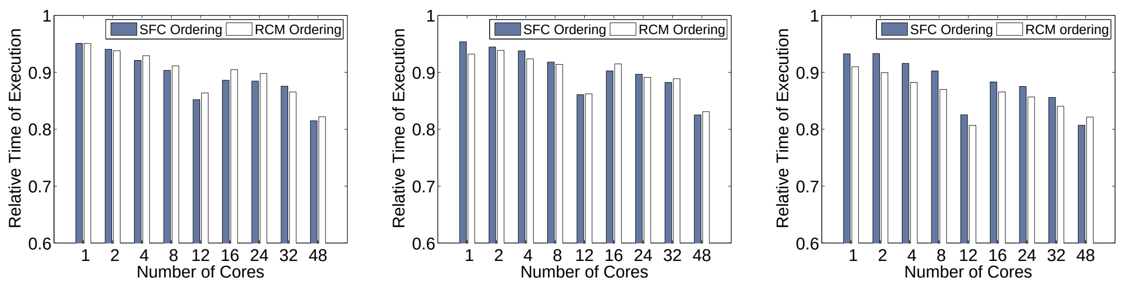
The termination criteria for recursion in the SFC reordering algorithm can be modified to a cache topology-aware condition in which the memory space occupied by storing the elements and vertices in each sub-octant is slightly less than the L2 cache size. Further reordering would likely provide diminishing returns.

In their paper, Oliker *et al.* [17] showed that combining a graph-partitioning technique and the SAW technique to reorder elements and vertices yields superior performance over each of the individual techniques. Since the SFC-based technique and the SAW technique are both geometric in nature, we expect that combining the SFC technique with a graph-partitioning technique would yield even better performance.

Graph coloring-based preconditioners are widely used [31] for solving sparse linear systems. An SFC-based reordering technique may also accelerate an algorithm that colors mesh vertices, and thus helps in solving the systems of linear equations much faster. Also, both the geometry and the algebra could be taken into account to develop a new preconditioner that can be used to solve the linear system more efficiently.

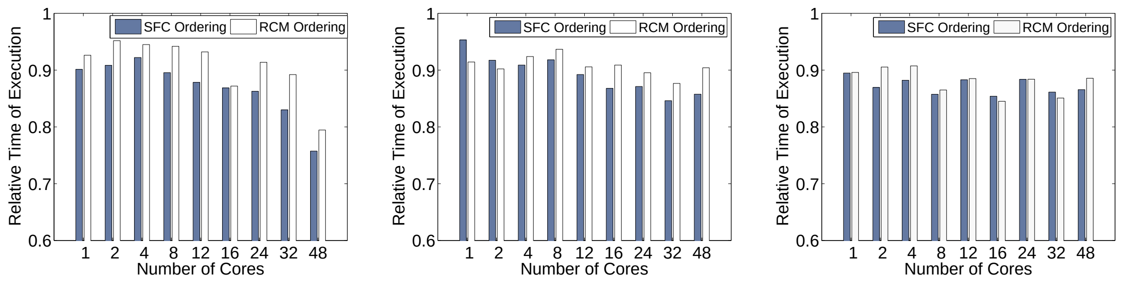
Acknowledgments

The authors would like to thank Rick Schraf and Todd Fetterolf for creating the CAD model of the IVC filter domain. The work of the first author is supported in part by the NSF Grant CNS-0720749, NSF CAREER Award OCI-1054459, NIH/NIGMS Center for Integrative Biomedical Computing, 2P41 RR0112553-12, and DOE NET DE-EE0004449 grants. The work of the third author was supported in part by NSF Grant CNS-0720749 and NSF CAREER Award ACI-1330056



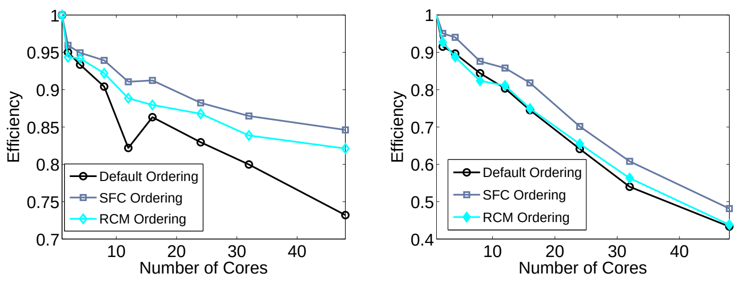
(a) cube (b) gear (c) IVC filter

Fig. 9 Relative time taken to assemble the stiffness matrix for the reordered gear mesh with respect to the default mesh for various numbers of cores.



(a) cube (b) gear (c) IVC filter

Fig. 10 Relative time taken to solve the system of linear equations to determine all the vertex coordinates in the reordered mesh with respect to the default mesh for various numbers of processors.



(a) Stiffness Matrix Assembly (b) Linear Equation Solution

Fig. 11 Strong scaling results for the gear mesh for the three ordering schemes. The results for the other two domains are similar.

(formerly OCI-1054459). This work of the second and fourth authors is supported in part by NSF grants 1147388, 1152479, 1017882, 0963839, 0720645, 0811687, 0702519, and a grant from Microsoft Corporation. The authors would also like to thank the three anonymous referee for their comments which improved the paper.

References

1. J. Shewchuk, "What is a good linear element? Interpolation, conditioning, and quality measures," in *Proc. of the* 11*th* *International Meshing Roundtable*, pp. 115–126, 2002.
2. H. Sagan, *Space-Filling Curves*. Springer, 1994.
3. S. Shontz and S. Vavasis, "Analysis of and workarounds for element reversal for a finite element-based algorithm for warping triangular and tetrahedral meshes," *BIT Numer. Math.*, vol. 50, pp. 863–884, 2010.
4. J. Park, S. Shontz, and C. Drapaca, "A combined level set/mesh warping algorithm for tracking brain and cerebrospinal fluid evolution in hydrocephalic patients," in *Image-based Modeling and Mesh Generation, Lecture Notes in Computational Vision and Biomechanics* (Y. Zhang, ed.), vol. 3, pp. 107–141, Springer, 2012.
5. J. Park, S. M. Shontz, and C. S. Drapaca, "Automatic boundary evolution tracking via a combined level set method and mesh warping technique: Application to hydrocephalus," in *Proc. of the Mesh Processing in Medi-*

cal Image Analysis 2012 - MICCAI 2012 International Workshop, MeshMed 2012, pp. 122–133, 2012.

6. J. Antaki, G. Blelloch, O. Ghattas, I. Malcevic, G. Miller, and N. Walkington, "A parallel dynamic-mesh lagrangian method for simulation of flows with dynamic interfaces," in *Proc. of the 2000 Supercomputing Conference*, 2000.
7. M. Adams and J. W. Demmel, "Parallel multigrid solvers for 3D unstructured element problems in large deformation elasticity and plasticity," *Int. J. Numer. Meth. Eng.*, pp. 48–65, 2000.
8. H. Adeli and O. Kamal, "Concurrent analysis of large structures-I: Algorithms," *Comput. Struct.*, vol. 42, no. 3, pp. 413 – 424, 1992.
9. H. Adeli and O. Kamal, "Concurrent analysis of large structures-II: Applications," *Comput. Struct.*, vol. 42, no. 3, pp. 425 – 432, 1992.
10. M. Rezende and J. Paiva, "A parallel algorithm for stiffness matrix assembling in a shared memory environment," *Comput. Struct.*, vol. 76, no. 5, pp. 593 – 602, 2000.
11. L. Chien and C. Sun, "Parallel processing techniques for finite element analysis of nonlinear large truss structures," *Comput. Struct.*, vol. 31, no. 6, pp. 1023 – 1029, 1989.
12. E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. of* 24^{th} *Nat. Conf. ACM*, pp. 157–172, 1969.
13. G. Heber, R. Biswas, G. Gao, Guang, and R. Gao, "Self-avoiding walks over adaptive unstructured grids," *Concurrency: Pract. Exper.*, vol. 12, pp. 85–109, 2000.
14. M. Zhou, O. Sahni, M. Shephard, C. Carothers, and K. Jansen, "Adjacency-based data reordering algorithm for acceleration of finite element computations," *Scientific Programming*, vol. 18, pp. 107–123, 2010.
15. H. Han and C. Tseng, "Exploiting locality for irregular scientific codes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 7, pp. 606–618, 2006.
16. M. Strout and P. Hovland, "Metrics and models for reordering transformations," in *Proc. of The Second ACM SIGPLAN Workshop on Memory System Performance (MSP)*, pp. 23–34, 2004.
17. L. Oliker, X. Li, P. Husbands, and R. Biswas, "Effects of ordering strategies and programming paradigms on sparse matrix computations," *SIAM Rev.*, vol. 44, no. 3, pp. 373–393, 2002.
18. L. Oliker, X. Li, G. Heber, and R. Biswas, "Parallel conjugate gradient: Effects of ordering strategies, programming paradigms, and architectural platforms," in *IEEE Trans. Parallel Distrib. Syst.*, 2000.
19. S. Shontz and P. Knupp, "The effect of vertex reordering on 2D local mesh optimization efficiency," in *Proc. of the* 17^{th} *International Meshing Roundtable*, pp. 107–124, 2008.
20. J. Park, P. Knupp, and S. Shontz, "Static vertex reordering schemes for local mesh quality improvement," tech. rep., Sandia National Laboratories, 2010.
21. S. Chatterjee, V. Jain, A. Lebeck, S. Mundhra, and M. Thottethodi, "Nonlinear array layouts for hierarchical memory systems," in *Proc. of the 1999 ACM International Conference on Supercomputing*, pp. 444–453, 1999.
22. T. Vo, T. Silva, F. Scheidegger, and V. Pascucci, "Simple and efficient mesh layout with space-filling curves," *J. Graph. Tools*, vol. 16, no. 1, pp. 25–39, 2012.
23. J. Behrens and J. Zimmermann, "Parallelizing an unstructured grid generator with a space-filling curve approach," in *EURO-PAR 2000*, pp. 815–823, Springer, 2000.
24. F. Alauzet and A. Loseille, "On the use of space filling curves for parallel anisotropic mesh adaptation," in *Proc. of the* 18^{th} *International Meshing Roundtable*, pp. 337–357, 2009.
25. A. Yzelman and R. Bisseling, "A cache-oblivious sparse matrixvector multiplication scheme based on the hilbert curve," in *Progress in Industrial Mathematics at ECMI 2010*, vol. 17 of *Mathematics in Industry*, pp. 627–633, Springer Berlin Heidelberg, 2012.
26. J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving memory hierarchy performance for irregular applications using data and computation reorderings," *Int J Parallel Prog*, vol. 29, no. 3, pp. 217–247, 2001.
27. T. Gerhold and J. Neumann, "The parallel mesh deformation of the DLR TAU-code," in *New Results in Numerical and Experimental Fluid Mechanics VI*, vol. 96 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pp. 162–169, Springer Berlin / Heidelberg, 2008.
28. H. M. Tsai, A. S. F. Wong, J. Cai, Y. Zhu, and F. Liu, "Unsteady flow calculations with a parallel multiblock moving mesh algorithm," *AIAA Journal*, vol. 39, pp. 1021–1029, June 2001.
29. J. George and J. Liu, "Computer solution of large sparse positive definite systems," *Prentice-Hall*, 1981.
30. D. Logan, *A First Course in the Finite Element Method*. Pacific Grove, CA, USA: Brooks/Cole Publishing Co., 2nd ed., 2000.
31. Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.
32. T. B. Kinney, "Inferior vena cava filters," *Semin Intervent Radiol.*, vol. 23, pp. 230–239, 2006.
33. H. Si, "TetGen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator." http://tetgen.berlios.de/.

34. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållbergv, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: A full system simulation platform,” *Computer*, vol. 35, no. 2, pp. 50–58, 2002.
35. R. Nishtala, R. Vuduc, J. Demmel, and K. Yelick, “Performance modeling and analysis of cache blocking in sparse matrix vector multiply,” tech. rep., University of California, Berkeley, 2004.
36. A. Gupta, V. Kumar, and A. Sameh, “Performance and scalability of preconditioned conjugate gradient methods on parallel computers,” tech. rep., Department of Computer Science, University of Minnesota, 1995.