NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2013; **20**:74–92
Published online 28 February 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.1826

Finding off-diagonal entries of the inverse of a large symmetric sparse matrix

Shawn Eastwood*,† and Justin W. L. Wan

Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada

SUMMARY

The method fast inverse using nested dissection (FIND) was proposed to calculate the diagonal entries of the inverse of a large sparse symmetric matrix. In this paper, we show how the FIND algorithm can be generalized to calculate off-diagonal entries of the inverse that correspond to 'short' geometric distances within the computational mesh of the original matrix. The idea is to extend the downward pass in FIND that eliminates all nodes outside of each node cluster. In our advanced downwards pass, it eliminates all nodes outside of each 'node cluster pair' from a subset of all node cluster pairs. The complexity depends on how far (i, j) is from the main diagonal. In the extension of the algorithm, all entries of the inverse that correspond to vertex pairs that are geometrically closer than a predefined length limit l will be calculated. More precisely, let α be the total number of nodes in a two-dimensional square mesh. We will show that our algorithm can compute $O(\alpha^{3/2+2\epsilon})$ entries of the inverse in $O(\alpha^{3/2+2\epsilon})$ time where $l = O(\alpha^{1/4+\epsilon})$ and $0 \leqslant \epsilon \leqslant 1/4$. Numerical examples are given to illustrate the efficiency of the proposed method. Copyright © 2012 John Wiley & Sons, Ltd.

Received 12 December 2010; Revised 23 December 2011; Accepted 30 December 2011

KEY WORDS: nested dissection; sparse matrix; matrix inversion; off-diagonal entries; computational mesh; computational complexity

1. INTRODUCTION

The concept of using nested dissection to factorize a sparse symmetric matrix was first proposed in [1]. Details about the original nested dissection algorithm can be found in [2]. The algorithm devised only encompassed the LDU factorization and did not return any entries from the inverse itself. In [3], the authors presented the fast inverse using nested dissection (FIND) algorithm in two parts. The first part, which is called the upwards pass, consisted of the original process of reordering and reducing the sparse symmetric matrix introduced in [1]. The second part, which is called the downwards pass, uses the elimination results that were calculated during the upwards pass to then calculate each diagonal entry of the inverse. An algorithm for recursively computing entries of the inverse of a sparse matrix using the LDU factorization and the matrix's adjacency graph is given in [4]. In a recent paper [5], the downwards pass is replaced with a recursive variant of the algorithm stated in [4] which enables the computation of all inverse entries that correspond to nonzero entries in the original matrix. However, this modification is unable to compute most entries of the inverse that correspond to the zero entries in the original matrix. A similar approach based on a hierarchy of Schur complements is developed in [6] for computing the diagonal of the inverse matrix.

The motivation given in [3] for finding the diagonal entries of the inverse of a large sparse matrix is the problem of finding the 'retarded Green's function' and the 'less-than Green's function' for

*Correspondence to: Shawn Eastwood, Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
†E-mail: shawn.eastwood@gmail.com

quantum nanodevices. The retarded Green's function is the inverse of the matrix $A = EI - H - \Sigma$, where E is a scalar denoting the nanosystem's energy, I is the identity matrix, H is the system's Hamiltonian operator, and Σ is a diagonal operator [3] (i.e., scalar potential) denoting the 'self-energy' of the system [7, 8]. The less-than Green's function is $G^< = G^r \Sigma^< (G^r)^\dagger$, where G^r denotes the retarded Green's function and $\dagger$ denotes the Hermitian transpose. $\Sigma^<$ is another diagonal matrix [3] denoting the 'less-than self energy' of the system [7, 8]. The less-than Green's function can be rewritten as $G^< = G^r \Sigma^< (G^r)^\dagger = A^{-1} \Sigma^< A^{-\dagger} = (A^\dagger (\Sigma^<)^{-1} A)^{-1}$. $\Sigma^<$ is a diagonal matrix, so computing $(\Sigma^<)^{-1}$ is straightforward. A and $A^\dagger$ are sparse, and $(\Sigma^<)^{-1}$ is diagonal, so $B = A^\dagger (\Sigma^<)^{-1} A$ can be assumed to be also sparse. Computing both $G^r = A^{-1}$ and $G^< = B^{-1}$ involve finding the inverse of a sparse symmetric matrix.

From the matrices G^r and $G^<$, the entries of G^r and $G^<$ are used to find the density of states, the electric charge density, and the current at each node in the computational mesh. At each node n in the computational mesh, the diagonal entry of G^r corresponding to n is used to compute the density of states at the current mesh node, the diagonal entry of $G^<$ corresponding to n is used to compute the electric charge density at the current mesh node, and the off-diagonal entries of $G^<$ corresponding to the edges leaving n are used to calculate the current density passing through the current mesh node [3, 8].

The nanodevice modeling example suggests that the diagonal entries of the inverse are most importantly followed by off-diagonal entries that correspond to connections that link adjacent nodes in the computational mesh. In other words, off-diagonal entries that correspond to node pairs that are close to each other are more important than off-diagonal entries that correspond to node pairs that are far from each other (see Figure 1). Our algorithm will attempt to calculate as much off-diagonal entries as possible while minimizing the time complexity. Let α denote the number of nodes present in a roughly square two-dimensional computational mesh, and let l be a geometric length threshold. If a pair of nodes from the computational mesh is separated by a geometric distance that is greater than l, the algorithm will not attempt to compute the entry of the inverse that corresponds to that pair of nodes. More precisely, if l is of order $O(\alpha^{1/4+\epsilon})$ where $0 \leqslant \epsilon \leqslant 1/4$ is arbitrary, then the extended FIND algorithm will compute the $O(\alpha^{3/2+2\epsilon})$ off-diagonal entries of the inverse that correspond to node pairs that are separated geometrically by no more than l in time $O(\alpha^{3/2+2\epsilon})$. This results in $O(1)$ time per entry of the inverse computed. We note that the complexity of the original FIND algorithm for computing the diagonal entries is $O(\alpha^{3/2})$. For $\epsilon = 0$, our algorithm is able to compute additional $O(\alpha^{3/2})$ off-diagonal entries with the same order of complexity of FIND.

Similar to FIND, our algorithm is based on the nested dissection method [1, 2]. We will make the standard assumption that the matrix is symmetric positive definite. This assumption is mainly for numerical stability consideration. Our algorithm itself does not rely on this property.

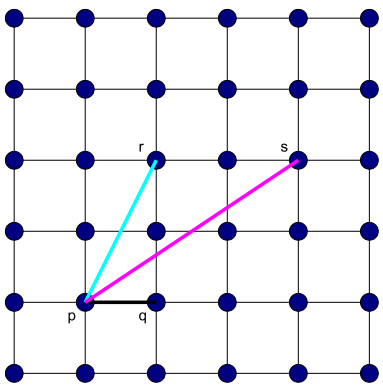

Figure 1. An example computational mesh showing some connections whose calculation in the inverse take precedence over others.

DOI: 10.1002/nla

Throughout this paper, we will use the following notation: if B denotes a matrix, then $B(S_r, S_c)$ will denote the submatrix of B whose row indices belong to the set S_r and column indices belong to the set S_c. Moreover, $B(S_r, S_c)$ will be referred to as the (S_r, S_c) block of B.

In Figure 1, the connection between nodes p and q has a graph distance of 1, whereas the connections between nodes p and r, and p and s have graph distances of 3 and 5, respectively. In this paper, we will place priority on computing entries of the inverse that correspond to shorter connections than entries that correspond to longer connections. Hence, the computation of $A^{-1}(p, q)$ takes priority over the computation of $A^{-1}(p, r)$ which in turn takes priority over the computation of $A^{-1}(p, s)$.

In Section 2, we will describe the FIND algorithm as presented in [3]. In Section 3, we will discuss an extension to the FIND algorithm in order to compute all off-diagonal entries of the inverse that correspond to mesh node pairs that are closer to each other than a certain length limit. In Section 4, we will theoretically analyze the computational complexity of computing off-diagonal entries of the inverse. Finally, in Section 5, we will give experimental results that verify the computational complexity presented in Section 4 for the case when $\epsilon = 0$.

2. THE FIND ALGORITHM

This section will describe the upwards and downwards passes as proposed in [3]. Throughout this and the subsequent sections, the large sparse symmetric matrix for which we will be computing entries of the inverse will be denoted by A.

Before we start describing the FIND algorithm, we will first define the 'node cluster' and 'node cluster tree' for the computational mesh.

Definition 1
A 'node cluster' is a subset of the set of nodes that makeup the computational mesh.

Definition 2
A 'node cluster tree' is a binary tree where each node in the tree denotes a node cluster. Node cluster trees obey the following properties:

- The root node cluster is the entire computational mesh.
- Given a nonleaf node cluster, the two child node clusters make up a partition of the parent node cluster.

To assist with the description of our proposed extension to the FIND algorithm, we will also assume that the node cluster tree is a perfectly balanced tree, meaning that all leaf node clusters are on the same level. If we happen to have a node cluster tree that is not perfectly balanced, then we can further subdivide shallower leaf node clusters until the tree is perfectly balanced. For the case where we would have to subdivide a leaf node cluster containing a single node, it is important to note that empty node clusters are still considered valid.

On any given level, all node clusters are assumed to have approximately the same number of mesh nodes. Leaf node clusters are assumed to have $O(1)$ mesh nodes. Figure 2 shows an example of how an 8×8 computational mesh (using the five-point stencil) is partitioned to form a node cluster tree. The root node cluster (Level 0) is the entire mesh as laid out in the definition. The central vertical line separates the root node cluster into two child node clusters (Level 1). The node clusters on Level 1 are, themselves, bisected by horizontal lines to obtain the four node clusters on Level 2. By recursively bisecting the node clusters, we eventually obtain 64 leaf node clusters, each consisting of a single node.

2.1. The upwards pass

We will first define the boundary, interior, and the private interior of a node cluster N.

DOI: 10.1002/nla

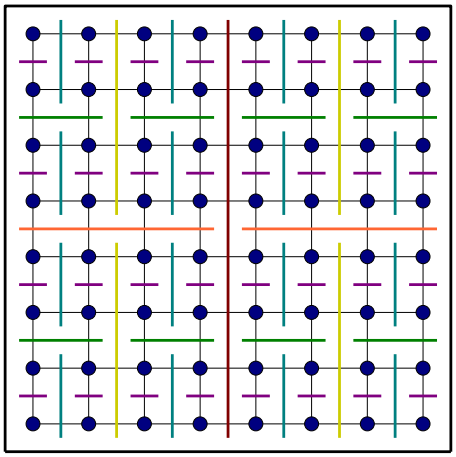

Figure 2. An example computational mesh with node cluster tree subdivisions.

Definition 3
Given a node cluster N, the 'boundary' of N, denoted by ∂N, is the set of all nodes in N that has a connection with a node outside of N.

The boundary of the root node cluster is always empty.

Definition 4
Given a node cluster N, the 'interior' of N, denoted by $Int(N)$, is the set of all nodes in N that *do not* have a connection with a node outside of N.

∂N and $Int(N)$ form a partition of N: $N = \partial N \sqcup Int(N)$, where $\sqcup$ denotes the disjoint union.

In Figure 3 (left), nodes that are contained within node cluster N are shaded, whereas nodes that are outside N are grid textured. The shaded area is ∂N, whereas the diagonal-textured area is $Int(N)$.

The action of eliminating a node n from the computational mesh is equivalent to pivoting from the (n,n) entry of the matrix during Gaussian elimination. Assume that node n is ordered first so that the matrix can be written as the following:

$$\begin{bmatrix} d & v^T \\ v & B \end{bmatrix}$$

where d is a scalar, v is a column vector, and B is a square matrix.

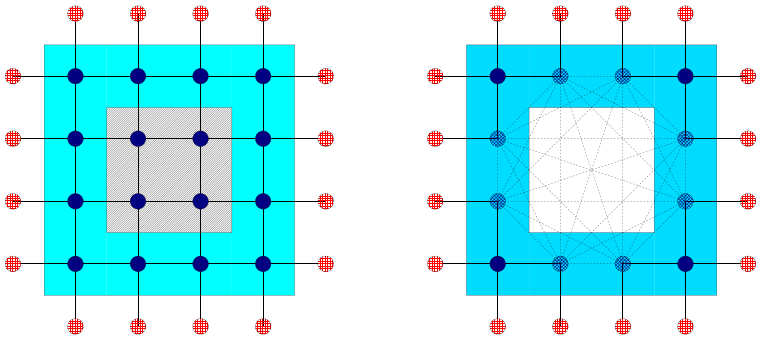

Figure 3. (Left) An example showing the boundary and interior of a node cluster and (right) the connectivity of the node cluster after the interior nodes have been eliminated.

DOI: 10.1002/nla

After eliminating node n, the matrix becomes:

$$\begin{bmatrix} d & v^T \\ 0 & B - \frac{vv^T}{d} \end{bmatrix}.$$

The remaining part of the matrix, $B - \frac{vv^T}{d}$, is symmetric, and the computational mesh associated with the remaining part is the same as before, only that n has been removed and all neighbors of n are now all mutually linked to account for all the fill-ins that occur. We will subsequently refer to the Schur complement $B - \frac{vv^T}{d}$ as the 'reduced matrix'.

The importance of defining the interior of node cluster N lies with the divide-and-conquer nature of the FIND algorithm. As long as we do not eliminate any boundary nodes, the effect of eliminating nodes from the interior has absolutely no effect outside the cluster. Given any two disjoint node clusters N_1 and N_2, the result of eliminating all nodes from $Int(N_1)$ can be superimposed on the result of eliminating all nodes from $Int(N_2)$ to obtain the result of eliminating all nodes from $Int(N_1) \sqcup Int(N_2)$. For a node cluster N, eliminating all nodes from $Int(N)$ only affects ∂N, so the only results we need to store is the $(\partial N, \partial N)$ block of the reduced matrix. This $(\partial N, \partial N)$ block is all that we need to call upon when we wish to instantly eliminate $Int(N)$ in another scenario (provided ∂N is untouched). The overall aim of the upwards pass is to, for each node cluster N, calculate the $(\partial N, \partial N)$ block that remains once all nodes from $Int(N)$ have been eliminated.

Figure 3 (right) shows the result of eliminating the interior of the node cluster. The dashed edges are edges that have been added (fill-in) or changed, and the diamond-textured nodes are nodes whose corresponding diagonal entry has been changed. Notice the complete graph of eight nodes mutually connecting all boundary nodes that had a connection with the interior. The four corners of the cluster had no connection with the interior and are hence not part of the complete graph. Notice how the changes are all constrained to the cluster itself.

Assume that we now wish to eliminate the interior of a nonleaf node cluster N that has children M_1 and M_2. We start by eliminating $Int(M_1)$ and $Int(M_2)$ which are both disjoint subsets of $Int(N)$. When we eliminate $Int(M_1)$, the effect of the elimination is constrained to M_1. When we eliminate $Int(M_2)$, the effect of the elimination is constrained to M_2. After eliminating both $Int(M_1)$ and $Int(M_2)$, the set of nodes remaining in $Int(N)$ is called the private interior of N. The private interior of N is what we must eliminate in order to complete the elimination of $Int(N)$.

Definition 5
If N is a nonleaf cluster with children M_1 and M_2, then the 'private interior' of N, denoted by $pi(N)$, is defined by $pi(N) = Int(N) - (Int(M_1) \cup Int(M_2))$. For convenience, if N is a leaf cluster, then we simply let $pi(N) = Int(N)$.

For a nonleaf cluster N with children M_1 and M_2, $\partial N \sqcup pi(N) = \partial M_1 \sqcup \partial M_2$. This equation is important as it shows that $pi(N)$ is all we need to subsequently remove for us to be left with ∂N.

Figure 4 shows the steps involved in the elimination of the interior of a nonleaf node cluster N with children M_1 and M_2. The shaded region is ∂N, whereas the diagonal-textured region is $pi(N)$. The top mesh shows the mesh before any elimination of $Int(N)$ has occurred. The middle mesh shows the elimination of $Int(M_1)$ and $Int(M_2)$. The final mesh shows the middle mesh after $pi(N)$ has been removed. These steps form the basic idea of the upwards pass algorithm.

Algorithm 1 describes the upwards pass algorithm in full detail. Let $\mathbf{A}$ be the original sparse matrix before any nodes have been removed. Consider a node cluster N with children M_1 and M_2. The goal of the upwards pass is to compute $U_N(\partial N, \partial N)$ which is the reduced matrix $\mathbf{A}$ after the interior of node cluster N has been removed. The idea is to eliminate the interior of node cluster N by eliminating the interior of its children M_1 and M_2.

The algorithm starts with computing U_{M_1} and U_{M_2} by a recursive call to the upwards pass. It then computes $U_{c(N)}$, the reduced matrix $\mathbf{A}$ after the interiors of M_1 and M_2 have been removed. (Here, $c(N)$ refers to the children of N.) Note that eliminating the interior of M_1

DOI: 10.1002/nla

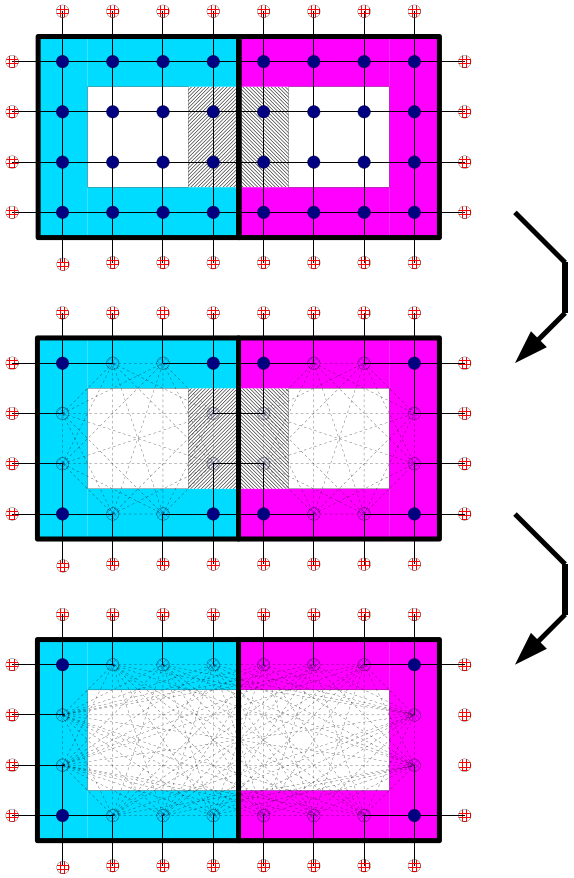

Figure 4. The stages in the elimination of the interior of a node cluster N with children M_1 and M_2: (top) no elimination, (middle) elimination of interiors of M_1 and M_2, and (bottom) further elimination of the private interior of N.

only affects the $(\partial M_1, \partial M_1)$ block of A. Similarly, eliminating the interior of M_2 only affects the $(\partial M_2, \partial M_2)$ block. As a result, if n_1 and n_2 are two nodes from ∂M_1 (∂M_2), then $U_{c(N)}(n_1, n_2) = U_{M_1}(n_1, n_2)$ ($U_{c(N)}(n_1, n_2) = U_{M_2}(n_1, n_2)$). If one node is from M_1 and the other from M_2, then $U_{c(N)}(n_1, n_2) = \mathbf{A}(n_1, n_2)$.

Consider nodes in ∂N, and let $S_1 = \partial N \cap M_1$ and $S_2 = \partial N \cap M_2$. Index the nodes in M_1 first, followed by the nodes in M_2. Using the aforementioned formulas for $U_{c(N)}$, we have

$$U_{c(N)}(\partial N, \partial N) = \begin{bmatrix} U_{M_1}(S_1, S_1) & \mathbf{A}(S_1, S_2) \\ \mathbf{A}(S_2, S_1) & U_{M_2}(S_2, S_2) \end{bmatrix}. \tag{1}$$

Similarly, let $P_1 = pi(N) \cap M_1$ and $P_2 = pi(N) \cap M_2$. We have

$$\begin{aligned} U_{c(N)}(\partial N, pi(N)) &= \begin{bmatrix} U_{M_1}(S_1, P_1) & \mathbf{A}(S_1, P_2) \\ \mathbf{A}(S_2, P_1) & U_{M_2}(S_2, P_2) \end{bmatrix}, \\ U_{c(N)}(pi(N), pi(N)) &= \begin{bmatrix} U_{M_1}(P_1, P_1) & \mathbf{A}(P_1, P_2) \\ \mathbf{A}(P_2, P_1) & U_{M_2}(P_2, P_2) \end{bmatrix}, \\ U_{c(N)}(pi(N), \partial N) &= \begin{bmatrix} U_{M_1}(P_1, S_1) & \mathbf{A}(P_1, S_2) \\ \mathbf{A}(P_2, S_1) & U_{M_2}(P_2, S_2) \end{bmatrix}. \end{aligned} \tag{2}$$

DOI: 10.1002/nla

Algorithm 1 ($U_N(\partial N, \partial N)$ = UpwardsPass(N)

if ClusterType(N) = Leaf **then**
{If N is a leaf node cluster, we simply eliminate the interior.}
{Note our convention that for a leaf node cluster, $pi(N) = Int(N)$.}
$U_N(\partial N, \partial N) = A(\partial N, \partial N) - A(\partial N, pi(N))A(pi(N), pi(N))^{-1}A(pi(N), \partial N)$
else
$U_{M_1}(\partial M_1, \partial M_1) =$ UpwardsPass(M_1)
$U_{M_2}(\partial M_2, \partial M_2) =$ UpwardsPass(M_2)
{Compute $U_{c(N)}(\partial N, \partial N)$, $U_{c(N)}(\partial N, pi(N))$, $U_{c(N)}(pi(N), pi(N))$, and $U_{c(N)}(pi(N), \partial N)$ by equations (1) and (2).}
$U_N(\partial N, \partial N) = U_{c(N)}(\partial N, \partial N)$
$\quad -U_{c(N)}(\partial N, pi(N))U_{c(N)}(pi(N), pi(N))^{-1}U_{c(N)}(pi(N), \partial N)$
end if
return $U_N(\partial N, \partial N)$

Finally, we compute $U_N(\partial N, \partial N)$ by eliminating the $pi(N)$ nodes from the computational mesh, leaving us with the Schur complement:

$$U_N(\partial N, \partial N) = U_{c(N)}(\partial N, \partial N) - U_{c(N)}(\partial N, pi(N))U_{c(N)}(pi(N), pi(N))^{-1}U_{c(N)}(pi(N), \partial N).$$

In the case where N is the root cluster, $\partial N = \emptyset$, so $U_N(\partial N, \partial N)$ stores an empty (0×0) matrix.

2.2. *The downwards pass*

After the upwards pass has been terminated, we will have computed $U_N(\partial N, \partial N)$ for every node cluster N.

Suppose that we are inverting $\mathbf{A}$ by performing simple forward reduction on the augmented matrix $\left[\begin{array}{cc} \mathbf{A} & I \end{array}\right]$. Let the arbitrary node n occur last in the ordering. After pivoting from all of the previous nodes, the bottom row becomes $\left[\begin{array}{ccccc|ccc} 0 & \ldots & 0 & u & | & * & \ldots & * & 1 \end{array}\right]$. u denotes the (n,n) entry that remains after all nodes, other than n, have been eliminated, whereas each $*$ denotes an unknown fill-in. Thus, $\mathbf{A}^{-1}(n,n) = \frac{1}{u}$. Similarly, if node cluster N occurs last in the ordering, then $\mathbf{A}^{-1}(N,N) = U^{-1}$, where U is the (N,N) block of the reduced $\mathbf{A}$ after all nodes outside of N have been removed.

The overall aim of the downwards pass is to calculate $\mathbf{A}^{-1}(N,N)$ for every leaf node cluster N. As described earlier, it is equivalent to finding the (N,N) block of the reduced matrix $\mathbf{A}$ obtained from eliminating all nodes outside of N. We first define the following.

Definition 6
Given a node cluster N, the 'complement' of N, denoted by $\overline{N}$, is the set of all mesh nodes outside of N.

Definition 7
Given a node cluster N, the 'adjacent' of N, denoted by $\varrho N = \partial\overline{N}$, is the boundary of the complement of N.

Definition 8
Given a node cluster N, the 'exterior' of N, denoted by $Ext(N) = Int(\overline{N})$, is the interior of the complement of N.

Similar to the upwards pass, we employ a divide-and-conquer method for eliminating the exterior nodes in the case of nonroot clusters. Consider a node cluster N with parent cluster P and sibling

DOI: 10.1002/nla

cluster S. To eliminate the exterior of N, we first eliminate the exterior of P and the interior of S (note that $\overline{N} = \overline{P} \sqcup S$). We will call the set of remaining nodes as the private exterior of N.

Definition 9
Given a nonroot node cluster N with parent cluster P and sibling cluster S, the 'private exterior' of N, denoted by $pe(N)$, is defined by $pe(N) = Ext(N) - (Ext(P) \cup Int(S))$.

Figure 5 depicts the adjacent and private exterior set for a node cluster N with parent cluster P and sibling cluster S. The dark/red area is the adjacent set $\varrho N \cap \overline{P}$ of N, and the light/grey area is the private exterior set $pe(N)$ of N.

The notion of the exterior and private exterior node sets were not part of the notation used in [3]. These definitions are introduced in this paper to simplify the discussion.

Algorithm 2 describes the downwards pass algorithm in full detail. The goal of the downwards pass is to compute $U_{\overline{N}}(\varrho N, \varrho N)$, the reduced matrix $\mathbf{A}$ after the exterior of N has been removed. The idea is to eliminate the exterior of N by eliminating the exterior of its parent P and the interior of its sibling S.

The algorithm starts with calling DownwardsPass(Root), where Root denotes the root node cluster. After the computation of $U_{\overline{N}}$ (which is just an empty matrix for Root), it makes a recursive call to compute U_{M_1} and U_{M_2} for its children M_1 and M_2.

At a general recursive step for a node cluster N, $U_{\overline{P}}$ has already been computed for its parent cluster P. Also, U_S can be obtained from the upwards pass for its sibling cluster S. The algorithm then computes $U_{f(N)}$, the result of eliminating both the exterior of the parent, and the interior of the sibling. (Here, $f(N)$ refers to the 'family' of N.) Note that eliminating the exterior of P will only affect the $(\varrho P, \varrho P)$ block of $\mathbf{A}$. Similarly, eliminating the interior of S will only affect the $(\partial S, \partial S)$ block. Thus, if n_1 and n_2 are two nodes from ϱP (∂S), then $U_{f(N)}(n_1, n_2) = U_{\overline{P}}(n_1, n_2)$ $(U_{f(N)}(n_1, n_2) = U_S(n_1, n_2))$. Otherwise, $U_{f(N)}(n_1, n_2) = \mathbf{A}(n_1, n_2)$. This is similar to the computation of $U_{c(N)}$ in the upwards pass.

Let $X_1 = \varrho N \cap \overline{P}$, $X_2 = \varrho N \cap S$, $Y_1 = pe(N) \cap \overline{P}$, and $Y_2 = pe(N) \cap S$. Also, index the nodes in $\overline{P}$ first, followed by the nodes in S. Using the aforementioned formulas for $U_{f(N)}$, we have

$$
\begin{aligned}
U_{f(N)}(\varrho N, \varrho N) &= \begin{bmatrix} U_{\overline{P}}(X_1, X_1) & \mathbf{A}(X_1, X_2) \\ \mathbf{A}(X_2, X_1) & U_S(X_2, X_2) \end{bmatrix}, \\
U_{f(N)}(\varrho N, pe(N)) &= \begin{bmatrix} U_{\overline{P}}(X_1, Y_1) & \mathbf{A}(X_1, Y_2) \\ \mathbf{A}(X_2, Y_1) & U_S(X_2, Y_2) \end{bmatrix}, \\
U_{f(N)}(pe(N), pe(N)) &= \begin{bmatrix} U_{\overline{P}}(Y_1, Y_1) & \mathbf{A}(Y_1, Y_2) \\ \mathbf{A}(Y_2, Y_1) & U_S(Y_2, Y_2) \end{bmatrix}, \\
U_{f(N)}(pe(N), \varrho N) &= \begin{bmatrix} U_{\overline{P}}(Y_1, X_1) & A(Y_1, X_2) \\ \mathbf{A}(Y_2, X_1) & U_S(Y_2, X_2) \end{bmatrix}.
\end{aligned}
\tag{3}
$$

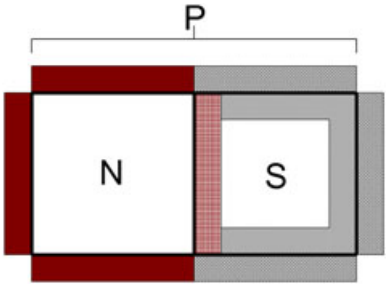

Figure 5. An example showing the adjacent (dark) and private exterior (light) of a nonroot node cluster N.

DOI: 10.1002/nla

Algorithm 2 ($U_{\overline{N}}(\varrho N, \varrho N)$ = DownwardsPass(N)

if ClusterType(N) = Root **then**
{If N is the root node cluster, then $\varrho N = \emptyset$}
$U_{\overline{N}}(\varrho N, \varrho N) = [\,]$ (a 0x0 empty matrix)
else
{Compute $U_{f(N)}(\varrho N, \varrho N)$, $U_{f(N)}(\varrho N, pe(N))$, $U_{f(N)}(pe(N), pe(N))$, and $U_{f(N)}(pe(N), \varrho N)$ by equation (3).}
$U_{\overline{N}}(\varrho N, \varrho N) = U_{f(N)}(\varrho N, \varrho N)$
$-U_{f(N)}(\varrho N, pe(N))U_{f(N)}(pe(N), pe(N))^{-1}U_{f(N)}(pe(N), \varrho N)$
end if
if ClusterType(N) = Leaf **then**
{If we have a leaf node cluster, we finally eliminate the adjacent node set and invert the remaining (N, N) block.}
$A^{-1}(N, N) = (A(N, N) - A(N, \varrho N)U_{\overline{N}}(\varrho N, \varrho N)^{-1}A(\varrho N, N))^{-1}$
else
{If we do not have a leaf node cluster, we pass the results of the computation down to the children of N.}
$U_{\overline{M_1}}(\varrho M_1, \varrho M_1) = \text{DownwardsPass}(M_1)$
$U_{\overline{M_2}}(\varrho M_2, \varrho M_2) = \text{DownwardsPass}(M_2)$
end if
return $U_{\overline{N}}(\varrho N, \varrho N)$

Then, we can compute $U_{\overline{N}}(\varrho N, \varrho N)$ by eliminating the $pe(N)$ nodes from the computational mesh, leaving us with the Schur complement:

$$U_{\overline{N}}(\varrho N, \varrho N) = U_{f(N)}(\varrho N, \varrho N) - U_{f(N)}(\varrho N, pe(N))U_{f(N)}(pe(N), pe(N))^{-1}U_{f(N)}(pe(N), \varrho N).$$

At the leaf node cluster, we finally eliminate the adjacent node set ϱN and invert the remaining (N, N) block.

3. THE ADVANCED FIND ALGORITHM

This section will describe the extensions to the FIND algorithm that will be made to compute the off-diagonal entries of the inverse of **A**.

As in the previous section, suppose that we are inverting **A** by performing simple forward reduction on the augmented matrix $\left[\begin{array}{cc} A & I \end{array}\right]$. Let the nodes n_1 and n_2 be the second-to-last and the last nodes in the ordering, respectively. After pivoting from all nodes other than n_1 and n_2, the bottom two rows of the augmented matrix become $\left[\begin{array}{ccccc|ccccc} 0 & \dots & 0 & & & * & \dots & * & 1 & 0 \\ 0 & \dots & 0 & \multicolumn{2}{c}{U} & * & \dots & * & 0 & 1 \end{array}\right]$. U denotes the $\{n_1, n_2\} \times \{n_1, n_2\}$ block that remains after all nodes other than n_1 and n_2 have been removed, whereas each $*$ denotes an unknown fill-in. Multiplying the bottom two rows by U^{-1} gives $\left[\begin{array}{ccccc|ccc} 0 & \dots & 0 & 1 & 0 & * & \dots & * \\ 0 & \dots & 0 & 0 & 1 & * & \dots & * \end{array}\right]$. The bottom two rows are not affected by any of the subsequent row operations during back substitution, so we can conclude that $A^{-1}(n_1, n_2) = U^{-1}(1, 2)$ and $A^{-1}(n_2, n_1) = U^{-1}(2, 1)$. Similarly, if node clusters N_1 and N_2 occur last in the ordering, then $A^{-1}(N_1, N_2) = U^{-1}(N_1, N_2)$ and $A^{-1}(N_2, N_1) = U^{-1}(N_2, N_1)$, where U denotes the $(N_1 \sqcup N_2) \times (N_1 \sqcup N_2)$ block that remains after all nodes that are outside of $N_1 \sqcup N_2$ have been removed.

To summarize, FIND algorithm (downwards pass) computes the diagonal entries, $A^{-1}(n, n)$, by eliminating the exterior of each node cluster. The main idea of our proposed method ('advanced

DOI: 10.1002/nla

downwards pass') for computing the off-diagonal entries, $A^{-1}(n_1, n_2)$, is to eliminate the exterior of what we call the 'node cluster pairs'.

3.1. Node cluster pairs

In this section, we define the notion of 'node cluster pairs', which is the key for computing off-diagonal entries of the inverse. In addition, we extend the concepts of the node cluster tree, complement, adjacency, (private) interior, and (private) exterior for node cluster pairs.

Definition 10
A 'node cluster pair' is an unordered pair of distinct node clusters that both exist on the same level in the node cluster tree.

We now define the different types of node cluster pairs.

Definition 11
A node cluster pair is called a 'twin pair' if the node clusters are siblings of each other. A node cluster pair is called a 'leaf pair' if the node clusters exist at the leaf level of the node cluster tree.

We note that there is no ambiguity in the definition of the leaf pair because we specified that the node cluster tree will be perfectly balanced with all leaf clusters on the same level. Also, it is possible for a node cluster pair to be both a twin pair and a leaf pair.

To accommodate the computation involving node cluster pairs, we need to extend the structure of the node cluster tree. The motivation behind these extensions will become clear when we describe the recursive algorithms that makeup the advanced downwards pass. The changes are as follows:

- Any nonleaf node cluster N with children M_1 and M_2 will receive a twin pair as an additional child. This twin pair will consist of the children of N, that is $\{M_1, M_2\}$, and is called the 'twin child' of N.
- Every nonleaf node cluster pair $\{N_1, N_2\}$ has four children:
  - $child11(\{N_1, N_2\}) = \{child1(N_1), child1(N_2)\}$,
  - $child12(\{N_1, N_2\}) = \{child1(N_1), child2(N_2)\}$,
  - $child21(\{N_1, N_2\}) = \{child2(N_1), child1(N_2)\}$,
  - $child22(\{N_1, N_2\}) = \{child2(N_1), child2(N_2)\}$.

After making the earlier extensions, we now have what we call the 'extended node cluster tree'. Figure 6 gives an extended node cluster tree for a 2×2 computational mesh with a five-point stencil. Each node cluster/pair is denoted by a 2×2 grid. For example, the root node cluster occupies the entire grid and is denoted by the dark-shaded square. It has two children and each covers half of the nodes. Nodes that are not part of the cluster are white squares. In the extended node cluster tree, each node has an additional twin child, connected by a dashed edge. The twin child pair of the root node cluster consists of two node clusters: the first cluster (left child of root) is shown by dark-shaded squares; and the second cluster (right child of root) is shown by lightly shaded squares. The remaining cluster nodes and twin child pairs are shown similarly. Notice that node cluster pairs have four children and no twin child.

We now extend the notations from the previous section to node cluster pairs. Given any node cluster pair $\{N_1, N_2\}$, the complement cluster $\overline{\{N_1, N_2\}} = \overline{N_1} \cap \overline{N_2}$, the adjacent set $\varrho\{N_1, N_2\} = \partial\overline{\{N_1, N_2\}}$, and the exterior $Ext(\{N_1, N_2\}) = Int(\overline{\{N_1, N_2\}})$ are defined in a similar fashion to how they are defined for node clusters. Note that ϱN_1 and ϱN_2 are not necessarily disjoint nor is their union equal to $\varrho\{N_1, N_2\}$. Figure 7 (left) shows how ϱN_1 and ϱN_2 are not always disjoint, and Figure 7 (right) shows how the union of ϱN_1 and ϱN_2 is not always equal to $\varrho\{N_1, N_2\}$.

In addition, the sibling pair $\{S_1, S_2\}$ is defined such that S_1 and S_2 are the sibling clusters to N_1 and N_2, respectively. Given any nontwin pair $\{N_1, N_2\}$ with parent pair $\{P_1, P_2\}$ and sibling pair $\{S_1, S_2\}$, the private exterior $pe(\{N_1, N_2\}) = Ext(\{N_1, N_2\}) - (Ext(\{P_1, P_2\}) \cup Int(S_1) \cup Int(S_2))$ is defined in a similar fashion to how it is defined for node clusters.

DOI: 10.1002/nla

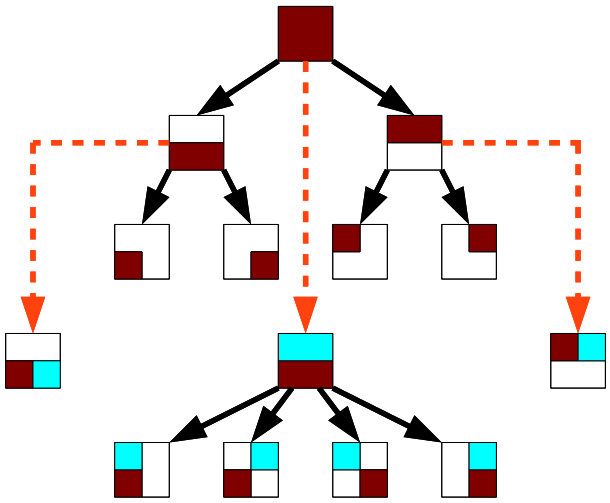

Figure 6. The extended node cluster tree for a 2×2 computational mesh.

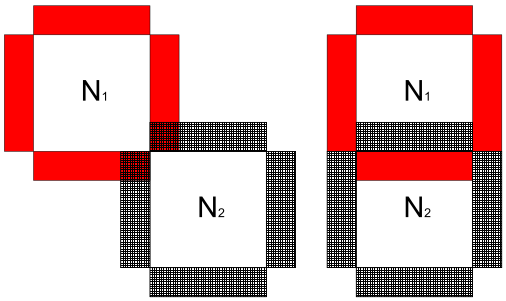

Figure 7. (Left) Adjacent sets ϱN_1 and ϱN_2 intersect at two corner nodes and (right) the union of adjacent sets ϱN_1 and ϱN_2 includes nodes between N_1 and N_2 which do not belong to $\varrho\{N_1, N_2\}$.

3.2. Minimum separation

Before we describe the advanced downwards pass, we remark that, ideally, one would want to compute the elimination of the exterior for *all* node cluster pairs which would in turn enable one to compute *every* entry of A^{-1}. Because of time complexity constraints, however, we will limit ourselves to only work with pairs whose 'minimum separation' is less than a user-defined threshold.

Definition 12
Let $d(n_1, n_2)$ denote the graph distance between nodes n_1 and n_2 as measured on the computational mesh. The 'minimum separation' between node clusters N_1 and N_2, denoted by $d(N_1, N_2)$, is the minimum value of $d(n_1, n_2)$, where $n_1 \in N_1$ and $n_2 \in N_2$. For a node cluster pair $\{N_1, N_2\}$, the minimum separation of $\{N_1, N_2\}$ is simply $d(N_1, N_2)$.

Let l be the cut-off distance that determines the node cluster pairs for which we will calculate the elimination of the exterior. If the minimum separation of a node cluster pair is less than or equal to l, then we compute the elimination of the exterior for this pair. If the minimum separation of a node cluster pair is greater than l, then we do not compute the elimination of the exterior for this pair. We make two observations.

Observation 1: If $d(N_1, N_2) > l$, then the minimum separations of all child pairs of $\{N_1, N_2\}$ are also greater than l. The moment we exclude a pair from our calculations because the minimum separation is greater than l, we can exclude the entire tree rooted at that pair from our calculations.

DOI: 10.1002/nla

Observation 2: If $d(N_1, N_2) \leqslant l$, then there exists at least one child pair of $\{N_1, N_2\}$ that has a minimum separation less than or equal to l. When we include a pair in our calculations because the minimum separation is less than or equal to l, we can move on to at least one of the child pairs.

The aforementioned two observations imply that if we restrict the extended node cluster tree as to exclude pairs whose minimum separation is greater than l, the resultant tree is connected because of the first observation. Also, there are no 'dead ends' in the sense that there always exists a nonempty child pair before we reach the leaf node or leaf-pair level because of the second observation.

3.3. Advanced downwards pass

We will now describe the advanced downwards pass, which will replace the downwards pass. The essence of the advanced downwards pass is not much different from the downward pass. It computes the elimination of the exterior of the node clusters. However, in addition, it also computes the elimination of the exterior of node cluster pairs. This is the main difference from the downwards pass that enables us to compute off-diagonal entries of the inverse.

The advanced downwards pass consists of two recursive subroutines: ClusterDownwardsPass and PairDownwardsPass. ClusterDownwardsPass operates on input node cluster N and computes $U_{f(N)}$ and $U_{\overline{N}}$ in the same way as the downwards pass, see Algorithm 3. At the leaf node, it computes $A^{-1}(N, N)$, the diagonal entries of the inverse. To compute the off-diagonal entries, it also calls PairDownwardsPass on the twin child (cf. Figure 6) of a nonleaf cluster node N.

PairDownwardsPass computes $U_{\overline{\{N_1,N_2\}}}$, the reduced matrix $\mathbf{A}$ after the exterior of the node cluster pair $\{N_1, N_2\}$ has been removed. The algorithm resembles the procedures of the downwards pass. It merges the results of the elimination of the exterior of the parent pair with the results of the elimination of the interiors of the sibling clusters and finally computes the elimination of the exterior of the node cluster pair itself.

Algorithm 3 $U_{\overline{N}}(\varrho N, \varrho N) = \text{ClusterDownwardsPass}(N)$

if ClusterType(N) = Root **then**
  {If N is the root node cluster, then $\varrho N = \emptyset$}
  $U_{\bar{N}}(\varrho N, \varrho N) = [\,]$ (a 0x0 empty matrix)
else
  {Compute $U_{f(N)}(\varrho N, \varrho N)$, $U_{f(N)}(\varrho N, pe(N))$, $U_{f(N)}(pe(N), pe(N))$, and $U_{f(N)}(pe(N), \varrho N)$ by equation (3).}
  $U_{\overline{N}}(\varrho N, \varrho N) = U_{f(N)}(\varrho N, \varrho N)$
  $\quad - U_{f(N)}(\varrho N, pe(N)) U_{f(N)}(pe(N), pe(N))^{-1} U_{f(N)}(pe(N), \varrho N)$
end if
if ClusterType(N) = Leaf **then**
  {If we have a leaf node cluster, we finally eliminate the adjacent node set and invert the remaining (N, N) block.}
  $A^{-1}(N, N) = (A(N, N) - A(N, \varrho N) U_{\overline{N}}(\varrho N, \varrho N)^{-1} A(\varrho N, N))^{-1}$
else
  {If we have a non-leaf cluster, we pass the results of the computation to the children of N, and we also call PairDownwardsPass on the twin child of N.}
  $U_{\overline{M_1}}(\varrho M_1, \varrho M_1) = \text{ClusterDownwardsPass}(M_1)$
  $U_{\overline{M_2}}(\varrho M_2, \varrho M_2) = \text{ClusterDownwardsPass}(M_2)$
  $U_{\overline{\{M_1,M_2\}}}(\varrho\{M_1, M_2\}, \varrho\{M_1, M_2\}) = \text{PairDownwardsPass}(\{M_1, M_2\})$
end if
return $U_{\overline{N}}(\varrho N, \varrho N)$

DOI: 10.1002/nla

Algorithm 4 describes the PairDownwardsPass algorithm in full detail. At a general recursive step for a node cluster $\{N_1, N_2\}$, $U_{\overline{\{P_1,P_2\}}}$ has already been computed for its parent cluster pair $\{P_1, P_2\}$. Also, U_{S_1} and U_{S_2} can be obtained from the upwards pass for the sibling cluster S_1 of N_1 and the sibling cluster S_2 of N_2. The algorithm then computes $U_{f(\{N_1,N_2\})}$, the result of eliminating both the exterior of the parent pair and the interiors of the siblings. (Here, $f(\{N_1, N_2\})$ refers to the 'family' of $\{N_1, N_2\}$.)

Similar to the computation of $U_{f(N)}$, eliminating the exterior of $\{P_1, P_2\}$ will only affect the $(\varrho\{P_1, P_2\}, \varrho\{P_1, P_2\})$ block of $\mathbf{A}$. Also, eliminating the interior of S_1 (S_2) will only affect the $(\partial S_1, \partial S_1)$ block ($(\partial S_2, \partial S_2)$ block). These processes do not interact. So, if n_1 and n_2 are two nodes from $\varrho\{P_1, P_2\}$, then $U_{f(\{N_1,N_2\})}(n_1, n_2) = U_{\overline{\{P_1,P_2\}}}(n_1, n_2)$. Similarly, if n_1 and n_2 both come from ∂S_1 (∂S_2), then $U_{f(\{N_1,N_2\})}(n_1, n_2) = U_{S_1}(n_1, n_2)$ ($U_{f(\{N_1,N_2\})}(n_1, n_2) = U_{S_2}(n_1, n_2)$). In all other cases, $U_{f(\{N_1,N_2\})}(n_1, n_2) = A(n_1, n_2)$.

Algorithm 4 $U_{\overline{\{N_1,N_2\}}}(\varrho N_P, \varrho N_P)$=PairDownwardsPass($\{N_1, N_2\}$)

if PairType($\{N_1, N_2\}$) = Twin OR LeafTwin **then**
  $U_{\overline{\{N_1,N_2\}}}(\varrho\{N_1, N_2\}, \varrho\{N_1, N_2\}) = U_{\overline{P}}(\varrho P, \varrho P)$ (P=parent of $\{N_1, N_2\}$)
else
  {Compute $U_{f(\{N_1,N_2\})}(Q_1, Q_1)$, $U_{f(\{N_1,N_2\})}(Q_1, Q_5)$, $U_{f(\{N_1,N_2\})}(Q_5, Q_5)$, and $U_{f(\{N_1,N_2\})}(Q_5, Q_1)$ by equation (4).}
  $U_{\overline{\{N_1,N_2\}}}(Q_1, Q_1) = U_{f(\{N_1,N_2\})}(Q_1, Q_1)$
    $-U_{f(\{N_1,N_2\})}(Q_1, Q_5)U_{f(\{N_1,N_2\})}(Q_5, Q_5)^{-1}U_{f(\{N_1,N_2\})}(Q_5, Q_1)$
end if
if PairType($\{N_1, N_2\}$) = Leaf OR LeafTwin **then**
  {If we have a leaf cluster pair, we finally eliminate the adjacent node set and invert the remaining $(N_1 \sqcup N_2) \times (N_1 \sqcup N_2)$ block and extract the off-diagonal blocks with $N_P = \{N_1, N_2\}$.}
  $A^{-1}(\{N_1, N_2\}, \{N_1, N_2\}) = (A(N_P, N_P) -$
    $A(N_P, \varrho N_P)U_{\overline{N_P}}(\varrho N_P, \varrho N_P)^{-1}A(\varrho N_P, N_P))^{-1}$
else
  {$\{N_1, N_2\}$ is assumed to have a minimum separation less than or equal to l. We will only call PairDownwardsPass on the children of $\{N_1, N_2\}$ that have a minimum separation less than or equal to l.}
  if $d(M_{11}, M_{21}) \le l$ **then**
    $U_{\overline{\{M_{11},M_{21}\}}}(\varrho M_{P11}, \varrho M_{P11})$ = PairDownwardsPass($\{M_{11}, M_{21}\}$)
    ($\varrho M_{P11} = \varrho\{M_{11}, M_{21}\}$)
  end if
  if $d(M_{11}, M_{22}) \le l$ **then**
    $U_{\overline{\{M_{11},M_{22}\}}}(\varrho M_{P12}, \varrho M_{P12})$ = PairDownwardsPass($\{M_{11}, M_{22}\}$)
    ($\varrho M_{P12} = \varrho\{M_{11}, M_{22}\}$)
  end if
  if $d(M_{12}, M_{21}) \le l$ **then**
    $U_{\overline{\{M_{12},M_{21}\}}}(\varrho M_{P21}, \varrho M_{P21})$ = PairDownwardsPass($\{M_{12}, M_{21}\}$)
    ($\varrho M_{P21} = \varrho\{M_{12}, M_{21}\}$)
  end if
  if $d(M_{12}, M_{22}) \le l$ **then**
    $U_{\overline{\{M_{12},M_{22}\}}}(\varrho M_{P22}, \varrho M_{P22})$ = PairDownwardsPass($\{M_{12}, M_{22}\}$)
    ($\varrho M_{P22} = \varrho\{M_{12}, M_{22}\}$)
  end if
end if
return $U_{\overline{\{N_1,N_2\}}}(\varrho\{N_1, N_2\}, \varrho\{N_1, N_2\})$

DOI: 10.1002/nla

To concisely express the formulas, we define the following sets:

$Q_1 = \varrho\{N_1, N_2\}$, $Q_2 = \varrho\{N_1, N_2\} \cap \overline{\{P_1, P_2\}}$, $Q_3 = \varrho\{N_1, N_2\} \cap S_1$, $Q_4 = \varrho\{N_1, N_2\} \cap S_2$, $Q_5 = pe(\{N_1, N_2\})$, $Q_6 = pe(\{N_1, N_2\}) \cap \overline{\{P_1, P_2\}}$, $Q_7 = pe(\{N_1, N_2\}) \cap S_1$, $Q_8 = pe(\{N_1, N_2\}) \cap S_2$.

Index the nodes in $\overline{\{P_1, P_2\}}$ first, the nodes in S_1 second, and then followed by the nodes in S_2. Using the aforementioned formulas for $U_{f(\{N_1,N_2\})}$, we have

$$
\begin{aligned}
U_{f(\{N_1,N_2\})}(Q_1, Q_1) &= \begin{bmatrix} U_{\overline{\{P_1,P_2\}}}(Q_2, Q_2) & A(Q_2, Q_3) & A(Q_2, Q_4) \\ A(Q_3, Q_2) & U_{S_1}(Q_3, Q_3) & A(Q_3, Q_4) \\ A(Q_4, Q_2) & A(Q_4, Q_3) & U_{S_2}(Q_4, Q_4) \end{bmatrix}, \\
U_{f(\{N_1,N_2\})}(Q_1, Q_5) &= \begin{bmatrix} U_{\overline{\{P_1,P_2\}}}(Q_2, Q_6) & A(Q_2, Q_7) & A(Q_2, Q_8) \\ A(Q_3, Q_6) & U_{S_1}(Q_3, Q_7) & A(Q_3, Q_8) \\ A(Q_4, Q_6) & A(Q_4, Q_7) & U_{S_2}(Q_4, Q_8) \end{bmatrix}, \\
U_{f(\{N_1,N_2\})}(Q_5, Q_5) &= \begin{bmatrix} U_{\overline{\{P_1,P_2\}}}(Q_6, Q_6) & A(Q_6, Q_7) & A(Q_6, Q_8) \\ A(Q_7, Q_6) & U_{S_1}(Q_7, Q_7) & A(Q_7, Q_8) \\ A(Q_8, Q_6) & A(Q_8, Q_7) & U_{S_2}(Q_8, Q_8) \end{bmatrix}, \\
U_{f(\{N_1,N_2\})}(Q_5, Q_1) &= \begin{bmatrix} U_{\overline{\{P_1,P_2\}}}(Q_6, Q_2) & A(Q_6, Q_3) & A(Q_6, Q_4) \\ A(Q_7, Q_2) & U_{S_1}(Q_7, Q_3) & A(Q_7, Q_4) \\ A(Q_8, Q_2) & A(Q_8, Q_3) & U_{S_2}(Q_8, Q_4) \end{bmatrix}.
\end{aligned} \tag{4}
$$

We then compute $U_{\overline{\{N_1,N_2\}}}(Q_1, Q_1)$ by eliminating the private exterior $pe(\{N_1, N_2\})$ nodes, which results in the Schur complement:

$$U_{\overline{\{N_1,N_2\}}}(Q_1, Q_1) = U_{1,1} - U_{1,5}U_{5,5}^{-1}U_{5,1},$$

where $U_{1,1} = U_{f(\{N_1,N_2\})}(Q_1, Q_1)$, $U_{1,5} = U_{f(\{N_1,N_2\})}(Q_1, Q_5)$, $U_{5,5} = U_{f(\{N_1,N_2\})}(Q_5, Q_5)$, and $U_{5,1} = U_{f(\{N_1,N_2\})}(Q_5, Q_1)$.

If it is a nonleaf cluster pair, then the procedure is repeated recursively for its four children cluster pairs: $\{M_{11}, M_{21}\}$, $\{M_{11}, M_{22}\}$, $\{M_{12}, M_{21}\}$, and $\{M_{12}, M_{22}\}$, where $\{M_{11}, M_{12}\}$ and $\{M_{21}, M_{22}\}$ are the children of N_1 and N_2, respectively. We remark that we only perform the computation if the minimum separation of the cluster pair is less than or equal to l. If it is a leaf cluster pair, we have eliminated most of the exterior nodes except the adjacent node set. After eliminating the adjacent nodes, we can invert the remaining $(N_1 \sqcup N_2) \times (N_1 \sqcup N_2)$ block to extract the off-diagonal entries:

$$
\begin{aligned}
&A^{-1}(\{N_1, N_2\}, \{N_1, N_2\}) = \\
&\qquad (A(N_P, N_P) - A(N_P, \varrho N_P)U_{\overline{N_P}}(\varrho N_P, \varrho N_P)^{-1}A(\varrho N_P, N_P))^{-1},
\end{aligned}
$$

where $N_P = \{N_1, N_2\}$ and $\varrho N_P = \varrho\{N_1, N_2\}$.

4. COMPLEXITY ANALYSIS

In this section, the time complexity of the upwards, the downwards, and the advanced downwards passes will be analyzed for a square computational mesh with area α (by area, we mean the number of mesh nodes). We assumed that the node cluster tree is perfectly balanced. Thus, the number of levels in the node cluster tree is $\log_2$. Each node n in the mesh will be assumed to be only connected to nodes that are in the immediate vicinity of n. This assumption means that the number of neighbors each mesh node has is $O(1)$.

Given a node cluster N with area β, let L be of the same order as the length of the node cluster's perimeter. We will call L the 'characteristic' length of node cluster N. As will become apparent later, it will be desirable to keep L to a minimum. L is minimized with order $\sqrt{\beta}$ when N is square or near-square, so an assumption that we will make is that every node cluster is a square or at least

DOI: 10.1002/nla

a near-square rectangle. Thus, nonleaf node clusters are divided into their child clusters by a cut parallel to their shorter dimension.

During the upwards, downwards, and the advanced downwards passes, we perform the following computation at each node cluster/pair: Start with the block matrix $\begin{bmatrix} X & Y \\ Z & W \end{bmatrix}$, where X is $\mu \times \mu$, Y is $\mu \times \nu$, Z is $\nu \times \mu$, and W is a $\nu \times \nu$. We then block reduce to obtain $\begin{bmatrix} X & Y \\ 0 & W - ZX^{-1}Y \end{bmatrix}$. Efficiently computing $U = W - ZX^{-1}Y$ will require $\mu\nu^2 + \frac{1}{3}\mu^3 + \mu^2\nu$ multiplication flops, neglecting lower-order terms.

- When we perform the upwards pass on nonleaf node clusters N, $\mu = |pi(N)|$, and $\nu = |\partial N|$.
- When we perform the downwards pass on nonroot node clusters N, $\mu = |pe(N)|$, and $\nu = |\varrho N|$.
- When we perform the advanced downwards pass on a nontwin node cluster pair $\{N_1, N_2\}$, $\mu = |pe(\{N_1, N_2\})|$, and $\nu = |\varrho\{N_1, N_2\}|$.

In all these cases, μ and ν have order L. Therefore, the complexity of computing $U = O(L^3) = O(\beta^{3/2})$.

We now label the root level of the extended node cluster tree as Level 0 and the subsequent levels as Level 1, 2, etc. At Level i, there are exactly 2^i node clusters. Although there are approximately 4^i node cluster pairs at level i, we only need to consider the pairs whose minimum separation is less than or equal to l. The following will give an informal geometric argument to determine the order of the number of pairs at level i whose minimum separation is less or equal to l.

Let N be any node cluster at level i. Denote the number of level i node clusters other than N, whose minimum separation from N is less than or equal to l by $k(N)$. The number of level i pairs whose minimum separation is less than or equal to l is then given by $\frac{1}{2}\sum k(N)$. If we simply find the magnitude order of $k(N)$, we can multiply it by 2^i to obtain the order of the number of level i pairs whose minimum separation is less than or equal to l.

We now derive the magnitude order of the number of level i pairs for the following two cases:

Case 1: $0 \leqslant i \leqslant \log_2 \frac{\alpha}{l^2}$

In this case, the characteristic length $L = \sqrt{\frac{\alpha}{2^i}}$ of N satisfies $L \geqslant l$. $L \geqslant l$ implies that the number of level i node clusters whose minimum separation from N is less than or equal to l is $O(1)$ because there are only $O(1)$ node clusters around N because of a relatively large size. Figure 8 (left) shows an example when $L > l$. In this case, only eight neighboring node clusters are within l of N. Hence, the number of pairs at level i whose minimum separation is less than or equal to l is $O(2^i)$.

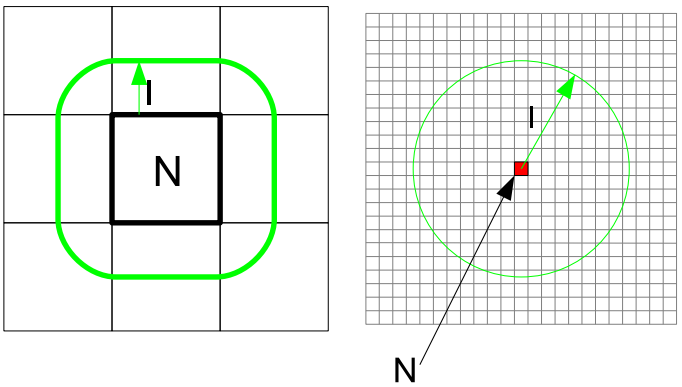

Figure 8. Number of node clusters whose minimum separation from $N \leqslant l$ for (left) large node clusters ($L \geqslant l$) and (right) small node clusters ($L \leqslant l$).

DOI: 10.1002/nla

Case 2: $\log_2 \frac{\alpha}{l^2} \leqslant i \leqslant \log_2 \alpha$

In this case, the characteristic length $L = \sqrt{\frac{\alpha}{2^i}}$ of N satisfies $L \leqslant l$. $L \leqslant l$ implies that the number of level i node clusters whose minimum separation from N is less than or equal to l is proportional to the area of a ball of radius l and inversely proportional to the area of each cluster. Figure 8 (right) shows how small node clusters are covered by a ball of radius l. Because the area of a ball of radius l is $O(l^2)$, the number of level i node clusters that fill this ball is thus $O(\frac{2^i l^2}{\alpha})$. Hence, the number of pairs at level i whose minimum separation is less than or equal to l is $O(\frac{4^i l^2}{\alpha})$.

Combining the two cases,

$$k(N) = \begin{cases} O(2^i) & \text{if } 0 \leqslant i \leqslant \log_2 \frac{\alpha}{l^2} \\ O\left(\frac{4^i l^2}{\alpha}\right) & \text{if } \log_2 \frac{\alpha}{l^2} \leqslant i \leqslant \log_2 \alpha \end{cases}.$$

During the upwards pass, the work per node cluster at level i is $O(L^3) = O(\beta^{3/2}) = O\left(\left(\frac{\alpha}{2^i}\right)^{3/2}\right)$. The total work at level i is $O\left(2^i \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) = O\left(\frac{\alpha^{3/2}}{2^{i/2}}\right)$. The total time complexity of the upwards pass is

$$\begin{aligned} O\left(\sum_{i=0}^{\log_2 \alpha} \frac{\alpha^{3/2}}{2^{i/2}}\right) &= O\left(\alpha^{3/2} \frac{1 - 2^{-\frac{1}{2}(\log_2 \alpha + 1)}}{1 - 2^{-1/2}}\right) \\ &= O\left(\alpha^{3/2} \frac{\sqrt{2} - \alpha^{-1/2}}{\sqrt{2} - 1}\right) \\ &= O(\alpha^{3/2}). \end{aligned}$$

Hence, the time complexity of the upwards and, by a similar argument, the downwards passes is $O(\alpha^{3/2})$.

We will now determine the time complexity of the advanced downwards pass. As with the upwards and downwards passes, the work per node cluster and node cluster pair at level i is still $O(L^3) = O(\beta^{3/2}) = O\left(\left(\frac{\alpha}{2^i}\right)^{3/2}\right)$. The total work at level i is

for $0 \leqslant i \leqslant \log_2 \frac{\alpha}{l^2}$:

$$\begin{aligned} O\left((\text{clusters} + \text{pairs}) \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) &= O\left((2^i + 2^i) \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) \\ &= O\left(2^i \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) \\ &= O\left(\frac{\alpha^{3/2}}{2^{i/2}}\right). \end{aligned}$$

for $\log_2 \frac{\alpha}{l^2} \leqslant i \leqslant \log_2 \alpha$:

$$\begin{aligned} O\left((\text{clusters} + \text{pairs}) \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) &= O\left(\left(2^i + \frac{4^i l^2}{\alpha}\right) \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) \\ &= O\left(\frac{4^i l^2}{\alpha} \times \left(\frac{\alpha}{2^i}\right)^{3/2}\right) \\ &= O\left(2^{i/2} \alpha^{1/2} l^2\right). \end{aligned}$$

DOI: 10.1002/nla

The total time complexity of the advanced downwards pass is

$$
\begin{aligned}
&O\left(\sum_{i=0}^{\log_2 \frac{\alpha}{l^2}} \frac{\alpha^{3/2}}{2^{i/2}} + \sum_{i=\log_2 \frac{\alpha}{l^2}}^{\log_2 \alpha} 2^{i/2}\alpha^{1/2}l^2\right)\\
&= O\left(\alpha^{3/2}\frac{1-2^{-\frac{1}{2}(\log_2 \frac{\alpha}{l^2}+1)}}{1-2^{-1/2}} + 2^{\frac{1}{2}\log_2 \frac{\alpha}{l^2}}\alpha^{1/2}l^2\frac{2^{\frac{1}{2}(\log_2 l^2+1)}-1}{2^{1/2}-1}\right)\\
&= O\left(\alpha^{3/2}\frac{\sqrt{2}-\alpha^{-\frac{1}{2}}l}{\sqrt{2}-1} + \alpha l\frac{l\sqrt{2}-1}{\sqrt{2}-1}\right)\\
&= O(\alpha^{3/2}+\alpha l^2).
\end{aligned}
$$

Hence, the time complexity of the advanced downwards pass is $O(\alpha^{3/2}+\alpha l^2)$. Note that if $l = 0$, which means that we are computing no off-diagonal entries, then the time complexity is $O(\alpha^{3/2})$ which is the same as the upwards and downwards passes. If $l = \sqrt{\alpha}$, which means that we are computing *all* off-diagonal entries, then the time complexity is $O(\alpha^2)$ which is the minimum time complexity required to compute *every* entry of the inverse.

The highest order for cut-off distance l that will still result in a time complexity of $O(\alpha^{3/2})$ for the advanced downwards pass is $l = O(\alpha^{1/4})$. In other words, l must have, at most, the same order as the square root of the computational mesh's width if the time complexity is to remain at $O(\alpha^{3/2})$.

Given a cut-off distance of l, it is of interest to note that the number of entries of A^{-1} that will ultimately be computed is of the same order as the number of leaf pairs whose minimum separation is less than or equal to l. The number of leaf pairs whose minimum separation is less than or equal to l is $O(\frac{4^{\log_2 \alpha} l^2}{\alpha}) = O(\frac{\alpha^2 l^2}{\alpha}) = O(\alpha l^2)$ (Here, we used the case 2 result derived when computing the number of level i pairs whose minimum separation is less than or equal to l). If we let the cut-off distance $l = \alpha^{1/4+\epsilon}$ where $0 \leqslant \epsilon \leqslant 1/4$, then we will be computing $O(\alpha^{3/2+2\epsilon})$ entries of A^{-1}. Because the computational time complexity in this case is also $O(\alpha^{3/2+2\epsilon})$, the computational time complexity per entry of A^{-1} calculated is $O(1)$. This concludes the result stated in the abstract.

5. NUMERICAL RESULTS

In this section, we will demonstrate that the computational complexity of the advanced FIND algorithm is $O(\alpha^{3/2})$, provided that the cut-off distance l is $O(\alpha^{1/4})$. The example scenario that was used in measuring the flop counts is an $L \times L$ square computational mesh utilizing a five-point stencil. The sparse matrix used was the five-point stencil approximation of the Laplacian operator. The node cluster tree depth is chosen to be deep enough so that all leaf node clusters have either 0 or 1 nodes. More precisely, the node cluster tree depth used for an $L \times L$ computational mesh is $2\lceil \log_2(L) \rceil$. The cut-off distance l used for an $L \times L$ computational mesh is $l = \lfloor \sqrt{L} \rfloor$.

Figure 9 is a base 2 log-log plot of the multiplication flops used versus the grid size L. The upper curve (interpolated from the points denoted by squares) shows the number of flops used in the upwards pass and advanced downwards pass, that is, the advanced FIND algorithm. The lower curve (interpolated from the points denoted by diamonds), included for reference, shows the number of flops used in the upwards pass and downwards pass, that is, the original FIND algorithm. As we are trying to show that the computational complexity is $O(\alpha^{3/2}) = O(L^3)$, the upper curve must tend towards having a slope of 3. The straight line included indicates a slope of 3. As can be seen, both the upper and the lower curves become increasingly parallel to the straight line.

Notice that the upper curve in Figure 9 has a number of apparent jumps: between grid sizes 8 and 9; between grid sizes 15 and 16; between grid sizes 24 and 25; and between grid sizes 35 and 36. Recalling that the cut-off distance $l = \lfloor \sqrt{L} \rfloor$, the jumps correspond exactly when l increases by 1.

Figure 10 is a base 2 log-log plot of run time versus the grid size. Again, the upper curve denotes the advanced FIND algorithm and the lower curve denotes the original FIND algorithm. The straight

DOI: 10.1002/nla

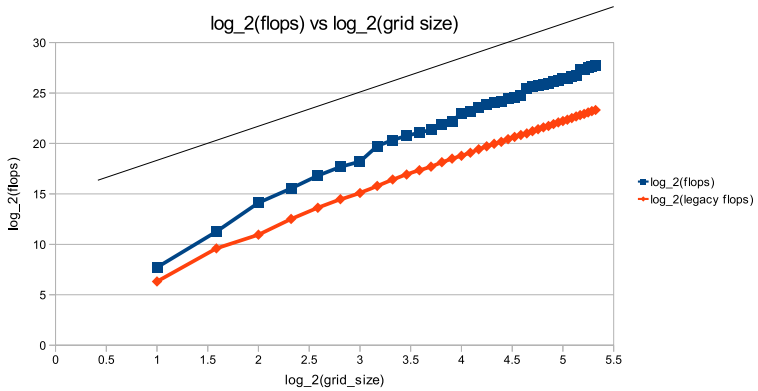

Figure 9. A base 2 log-log plot of flops versus grid size.

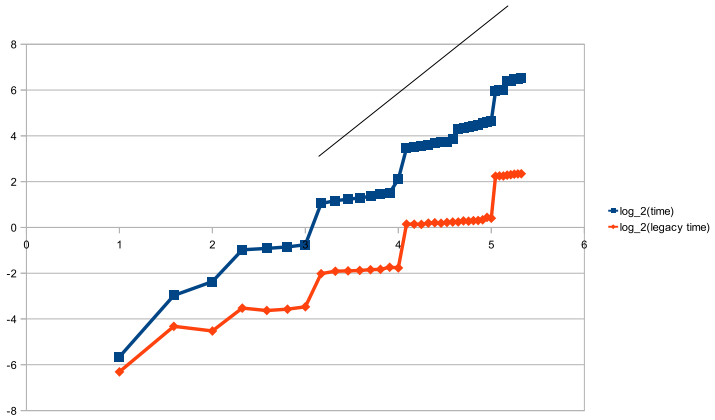

Figure 10. A base 2 log-log plot of CPU time versus grid size.

line included indicates a slope of 3. As can be seen, both the upper and the lower curves are (roughly) parallel to the straight line.

Both the upper and the lower curves in Figure 10 jump dramatically whenever the node cluster tree depth, $2\lceil \log_2(L) \rceil$, increases. This behavior, which is evident in the curves for both the original FIND algorithm and the advanced FIND algorithm, is caused primarily by a dramatic increase in the amount of integer computations that accompany the increase in the node cluster tree size. At each 'step', the depth of the node cluster tree increases by 2, and because the node cluster tree is a binary tree, this means that the number of node clusters has increased by fourfold. The upper curve also exhibits smaller jumps whenever the cut-off distance $l = \lfloor \sqrt{L} \rfloor$ increases. Despite this, both curves still exhibit an overall slope of approximately 3.

6. CONCLUSION

We have extended the algorithm first described in [3] to now calculate off-diagonal entries of the inverse of a large sparse matrix corresponding to a square computational mesh. The application given in the introduction of finding the Green's function for quantum nanodevices motivates the central idea that entries of the inverse that correspond to geometrically closer pairs of vertexes in the computational mesh are more important than entries of the inverse that correspond to more separated pairs of vertexes. Diagonal entries, which correspond to connections of length 0, have the highest priority. The extended algorithm computes all entries of the inverse that correspond to connections shorter than a given cut-off distance l. We proved that if l is of the order of $O(\alpha^{1/4+\epsilon})$, where α is the number of nodes in the square mesh and where $0 \leqslant \epsilon \leqslant 1/4$ is arbitrary, then the computational

DOI: 10.1002/nla

time complexity is $O(\alpha^{3/2+2\epsilon})$ and the number of inverse entries computed is also $O(\alpha^{3/2+2\epsilon})$, resulting in a constant amount time per entry computed. For $\epsilon = 0$, we obtain $O(\alpha^{3/2})$ additional off-diagonal entries while not changing the time complexity of the FIND algorithm given by [3] which is $O(\alpha^{3/2})$.

REFERENCES

1. George A. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis* 1973; **10**(2):345–363.
2. George A, Liu, Joseph W H. *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs: N.J., Prentice-Hall, 1981.
3. Li S, Ahmed S, Klimeck G, Darve E. Computing entries of the inverse of a sparse matrix using the FIND algorithm. *Journal of Computational Physics* 2008; **227**:9408–9427.
4. Erisman AM, Tinney WF. On computing certain elements of the inverse of a sparse matrix. *Communications of the ACM* 1975; **18**(3):177–179.
5. Darve E, *et al.* A hybrid method for the parallel computation of Green's functions. *Journal of Computational Physics* 2009; **228**:5020–5039.
6. Lin L, Lu J, Ying L, Car R, E W. Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems. *Communications in Mathematical Sciences* 2009; **7**(3):755–777.
7. Svizhenko A, Anantram MP, Govindan TR, Biegel B, Venugopal R. Two-dimensional quantum mechanical modeling of nanotransistors. *Journal of Applied Physics* 2002; **91**(4):2343–2354.
8. Anantram MP, Svizhenko A. Multidimensional modeling of nanotransistors. *IEEE transactions on Electron Devices* 2007; **54**(9):2100–2115.

DOI: 10.1002/nla