EXTENDING HIERARCHICAL PROBING FOR COMPUTING THE TRACE OF MATRIX INVERSES

JESSE LAEUCHLI* AND ANDREAS STATHOPOULOS†

Abstract. We present extensions to Hierarchical Probing, a method developed in [13] to reduce the variance of the Monte Carlo estimation of the trace or the diagonal of the inverse of a large, sparse matrix. In that context, probing is a method to determine the largest in magnitude elements of the matrix inverse and then annihilate their contributions to the variance by solving linear systems with appropriate probing vectors. It typically involves coloring the graph of A^n, since this matches the sparsity structure of a polynomial approximation to A^{-1}. This is equivalent to distance-n coloring of A, i.e., determining which nodes are connected to each other at distance $\leq n$. For matrices that display a Green's function decay, n is small, which reduces the number of linear systems to be solved.

Our Hierarchical Probing method was developed for matrices with a lattice structure, where distance-n coloring and the generation of probing vectors can be performed far more efficiently and in a way so that earlier vectors are subsets of vectors generated later in the process, meaning that it is simple to continue probing if additional accuracy is needed. However this method worked only on lattices with dimension lengths that were powers of two. In this paper we extend the method to work on lattices of arbitrary dimension lengths which is theoretically more challenging. Additionally, we expand the idea to a multilevel, hierarchical probing heuristic for matrices with any undirected graph structure that matches the performance of classical probing but with tractable memory requirements.

Key words. Probing, trace of the inverse, sparse matrix, torus, Hadamard, Fourier, Lattice QCD

AMS subject classifications. 65F15, 05B20, 81V05, 65C05, 65F50

1. Introduction and Preliminaries.

1.1. Introduction.

One important but computationally difficult problem in numerical linear algebra is the estimation of the trace or the diagonal of a function $f(A)$ of a large sparse matrix, A. In this paper we focus on $f(A) = A^{-1}$. This has applications in many areas, including Statistics [10, 12], Network Analysis [6], and Quantum Chromodynamics [5]. In addition, we limit our discussion to matrices with symmetric structure, although a generalization can be devised as in [2]. Initially, approaches to this problem were statistical in nature [10, 1] and did not take advantage of any knowledge of the structure of the matrix. If such knowledge can be obtained, some methods have been developed to take advantage of it, leading to better estimations [14, 3, 13]. We are interested in computing the trace of A^{-1} but the same techniques are used to estimate the diagonal.

An attempt to exploit less regularly ordered structure is behind the idea of probing [14]. Probing recovers the diagonal elements of a matrix by finding a coloring of its associated graph. Coloring an undirected graph involves assigning a color to each vertex in such a way that no two connected vertices share a color. When the rows and columns of a matrix are permuted so that all nodes that share the same color are adjacent, a series of diagonal blocks will appear along the diagonal, since nodes that share a color have no connection. An example of this is shown on the left graphic of Fig. 1. Given a coloring for a symmetric matrix A, with c total colors, we generate a probing vector for each color $m = 1, \ldots, c$ by setting its i-th element to 1 if the i-th

*School of Information Technology, Deakin University, Geelong, Victoria 3220, Australia (j.laeuchli@deakin.edu.au)

†Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23187-8795, U.S.A.(andreas@cs.wm.edu)

node of the graph of A is assigned the m-th color,

$$x_i^m = \begin{cases} 1, & \text{if } color(i) = m \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

We can use the probing vectors to recover the diagonal and trace of the matrix by c matrix vector multiplications, $diag(A) = \sum_{m=1}^{c} x^m \odot Ax^m$ where $\odot$ denotes the element-wise product, and $\mathbf{Tr}(A) = \sum_{m=1}^{c} x^{mT} Ax^m$.

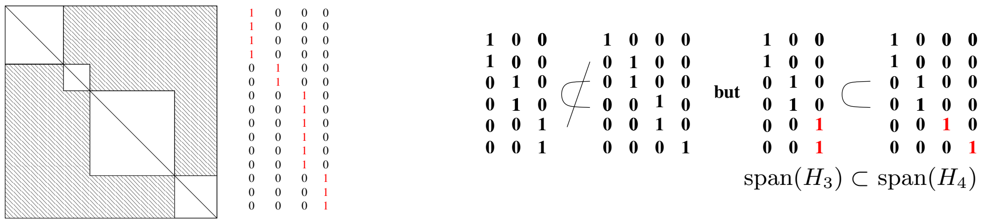


Fig. 1: Probing examples. Left, on a sparse matrix. Right, when approximating $A^{-1} \approx p_n(A)$, the probing vectors for $n = n_1$ may not be probing vectors for $n > n_1$.

Unfortunately, A^{-1} is normally dense, and coloring its associated graph requires a unique color for every node. To facilitate probing, we must induce sparsity structure for A^{-1} by sparsification. For example, we could drop the smallest magnitude elements of A^{-1}, and use probing on the resulting matrix. However, this is impractical.

Tang and Saad [14] introduce the idea of probing the non-zero structure of a matrix polynomial $p_n(A)$ of degree n, such that $p_n(A) \approx f(A)$, where $f(A)$ is some matrix function of A. While this could be any function that is well approximated by a matrix polynomial, they focus on the analysis of $f(A) = A^{-1}$. For this case they use the Neumann approximation $p_n(A) = M^{-1} \sum_{j=0}^{n} (M^{-1}N)^j$, where $A = M - N$, and $M = diag(A)$, to identify a degree n for which the approximation is sufficient. Because the nonzero structure of $p_n(A)$ is a subset of the nonzero structure of $p_{n+1}(A)$, we can simply color the graph of A^n and generate the corresponding probing vectors, x^m. Notice that coloring A^n is equivalent to producing a distance-n coloring of the graph of A. The approximation is then $diag(A^{-1}) \approx \sum_{m=1}^{c} x^m \odot A^{-1}x^m$, where $A^{-1}x^m$ is solved approximately through an iterative method. This sparsification idea works when the non-zeros introduced in higher degree polynomials reduce in magnitude. This is observed in many problems, for example certain PDEs where the elements $A^{-1}_{i,j}$ decay exponentially with the graph distance between nodes i and j [4].

Probing as described above has two main drawbacks. First, finding a distance-n coloring of A is difficult for higher distances of n. Some efficient algorithms exist for distances of $n = 1, n = 2$ such as those designed for Jacobian and Hessian computations [8, 7]. The algorithm of [8] even targets matrices with regular structures, such as 2D lattices, in a process similar to our own algorithm. However, these methods do not extend to arbitrary distances. Unfortunately, for many matrices low distance colorings are not sufficient to reduce sufficiently the variance of the trace estimator.

Second, when the estimate produced by probing for a particular n is not sufficiently accurate, higher degrees need to be tried. However, it is unlikely that the probing vectors of $p_n(A)$ will be spanned by those created for $p_k(A)$, $k > n$, see for example the right part of Fig. 1. Therefore, all previous work for solving linear systems with the n probing vectors will have to be discarded and k linear systems must be solved for a different set of probing vectors. These problems are a bottleneck for large matrices.

1.2. Hierarchical Probing. Our goal is to create probing vectors without explicitly computing A^n and in such a way that the subspace of probing vectors for A^n contains the probing vectors for lower powers of n, as in the right part of Fig. 1. We term such vectors hierarchical probing vectors. The main building block needed to create such vectors is a hierarchical coloring. This is a series of colorings for successive powers of n, such that two nodes that previously had different colors at A^n must never later share a color at higher values of n. Fig. 2 shows an example of both a hierarchical and a non-hierarchical coloring.

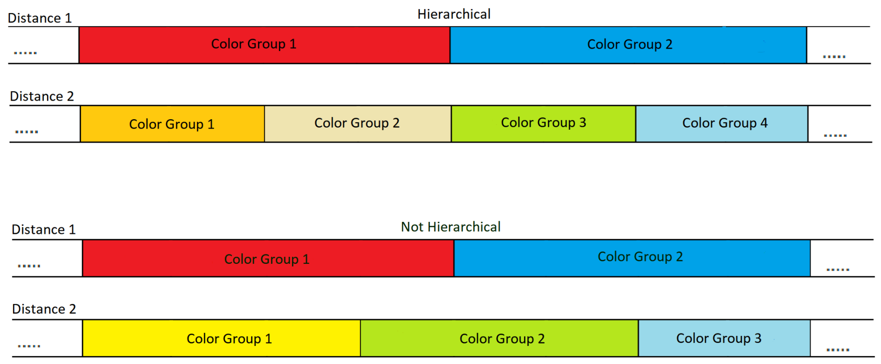


Fig. 2: Hierarchical vs non-hierarchical coloring. In the example at the top we have two color groups which at distance-2 split cleanly so that none of the nodes originally placed in group 1 later share a color with nodes that were in group 2. In the example at the bottom, some nodes from group 1 and group 2 share a color in group 2 after the distance-2 coloring. Although a valid coloring, it is no longer hierarchical.

Our previous work in [13] provided an efficient solution to this problem of creating hierarchical probing vectors when the structure of A is a toroidal lattice (i.e., a lattice of which each dimension is a torus). Such matrices occur in Lattice Quantum Chromodynamics (LQCD) and also in some finite difference applications. On lattices, it is not necessary to compute A^n to find a distance-n coloring since this can be derived implicitly from the positions of nodes in the lattice. For distance n=1 and n=2 this was also observed for 2D lattices in [8]. However, extending their results to higher dimensions and larger n proves quite difficult. More importantly, the optimal coloring for a certain distance may not yield nested colorings for larger distances. Taking advantage of the regularity of lattices, our algorithm in [13] produced such a hierarchical coloring for d-dimensional lattices when the length of each dimension was a power of two. This property of the lengths allowed us to split each lattice into 2^d conformal sublattices, where points in different sublattices are all at least distance-2 away from each other. Therefore, if all points in a sublattice are assigned the same color, we obtain a valid distance-1 coloring of the original lattice. The process is repeated recursively on each sublattice with nodes in different sublattices never sharing the same color at subsequent levels. Thus, at level i the number of colors is 2^{di}, guaranteeing at least a distance-2^i coloring. In addition, between levels i and $i+1$, we give the nodes of each sublattice an appropriate intermediate coloring in case we want to stop with less than $2^{d(i+1)}$ colors.

The hierarchical coloring could be used with (1) to produce the corresponding probing vectors. However, this creates the problem shown in the right part of Fig. 1. Although the subspaces of H_3 and H_4 are nested, the individual basis vectors are not. Thus, despite having computed $A^{-1}H_3$ when probed with H_3, to probe with H_4 we need to solve two additional systems, $A^{-1}H_4(:, 3:4)$, not one. The solution is to consider nested bases of these subspaces. For example adding the vector $[0\ 0\ 1\ \text{-}1]^T$ in H_3 would yield a basis for span(H_4). The structure of our coloring algorithm allows for an efficient generation of a hierarchical basis using appropriate permutations of columns and rows of the Hadamard matrix. Then a trace computation starts with the probing vectors from the distance-1 coloring, continues with the additional probing vectors required to complete the distance-2 coloring, and so on until the required accuracy on the trace is achieved. We term our algorithm Hierarchical Probing (HP),

Our HP algorithm provided speedups of one order of magnitude in trace computations in LQCD, but it was limited to lattices whose dimensions had power of two lengths; a requirement for both coloring and the generation of the Hadamard basis.

In this paper we first extend the original HP algorithm and the associated theory to sublattices of arbitrary dimensions, as long as the lengths of the dimensions share some common prime factors. Second, we extend these ideas to general sparse graphs by performing coloring on a sequence of hierarchically smaller graphs, obtained by aggregating neighboring nodes and merging distance-2 neighborhoods at each level.

To ensure a hierarchical basis, our methods produce more colors than the optimal coloring at every level. As a side benefit, these additional vectors reduce the error further than the optimal coloring for the same distance. Our experiments show that, in both cases, the variance reduction for our trace estimation method is close to classical probing but with significant savings in memory and computational time.

2. Hierarchical Probing on Lattices.

After introducing the notation used for lattices and sublattices, we present the HP method for general lattices.

2.1. Introduction to Lattice Notation.

Formally, a lattice is a discrete additive subgroup of $\mathbb{R}^n$. Intuitively, it is a collection of points, such that adding the location of any points together, gives the coordinates of another valid point, and there is a minimum distance between the closest two points. An example of a d-dimensional lattice would be the Cartesian product of the integers, which is the canonical d-dimensional regular grid. A finite lattice is defined similarly, only on a finite group. Intuitively, a finite lattice is a regular d-dimensional grid of which each dimension is a torus.

Let each of the d dimensions of the lattice have length d_i. Let $\mathbb{Z}_n$ denote the multiplicative group of integers modulo n and define $\mathbb{Z}^D = \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_d}$. Similarly to vector spaces, one can write down the basis for a finite lattice. We focus on regular toroidal grids, so we say the lattice is generated by the basis $\mathbf{I}$ as

$$\mathcal{L}(\mathbf{I}) = \left\{ \sum_{i=1}^{d} x_i * \mathbf{e}_i, \quad \mathbf{e}_i \in \mathbf{I}, x_i \in \mathbb{Z}_{d_i} \right\} = \left\{ \mathbf{Ix}, \quad \mathbf{x} \in \mathbb{Z}^D \right\}, \tag{2}$$

where $\mathbf{I}$ is the $d \times d$ identity matrix and $\mathbf{e}_i$ is the i-th column of $\mathbf{I}$. The multiplication $(*)$ and all arithmetic in the i-th dimension are performed mod d_i. This requirement enforces a minimum distance between two points, in contrast to a vector space.

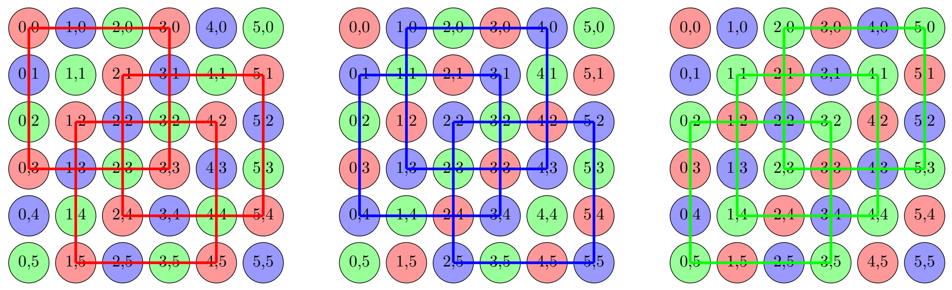
Lattices may contain closed sublattices. We are interested in the sublattices of $\mathcal{L}(\mathbf{I})$ because we will need to determine which sublattices of $\mathcal{L}(\mathbf{I})$ a point lies in. Let

$b \in \mathbb{Z}$ that divides all $d_i, i = 1, \ldots, d$. We define the sublattice $\mathcal{L}(b\mathbf{I})$ as the lattice that uses $b * \mathbf{e}_i$ as the generating basis. Thus, $\mathcal{L}(b\mathbf{I})$ describes $\frac{1}{b^d}$ of the points of $\mathcal{L}(\mathbf{I})$ with spacing b. We can extend the above definition to the more general concept of an affine sublattice as,

$$\mathcal{L}(b\mathbf{I})_{\mathbf{c}} = \left\{ \sum_{i=1}^{d} x_i * b * \mathbf{e}_i + \mathbf{c}, \quad \mathbf{e}_i \in \mathbf{I}, \mathbf{x}, \mathbf{c} \in \mathbb{Z}^D \right\} = \left\{ b\mathbf{x} + \mathbf{c}, \quad \mathbf{x}, \mathbf{c} \in \mathbb{Z}^D \right\}. \quad (3)$$

Clearly $\mathcal{L}(b\mathbf{I})$ is recovered when the offset vector $\mathbf{c}$ is zero. We emphasize again that all coordinate computations in the i-th dimension are performed mod d_i.

We can use these to decompose $\mathcal{L}(\mathbf{I})$ into a union of affine sublattices. For a given sublattice spacing b, any point in $\mathcal{L}(\mathbf{I})$ lies in one of b^d affine sublattices $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$. These sublattices can be said to span $\mathcal{L}(\mathbf{I})$. An example of this can be seen in Fig. 3, where the 6x6 lattice is spanned by 3^2 affine sublattices of spacing 3.



(a) Sublattices with offsets $\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}2\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix}$

(b) Sublattices with offsets $\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix}$

(c) Sublattices with offsets $\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}$

Fig. 3: The decomposition of a 6x6 lattice into 3^2 sublattices $\mathcal{L}(3I)_{\mathbf{c}_0}$.

More formally, given $b \in \mathbb{Z}$ that divides all d_i, the following sublattices span $\mathcal{L}(\mathbf{I})$:

$$\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i} = \mathcal{L}(b\mathbf{I}) + \mathbf{c}_i, \text{ with } \mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \mathbf{c}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \mathbf{c}_{b^d-1} = \begin{pmatrix} b-1 \\ b-1 \\ \vdots \\ b-1 \end{pmatrix}. \quad (4)$$

As there are b distinct options for each of the d elements of an offset $\mathbf{c}$, there are b^d distinct lattice bases that span $\mathcal{L}(\mathbf{I})$. Based on the b-radix representation of integers, we can find a one to one function that maps the integers $0 \leq i \leq b^d - 1$ to each offset vector $\mathbf{c}$, allowing each $\mathbf{c}$ to be associated with a unique sublattice number. The function that maps the offset vector $\mathbf{c}$ to the sublattice number i is

$$i = \sum_{j=1}^{d} c_j b^{j-1}. \quad (5)$$

Its inverse function that maps i to a particular offset $\mathbf{c}$ is computed by Algorithm 1.

This gives the following general equation for the i-th affine sublattice basis

$$\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i} = \mathcal{L}(b\mathbf{I}) + \begin{pmatrix} \lfloor \frac{r_{d-1}}{b^0} \rfloor \\ \vdots \\ \lfloor \frac{r_1}{b^{d-2}} \rfloor \\ \lfloor \frac{i}{b^{d-1}} \rfloor \end{pmatrix}. \tag{6}$$

Algorithm 1 $\mathbf{c} = \text{ConvertIndexToOffset}(i, b, d)$
% Find the affine offset $\mathbf{c}$, given its integer reference number i
% Input: i: integer lattice reference
% Output: The offset vector $\mathbf{c}$

```
1: for $m = d \to 1$ do
2:     $\mathbf{c}(m) \leftarrow \lfloor \frac{i}{b^{m-1}} \rfloor$
3:     $\mathbf{r}_m \leftarrow i (\text{mod } b^{m-1})$
4:     $i \leftarrow \mathbf{r}_m$
5: end for
       return c
```

2.2. Overview of the Algorithm. Our approach is to split the lattice into its spanning b^d sublattices for a particular b and give each sublattice a different color, i.e., all the nodes in a sublattice take the color of that sublattice. In the example of Fig. 3 each of the nine sublattices receives a unique color using (5), color 0 for all four nodes of $\mathcal{L}(3I)_{\binom{0}{0}}$, color 3 for the four nodes of $\mathcal{L}(3I)_{\binom{0}{1}}, \ldots,$ and color 8 for $\mathcal{L}(3I)_{\binom{2}{2}}$. We show that this is a valid distance-$(b-1)$ coloring for the nodes in each sublattice and because each sublattice takes a different color it is a valid distance-$(b-1)$ coloring for the initial lattice. This idea is then applied recursively on each sublattice $\mathcal{L}(bI)_{\mathbf{c}_i}$, for a possibly different b. Thus we obtain a series of progressively higher distance colorings which are naturally hierarchical (as in the example in Fig. 2). Before the completion of each recursive level, we give also an intermediate b-coloring of the lattice. For example in Fig. 3 the lattice is 3-colored (red, blue, green) which respects the hierarchy since all later sublattices involve nodes of the same color.

We organize our presentation for lattices as follows. In Section 2.3 we show how to determine which sublattice a point lies in which will determine its color at each recursion level. In Section 2.4 we show how to assign colors to sublattices and prove its correctness. In Section 2.5, we provide an algorithm for coloring lattices that have dimensions of equal lengths, i.e., lattices that split into the same number of sublattices at every level. In Section 2.6 we extend the algorithm to cover the case of lattices that have dimensions of unequal lengths. Generating the probing vectors from the colorings of these two cases is described in Sections 2.7 and 2.8. For readability, all proofs in this Section 2 are given in the Appendix.

2.3. Finding Sublattice Membership and Building the Hierarchy. Because we plan to assign colors based on sublattice membership, we need to determine which sublattice a point lies in at a given level of the recursive algorithm.

Consider a lattice point with coordinates $\mathbf{x}$ in the regular lattice $\mathcal{L}(I)$ and $b \in \mathbb{Z}$ that divides all d_i. From (4) $\mathcal{L}(I)$ is spanned by the sublattices $\mathcal{L}(bI)_{\mathbf{c}_i}$ so $\mathbf{x}$ belongs in one of these sublattices, say in $\mathcal{L}(bI)_{\mathbf{c}}$, with coordinates $\mathbf{x}'$. This means that

$\mathbf{x} = b\mathbf{x}' + \mathbf{c}$ or $x_i = x_i' * b + c_i,\ i = 1, \ldots, d$, with $x_i, x_i', c_i \in \mathbb{Z}_{d_i}, 0 \le c_i \le (b-1)$. Therefore, we can compute the offset vector $\mathbf{c} = (c_i)$ as $c_i = x_i \bmod b$. The offset $\mathbf{c}$ determines through (5) which sublattice the point lies in.

Let us work out an example in Fig. 3 with $b = 3$. The top left sublattice in (a), consists of the points $(0,0)$,$(0,3)$,$(3,0)$,$(3,3) \equiv (0,0) \pmod 3$. From (5), $i = 0 * b^1 + 0 * b^0 = 0$, indicating that all these points are in the 0-th sublattice. Alternatively, the points $(2,1)$,$(5,1)$,$(2,4)$,$(5,4) \equiv (2,1) \pmod 3$. Since $i = 1*3+2 = 5$, these points are in the 5-th sublattice. Finally, points $(1,2)$,$(1,5)$,$(4,5)$,$(4,2) \equiv (1,2) \pmod 3$, which means these points are in the 7-th sublattice ($i = 2*3+1 = 7$).

We now discuss what b to use for splitting the sublattices in the hierarchy. We remind the reader that $d_i, i = 1, \ldots, d$ are the lengths of each lattice dimension. Let $F_i = \text{factor}(d_i)$ be the multisets of integer factors from the prime factorization of d_i. Then define the list of common factors (where the intersection allows for a factor to appear multiple times) as

$$\mathbf{F} = \text{sort}\left(\bigcap_{i=1}^{d} F_i\right). \tag{7}$$

In the example of Fig. 3, $F_1 = F_2 = \{2,3\}$, and so $\mathbf{F} = \{2,3\}$. More interestingly, for a lattice of dimensions 60×140, $F_1 = \{2,2,3,5\}$, $F_2 = \{2,2,5,7\}$, and $\mathbf{F} = \{2,2,5\}$.

We use the list of common factors $\mathbf{F}$ to split the lattice $\mathcal{L}(\mathbf{I})$ into a hierarchy of spanning sublattices. We start with the smallest $b = \mathbf{F}(1)$ and obtain the sublattices in (4). We start with the smallest member of $\mathbf{F}$ because this will split the lattice into a smaller number of sublattices, which means we will have a smaller number of colors or equivalently probing vectors. This in turn means fewer calls to the iterative solver when estimating $diag(A^{-1})$. This allows us to examine the accuracy of the estimation at earlier stopping points and, if sufficient, avoid using the higher distance colorings.

After we have selected a $b \in \mathbf{F}$ and split our lattice, we continue recursively by splitting every sublattice into its own set of spanning sublattices based on the next common factor $\mathbf{F}(2)$. The process continues until all common factors have been exhausted. Using the fact that for any point $\mathbf{p}$ in a lattice, its sublattice offset after a split is ($\mathbf{p} \bmod b$), Algorithm 2 computes the sublattice offset vectors of $\mathbf{p}$ for all levels. Because of the equivalence between offsets and indices, Algorithm 2 returns only the index of the offsets through (5). Note that after splitting $\mathcal{L}(\mathbf{I})$ with $b = \mathbf{F}(1)$, the $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}$ have common factors $\mathbf{F}(2:end)$, and all have the same size with dimensions d_i/b. The coordinates of $\mathbf{p}$ in its sublattice are $\lfloor \frac{\mathbf{p}}{b} \rfloor$.

Algorithm 2 $[i^{(1)}, \ldots, i^{(f)}] = \text{SublatticeIndicesOfPoint}(\mathbf{p}, \mathbf{F})$
% Determine which sublattice a point lies in at each splitting level
% Input: $\mathbf{p}$ lattice point coordinates
% Input: $\mathbf{F}$ the common primme factors $\mathbf{F}(1) \le \cdots \le \mathbf{F}(f)$
% Output: $i^{(m)}$ the index corresponding to offset $\mathbf{c}^m, m = 1, \ldots, f$ of the m-th split

```
1: for $m = 1 \to size(\mathbf{F})$ do          % At each level use splitting spacing $b = \mathbf{F}(m)$
2:     $\mathbf{c}^m \leftarrow \mathbf{p} (\bmod \mathbf{F}(m))$  % determine $\mathbf{p}$'s affine offset (i.e., sublattice) at level $m$
3:     $i^{(m)} \leftarrow$ Convert $\mathbf{c}^m$ to index through (5)
4:     $\mathbf{p} \leftarrow \lfloor \mathbf{p}/\mathbf{F}(m) \rfloor$                 % point coordinates in this sublattice
5: end for
       return $i^{(m)}$ for all $m$
```

2.4. Coloring sublattices. The Manhattan distance between any two points $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ is $\|\mathbf{p}_1 - \mathbf{p}_2\|_1$. The minimum distance of these points is b, the spacing of the sublattice. More formally, from (3), there exist $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}^D$, such that $\mathbf{p}_1 = b\mathbf{x}_1 + \mathbf{c}, \mathbf{p}_2 = b\mathbf{x}_2 + \mathbf{c}$, and, if $\mathbf{p}_1, \mathbf{p}_2$ are distinct points, $\|\mathbf{p}_1 - \mathbf{p}_2\|_1 = b\|\mathbf{x}_1 - \mathbf{x}_2\|_1 \geq b > 0$. Thus, we may assign the same color in all points in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ and still have a valid distance $b-1$ coloring of the points within $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$.

Next we need to study the effect of coloring across sublattices. The minimum distance between points in two different sublattices is determined by the distance of their offsets. Using (3) again, if $\mathbf{p}_1 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}$ and $\mathbf{p}_2 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_j}$, $\|\mathbf{p}_1 - \mathbf{p}_2\|_1 = \|b(\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{c}_i - \mathbf{c}_j\|_1 \geq \|\mathbf{c}_i - \mathbf{c}_j\|_1$, since we can pick $\mathbf{x}_1 = \mathbf{x}_2$. For example, points $[0, \ldots, 0]^T \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_0}$ and $[1, 0, \ldots, 0]^T \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_1}$ are distance 1 apart. Therefore, if the nodes in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_0}$ and in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_1}$ are all assigned the same color, we cannot achieve a valid distance-1 coloring on the entire $\mathcal{L}(\mathbf{I})$. Thus, we need to decide what color to give to each sublattice.

Since the minimum distance between $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}$ and $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_j}$ is $\|\mathbf{c}_i - \mathbf{c}_j\|_1$, the problem of coloring the sublattices is equivalent to coloring the lattice,

$$\mathcal{C} = \{\mathbf{p} \bmod b,\ \mathbf{p} \in \mathcal{L}(\mathbf{I})\}\,, \tag{8}$$

whose points are the b^d offset vectors $\mathbf{c}_i$ in (4). Note that $\mathcal{C}$ can be used to tile the original lattice as seen in Fig. 4. Different coloring strategies of $\mathcal{C}$ achieve different distances between $\mathbf{c}_i, \mathbf{c}_j$ with the same color, and hence between the points of $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}, \mathcal{L}(b\mathbf{I})_{\mathbf{c}_j}$. More formally we have the following.

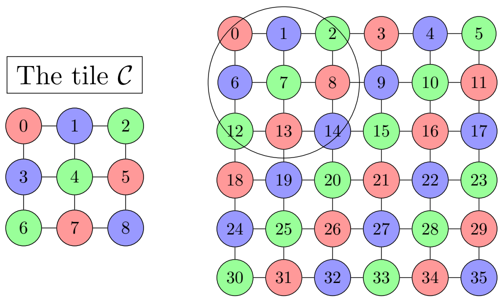


Fig. 4: In the same 2-D example as in Fig. 3 we show the lattice points in their natural ordering. The circled nodes constitute the $\mathcal{C}$ lattice of offsets, which is redrawn on the left with the offset numbers from (5). Note how $\mathcal{C}$ tiles the entire lattice and that its coloring reflects the coloring of each sublatice $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ as in Fig. 3. Since $b = 3$, each line of colors is the same as the previous line, shifted by 1 mod 3.

Lemma 2.1. *Assume that each $\boldsymbol{p} \in \mathcal{L}(\boldsymbol{I})$ is assigned a color, color$(\boldsymbol{p})$, and that all points in each $\mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}}$ are assigned the same color, i.e., $\forall \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}}$, color$(\boldsymbol{p}_i) =$ color$(\boldsymbol{p}_j)$. Then color$(\boldsymbol{p} \bmod b) =$ color$(\boldsymbol{p})$.*

To take advantage of the b spacing of the points within each $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$, one obvious strategy is to assign every $\mathbf{c}_i$ in $\mathcal{C}$ (equivalently each sublattice) a unique color. This guarantees a distance $b-1$ coloring for the entire lattice $\mathcal{L}(\mathbf{I})$. In the context of our recursive splitting algorithm, the first split with $b_1 \in \mathbf{F}$ uses b_1^d colors, and achieves $b_1 - 1$ distance coloring. At the second recursive level with $b_2 \in \mathbf{F}$, each $\mathcal{L}(b_1\mathbf{I})_{\mathbf{c}}$ is split into b_2^d sublattices, each with a unique color, for a total of $(b_1 b_2)^d$ colors. Points in $\mathcal{L}(b_2\mathbf{I})_\mathbf{c}$ are at least b_2 hops apart, but these hops are edges in the $\mathcal{L}(b_1\mathbf{I})_\mathbf{c}$ lattice. Thus the minimum distance achieved by this coloring at the second level is $b_1b_2 - 1$. A simple inductive argument shows the following.

Lemma 2.2. *If at every level of the recursive splitting algorithm each sublattice is assigned a unique color, then at level m we have used $(b_1 \cdots b_m)^d$ colors and have achieved a distance $b_1 \cdots b_m - 1$ coloring.*

Lemma 2.2 provides the main strategy of our algorithm which at the completion of each recursive level increases the coloring distance exponentially, so probing with the corresponding vectors should be very effective. However, the number of colors (and thus of probing vectors) increases rapidly too; by a factor of b_m^d over the previous level. This is not an issue that harms the effectiveness of the algorithm, but it does make it difficult to frequently monitor the quality of the produced trace estimation because there is substantial computation needed between levels. Additionally, we cannot properly evaluate its progress at intermediate numbers of colors because after level $m-1$, probing cannot fully annihilate elements of distance up to $b_1 \cdots b_m - 1$ until all $(b_1 \cdots b_m)^d$ colors have been used.

In other words, the off-diagonal blocks of Fig. 1 would not be fully removed. Since, it may not be affordable to go to the next level, it would be desirable to also have one or more intermediate evaluation points within a level where smaller distance colorings complete. The mechanism of this is explained next and an example is shown in Fig. 5.

Such an intermediate coloring should be hierarchical with respect to the next level m. However, at level m each node in $\mathcal{C}$ (equivalently each sublattice) gets a different color, so any valid coloring in $\mathcal{C}$ will necessarily be hierarchical to the coloring at the m-level. In fact, even a non-valid coloring will be hierarchical too. For example, a red-black coloring of $\mathcal{C}$ is not a valid distance-1 coloring for any odd b since its toroidal property links nodes with the same color. However, we will see experimentally that the errors from ignoring these connections can be significant. In [13] we showed that three colors can provide a valid distance-1 coloring of any toroidal lattice. However, when $b \neq 3$, each color subset does not have the same number of nodes, which is an additional constraint we would like to impose because it facilitates the generation of probing vectors on-demand (see Section 2.7). Thus, since b is prime, we can only consider colorings with b, or $b^2, \ldots,$ or b^{d-1} colors.

We are not aware of a method that produces optimal distance colorings of $\mathcal{C}$ for any b and d. For small values of b, d we have identified heuristics that using b colors produce a valid distance $O(b^{1/d})$ coloring of $\mathcal{C}$. For practical problems as in LQCD, $d \leq 4$ and $b \leq 7$, so the effective distance achieved is not better than distance 1. Besides, an optimal coloring is not necessary as the nodes in the same colors will be hierarchically colored too so that nearby permuted nodes eventually have large distances. Therefore, we focus our attention on the following simpler method as our only intermediate point between two recursive levels.

Consider the strategy $color(\mathbf{c}) = \sum_{i=1}^{d} \mathbf{c}(i) \bmod b$ which colors $\mathcal{C}$ with b colors and ensures that each color appears the same number of times. Similarly on the lattice, if $\mathbf{p} \in \mathcal{L}(\mathbf{I})$, with $\mathbf{p} \bmod b = \mathbf{c}$, define its color:

$$color(\mathbf{p}) = \sum_{i=1}^{d} \mathbf{p}(i) \bmod b. \tag{9}$$

Because of Lemma 2.1, we have $color(\mathbf{p}) = color(\mathbf{c})$. Then we have the following.

Lemma 2.3. *The strategy (9) is a valid distance-1 coloring of $\mathcal{L}(\boldsymbol{I})$.*

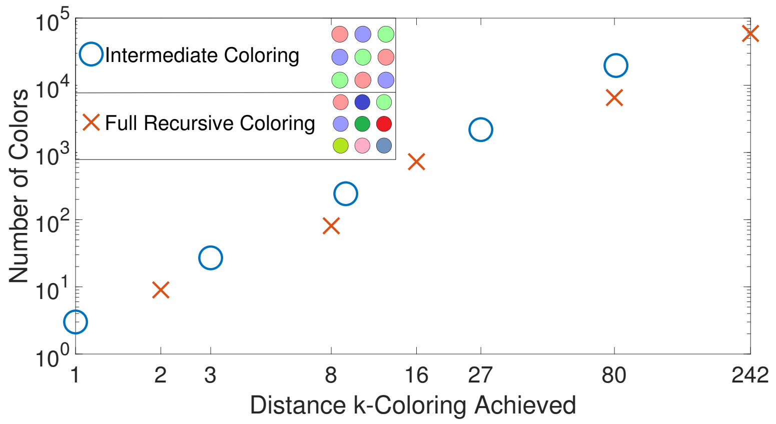


Fig. 5: Recursive and intermediate colorings of a 2D lattice of size $3^5 \times 3^5$. The recursive algorithm of Lemma 2.2 gives each sublattice a unique color and achieves distance-$\{2, 8, 26, 80, 242\}$ colorings only at the completion of each level, i.e., with $\{9, 81, 729, 6561, 59049\}$ colors. Since the number of colors between levels grows exponentially, to allow for more meaningful reporting points, we use (9) to color the tile with 3 colors before each recursive level completes. This produces also an intermediate coloring per recursive level. Then, each of the 10 colorings achieves distance-$\{1, 2, 3, 8, 9, 16, 27, 80, 81, 242\}$ coloring with $\{3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049\}$ colors respectively.

In the context of the recursive algorithm, assume that we have completed level m and points of the same color form the conformal lattice $\mathcal{L}^{(m)}$. From Lemma 2.2 we know that these points are at distance at least $b_1 \cdots b_m$ from each other in the original lattice. This distance corresponds to distance 1 in $\mathcal{L}^{(m)}$. Now we apply strategy (9) before the completion of level $m+1$. From Lemma 2.3 we know that this intermediate coloring guarantees that nodes of $\mathcal{L}^{(m)}$ of the same color are at least at distance two in $\mathcal{L}^{(m)}$. Then, we have shown the following in terms of the original lattice.

Corollary 2.4. *After the intermediate coloring with strategy (9) between m and $m+1$ recursive levels, we have used $(b_1 \cdots b_m)^d b_{m+1}$ colors and have achieved a distance $b_1 \cdots b_m 2 - 1$ coloring in $\mathcal{L}(\boldsymbol{I})$.*

The coloring strategy (9) has an efficient recursive implementation. Note that the colors in the i-th $(d-1)$-dimensional slice of $\mathcal{C}$ are the colors of the $(i-1)$-th $(d-1)$-dimensional slice shifted by 1 mod b. This can be seen in Fig. 4 for $d = 2$. Since the 0-th slice is the same as the coloring of $\mathcal{C}$ in $d-1$ dimensions, we can build the colorings for all dimensions in the following recursive manner.

Let $c_{d,b}$ be the array[1] of all b^d colors of the corresponding d-dimensional $\mathcal{C}$ in natural ordering. This $\mathcal{C}$ has b $(d-1)$-dimensional slices each corresponding to the $(d-1)$-dimensional lattice of offsets. Let $c_{d-1,b}$ be the array of the b^{d-1} colors of these $(d-1)$-dimensional lattices. Then, $c_{d,b}$ is a concatenation of b shifted $c_{d-1,b}$ arrays,

$$c_{d,b} = \{c_{d-1,b},\ c_{d-1,b} + 1 \bmod p,\ \ldots\ , c_{d-1,b} + (b-1) \bmod p\}\,. \tag{10}$$

Each shift applies to all the elements of the array $c_{d-1,b}$. In Fig. 4, for example, $c_{1,3} =$

[1]The notation of this array is not to be confused with the notation of offsets $\mathbf{c}$ which are in bold.

$\{0,1,2\}$ are the colors of a one dimensional lattice, but also the first row of the two dimensional $\mathcal{C}$. The colors of $\mathcal{C}$ are then $c_{2,3} = \{\{0,1,2\}, \{0,1,2\}+1 \bmod 3, \{0,1,2\}+2 \bmod 3\} = \{0,1,2,1,2,0,2,0,1\}$.

Algorithm 3 implements this recursive coloring of $\mathcal{C}$ starting with $c_{0,b} = 0$. Based on this coloring it generates the permutation, Perm, that reorders sublattices of the same color together. For example the nine sublattices in Fig. 3 we be ordered first the red ones corresponding to offsets 0, 5, and 7, then the blue ones corresponding to offsets 1, 3, and 8, and finally the green sublattices with offsets 2, 4, and 6.

Since the same number of sublattices share each of the b colors, we are able to index the location of the sublattices in the following way. The first sublattice of color i is assigned to the ib^{d-1} location, and the index to the next available free spot for this color stored in ColorIndex is incremented by one. We can then easily recursively process the sublattices in these locations and in that order. Algorithm 3 incurs negligible computational cost, even for very large lattices, while it enables the additional intermediate step which allows a more frequent monitoring of the trace estimation. This low cost is an advantage over other potential colorings that could be used to define different intermediate steps.

Algorithm 3 Perm$(0,\ldots,b^d-1)$ = GenOffsetPermutation(b,d)
% Generate the b-coloring permutation of $\mathcal{C}$ which reorders the sublattices (offsets)
% Input: prime factor b, lattice dimension d
% Output: Perm, the b-color permutation of the b^d sublattices

```
1: % Generate the coloring of the tile row by row using (10)
2: $c_{0,b} \leftarrow \{0\}$
3: for $j = 1 \to d$ do
4:     $c_{j,b} \leftarrow \{c_{j-1,b},\ c_{j-1,b}+1 \bmod b,\ \ldots,\ c_{j-1,b}+b-1 \bmod b\}$
5: end for
6: % Initialize index showing where the next sublattice of color $i$ should go
7: for $i = 0 \to b-1$ do
8:     ColorIndex(i) $\leftarrow i * b^{d-1}$
9: end for
10: for $i = 0 \to b^d-1$ do
11:     Color $\leftarrow c_{d,b}(i)$            % Lookup the color of sublattice $i$ in array $c_{d,b}$
12:     Perm($i$) $\leftarrow$ ColorIndex(Color)            % The new location of sublattice $i$
13:     ColorIndex(Color) $\leftarrow$ ColorIndex(Color) +1
14: end for
        return Perm
```

We now have a method that at each level m recursively splits a lattice into $\mathbf{F}(m)^d$ sublattices, giving each a different color. But before this level m completes, we have an intermediate coloring that groups together $\mathbf{F}(m)^{d-1}$ sublattices. The method provides the guarantees of Lemma 2.2 and Corollary 2.4. Next, we describe the global hierarchical permutation and in particular how to find the location in this permutation of an arbitrary node. This will be used to efficiently generate the hierarchical probing vectors.

2.5. Hierarchical Permutations of Lattices with Dimensions of Equal Length. The final permutation can be obtained recursively by applying the coloring permutation from level m on the permuted index from level $m-1$. This ordering ensures that the closer the nodes are geometrically, the farther they are ordered in the permutation. Orderings of lower levels provide no additional information, since nodes never move closer together in subsequent levels, so need not be stored. Moreover, we can avoid the above recursion by determining directly the final location of the node.

Consider the example in Fig. 3. At level 0, the 3-coloring of the lattice permutes the colors in three groups as shown below on the left(here we show the three coloring first to illustrate a more interesting split, although since the shared factors are 2 and 3, normally the algorithm would start with the smaller factor 2).

Level 0, after 3-coloring		Level 1, after splitting to 3^2 sublattices			
color 0:	0 3 8 11 13 16 18 21 26 29 31 34	offsets 0,5,7:	0 3 18 21	8 11 26 29	13 16 31 34
color 1:	1 4 6 9 14 17 19 22 24 27 32 35	offsets 1,3,8:	1 4 19 22	6 9 24 27	14 17 32 35
color 2:	2 5 7 10 12 15 20 23 25 28 30 33	offsets 2,4,6:	2 5 20 23	7 10 25 28	12 15 30 33

At level 1, since $b = 3$, we split the top level lattice to 9 sublattices. Three of those lattices (offsets 0, 5, 7) contain only the red nodes from the intermediate coloring and need to be ordered first. Notice how the actual permutation of the red nodes at level 0 is not needed as it is present in the ordering of the sublattices by Algorithm 3.

Algorithm 4 shows we can construct the final hierarchical permutation. Given the coordinates of a node, $\mathbf{p}$, Algorithm 2 generates the indices $[i^{(1)}, \ldots, i^{(f)}]$ of the sublattices the $\mathbf{p}$ lies in at each level. Algorithm 3 permutes these to $[\hat{i}^{(1)}, \ldots, \hat{i}^{(f)}] = \text{Perm}([i^{(1)}, \ldots, i^{(f)}])$. Because the permutation preserves the hierarchy, the location of the node $\mathbf{p}$ would be determined by all the nodes that belong to sublattices that appear prior to its own sublattices in the final permutation. For example, there are exactly $\hat{i}^{(1)}$ sublattices preceding it at the first level, $\hat{i}^{(2)}$ sublattices preceding it at the second level, and so on. At every level m the size of each sublattice reduces by a factor of $F(m)^d$. Thus, given the lattice size $L = \prod_{i=1}^{d} d_i$, we have

$$Location(\mathbf{p}) = \sum_{m=1}^{f} \hat{i}^{(m)} \frac{L}{\prod_{l=1}^{m} \mathbf{F}(l)^d}. \tag{11}$$

Algorithm 4 Location = HPpermutation($\mathbf{p}, d, \mathbf{F}$)
% Compute the Hierarchical Probing permutation of a node $\mathbf{p}$ when $d_1 = \cdots = d_d$
% Input: point coordinates $\mathbf{p}$, lattice dimension d, common prime factors $\mathbf{F}$
% Output: the location of $\mathbf{p}$ in the HP permutation

```
1: $[i^{(1)}, \ldots, i^{(f)}]$ = SublatticeIndicesOfPoint($\mathbf{p}, \mathbf{F}$)        (Algorithm 2)
2: subLatticeSize = $\prod_{i=1}^{d} d_i$
3: Location = 0
4: for $m = 1 \to f$ do
5:     Perm = GenOffsetPermutation($\mathbf{F}(m), d$)        (Algorithm 3)
6:     subLatticeSize = subLatticeSize/$\mathbf{F}(m)^d$
7:     Location = Location + Perm($i^{(m)}$)*subLatticeSize
8: end for
       return Location
```

2.6. Hierarchical Permutations of Lattices with Dimensions of Unequal Length. Algorithm 4 relies on having each lattice split into an equal number of sublattices of the same dimensionality. However, it is possible that one dimension of the lattice may be smaller than the others, leading to that dimension being exhausted before the others. If the rest of the dimensions share factors, the algorithm can continue but in a lower dimensionality lattice that removes exhausted dimensions. Let $\hat{d}^{(m)}$ be the number of active dimensions at level m. For example, the lattice $6 \times 6 \times 2$, has factors $(2,3),(2,3),(2)$, so for the first level $\hat{d}^{(1)} = 3$, while for the second level $\hat{d}^{(2)} = 2$, since at the second level all sublattices will be 2 dimensional. Thus, $\hat{d}^{(m)}$ can be computed simply by counting the number of common factors in each dimension that has not been exhausted. Then, the new location of a node is given by (12), [2]

$$Location(\mathbf{p}) = \sum_{m=1}^{f} \hat{\imath}^{(m)} \frac{L}{\prod_{l=1}^{m} \mathbf{F}(l)^{\hat{d}^{(m)}}}. \tag{12}$$

We can avoid computing and storing coloring permutations for lattices with reducing dimensionality by reusing the previously computed permutations for $\mathcal{C}$ of a higher dimensionality in (8), as long as the spacing b is the same. First, recall that for a given $\mathcal{C}$ of dimensionality d and spacing b, the coloring $c_{d,b}$ in (10) is created recursively. This means that the color of a particular node in $\mathcal{C}$ can be given in terms of either a higher or a lower dimensional $\mathcal{C}$ plus a correctional offset as below,

$$c_{d,b}(k) = c_{d-1,b}(k \bmod b^{d-1}) + \lfloor \frac{k}{b^{d-1}} \rfloor \bmod b, \ \forall k < b^d, \tag{13}$$

$$c_{d-1,b}(k \bmod b^{d-1}) = c_{d,b}(k) - \lfloor \frac{k}{b^{d-1}} \rfloor \bmod b, \ \forall k < b^d. \tag{14}$$

Lemma 2.5. *For any prime b, $c_{d,b}(ib) = c_{d-1,b}(i)$, $\forall i = 0,\dots,b^{d-1}-1$.*

Lemma 2.6. *For any prime b, $c_{d,b}(ib) = c_{d,b}(ib+q) - q \bmod b$, $\forall i = 0,\dots,b^{d-1}-1, \forall q = 0,\dots,b-1$.*

Lemma 2.7. *Let* Perm_d *be the permutation created by Algorithm 3 associated with dimension d. Then, for any prime b, $i = 0,\dots,b^d-1$, $Perm_d(i) = \lfloor i/b \rfloor + c_{d,b}(i) b^{d-1}$.*

Theorem 2.8. *For any $0 < m < d$,* Perm_m *can be obtained directly as follows,*

$$\mathrm{Perm}_m(i) = \lfloor \mathrm{Perm}_d(ib^{d-m})/b^{d-m} \rfloor, \ i = 0,\dots,b^m-1.$$

Based on Theorem 2.8, Algorithm 5 shows how to reuse previously generated $c_{d,b}$ to compute permutations for lower dimensional lattices when the smaller lattice dimensions are exhausted.

2.7. Generating Probing Vectors Quickly. After computing the hierarchical probing permutation of a lattice we need to generate the probing vectors. As explained in Section 1.2, the canonical probing vectors cannot be used as a hierarchical basis. We need a different orthonormal basis that spans the same space but it is efficient to compute. In [13], we introduce the following recursive method for generating probing vectors for a colored lattice,

$$\tilde{Z}_{(1)} = \mathcal{F}_{z(1)}, \tag{15}$$

$$\tilde{Z}_{(i)} = \left[\tilde{Z}_{(i-1)} \otimes \mathcal{F}_{z(i)}(:,1), \dots, \tilde{Z}_{(i-1)} \otimes \mathcal{F}_{z(i)}(:,z(i)) \right], \tag{16}$$

$$Z_{(i)} = \tilde{Z}_{(i)} \otimes \mathbf{1}_{N/\gamma_i}, \quad \text{where } \gamma_i = \prod_{j=1}^{i} z(j). \tag{17}$$

[2]It is worth noting that just as [13] interpreted this process as representing the node number in binary and then permuting the digits, we can represent each node in mixed radix, where the radix list is the color numbers used to color the sublattices at each level, and then permute these digits.

Algorithm 5 Location = HPpermutation_general($\mathbf{p}, d, d_i, \mathbf{F}$)
% Compute the Hierarchical Probing permutation of a node **p** for general d_i
% Input: point coordinates **p**, lattice dimension d, common prime factors **F**
% Output: the location of **p** in the HP permutation

```
1: $[i^{(1)}, \ldots, i^{(f)}]$ = SublatticeIndicesOfPoint($\mathbf{p}, \mathbf{F}$)        (Algorithm 2)
2: subLatticeSize = $\prod_{i=1}^{d} d_i$
3: Location = 0
4: $\hat{d}^{(1)} = d$                                % Number of active dimensions
5: $\bar{\mathbf{d}} \leftarrow []$
6: for $m = 1 \to f$ do
7:     $\hat{d}$ =setActiveDims()
8:     alreadyseen = false
9:     for $i = d : -1 : \hat{d}$ do
10:        perm=getHash(b,d);
11:        if perm = empty  then
12:            alreadyseen = true;
13:            break
14:        end if
15:    end for
16:    if alreadyseen==true then
17:        for $i = 1 : b^{\hat{d}}$ do
18:            newperm(i)=perm($ib^{d-\hat{d}}/b^{d-\hat{d}}$)
19:        end for
20:        perm = newperm
21:    else
22:        Perm = GenOffsetPermutation(F(m), d) (Algorithm 3)
23:    end if
24:    setHash(b,d)=perm;
25:    Perm = GenOffsetPermutation($\mathbf{F}(m), d$)        (Algorithm 3)
26:    subLatticeSize = subLatticeSize/$\mathbf{F}(m)^d$
27:    Location = Location + Perm($i^{(m)}$)*subLatticeSize
28: end for
29: %Make an array of the offsets $\mathbf{c}$ at each level $m$, where $k = size(\mathbf{F})$.
30: $\bar{\mathbf{d}} \leftarrow []$
31: for $m = 1 \to k$ do
32:    if  not lower dim version  then %Check for lower dimensional version
33:        $\mathbf{c}^{'m} \leftarrow$ Algorithm3($\mathbf{c}^m$) % Use Algorithm 3 to reorder the lattices
34:    else
        $\mathbf{c}^{'m}$ =$\mathbf{c}^{'m}_{d-1}$
35:    end if
36:    $\bar{d}_m \leftarrow$ size( $\mathbf{c}^{'m}$ ) %Find active dimensions
37: end for
38: $\mathbf{i} \leftarrow []$
39: for $m = 1 \to k$ do
40:    $i_m \leftarrow$ ConvertOffsetsToIndex($\mathbf{c}^{'m}$,$\mathbf{F}$) %Use 5 to convert $\mathbf{c}^{'m}$ to an integer
41: end for
42: newLocation ← FindLocation($\mathbf{i}$,$\mathbf{F}$,LatticeSize,$\bar{\mathbf{d}}$) % Use 12 for node ordering
43: return newLocation
```

Here $\mathcal{F}_{z(i)}$ is the Fourier transform of the identity matrix $I_{z(i)}$, $z(i)$ is the number of colors each sublattice is split into at level i, $\mathbf{1}_s$ is the vector of s ones, and $\otimes$ is the Kronecker product. Essentially, these vectors build recursively a basis for the probing vectors. At each level, we probe inside each color (i.e., sublattice) by smaller probing vectors hierarchically, which are all assembled into a basis through the Kronecker products. Instead of generating the whole matrix, however, we produce each probing vector one at a time, hence requiring the same memory as the Hutchinson method.

To produce the k-th vector of the probing matrix, we first need to identify the maximum level i needed such that $\gamma_{i-1} < k \leq \gamma_i$. Then at every lower recursive level of (16) we only need two vectors; one vector from $Z_{(i-1)}$ and one from $\mathcal{F}_{z(i)}$. By (16), the matrix $Z_{(i)}$ is divided into $z(i)$ blocks, with each block forming a Kronecker product with a different column of $\mathcal{F}_{z(i)}$. Since each block has $z(i-1)$ columns, the k-th vector is in $block = \lfloor \frac{k}{z(i-1)} \rfloor$, and thus we can generate directly the desired column $\mathcal{F}_{z(i)}(:, block)$. This should be paired with the $(k \bmod z(i-1))$ vector of $Z_{(i-1)}$ which we find recursively with the above procedure.[3]

When $\mathbf{F}(i) = 2,\ i = 1, \ldots, k$, the sublattices in the first k levels are red-black colorable, and thus use only $\mathcal{F}_2$. Because $\mathcal{F}_2$ is equal to the Hadamard matrix H_2, all vectors in the first k levels can be created using real arithmetic, which yields substantial savings over complex arithmetic. Moreover, we can use the fast bit-based method we introduced in [13] for producing the required Hadamard vectors, leading to an additional performance gain. This approach can be seen in Algorithm 6.

Algorithm 6 ProbingVector = GenerateProbingVector(k, z, N)
% Compute the k-th probing vector
% Input: k, the number of colors at each level $z(i)$, the matrix size N
% Output: The probing vector

```
1: for $j = size(z)$ downto 2 do          % Compute indices of needed $\mathcal{F}_{z(i)}$ vectors
2:     $block(j) = \lfloor k/z(i-1) \rfloor;\ \ k = k \bmod z(i-1)$
3: end for
4: $block(1) \leftarrow k$;
5: % Find the initial 2-colorable sublattices, which can be probed quickly as in [13]
6: while $z(j) == 2$ do $j \leftarrow j+1$; end while
7: fastLevels $\leftarrow j - 1$
8: ProbingVector $\leftarrow$ [1]
9: ProbingVector $\leftarrow$ FastHadamardMethod(1:fastLevels) % The Fast Hadamard
   Method of [9]
10: % The rest of the levels are built through Fourier vectors
11: for $j$ = fastLevels+1 $\rightarrow$ size(z) do
12:     % Create Fourier vectors $\mathcal{F}$
13:     $f \leftarrow 2 * \pi / z(i)$
14:     $w \leftarrow [0 : f : (2 * \pi - f/2)] * \sqrt{-1}$
15:     $\mathcal{F}_{z(i)}(:, block) \leftarrow exp(-w \otimes block(j))$
16:     ProbingVector $\leftarrow$ ProbingVector $\otimes \mathcal{F}_{z(i)}(block, :)$
17: end for
        return ProbingVector
```

[3] We note that this process can also be described in terms of radix conversion. Let the $z(i)$s be taken as the radix list. If k is converted to this mixed radix form [11], the vectors of the Fourier transforms $\mathcal{F}_{z(i)}$ needed at each level will be the digits of this representation.

As mentioned before, we would like to terminate probing early if the results are sufficiently accurate. Thus, we are interested in checking our results with the probing vectors we have generated thus far. However, not every subset of probing vectors is a valid intermediate coloring. Since Algorithm 6 is based on coloring strategy (9), the first γ_i probing vectors in (17) correspond to what we termed as the intermediate coloring between splitting levels $i-1$ and i. However, there are other numbers k, with $\gamma_{i-1} < k < \gamma_i$, for which $Z(:, 1:k)$ are the probing vectors for a different valid coloring, most importantly the one in Lemma 2.2 where each sublattice gets a unique color. This is because the coloring is hierarchical, i.e., (9) is applied independently on each sublattice. Let us return to the example in Fig. 3, where we first consider the factor $b = 3$ and then $b = 2$. The figure shows the results after the 3-coloring and the first level sublattice split.

Here we focus only on color 0:

Level 0, after 3-coloring
color 0: $\boxed{0\ 3\ 8\ 11\ 13\ 16\ 18\ 21\ 26\ 29\ 31\ 34}$

Level 1, after splitting to 3^2 sublattices
offsets 0,5,7: $\boxed{0\ 3\ 18\ 21}$ $\boxed{8\ 11\ 26\ 29}$ $\boxed{13\ 16\ 31\ 34}$

Level 1, after 2-coloring each sublattice we have a total of 2×3^2 colors
original color 0 now contains $6 = 2 \times 3$ colors $\boxed{0\ 21}$ $\boxed{3\ 18}$ $\boxed{8\ 29}$ $\boxed{11\ 26}$ $\boxed{13\ 34}$ $\boxed{16\ 31}$

Notice therefore that by construction the first nine indices in our final permutation (which is used to generate the probing vectors $Z(:, 1:9)$) correspond to the coloring at level 1 where each lattice has a different color.

2.8. Probing Vectors For Hierarchical Coloring on General Graphs. The above method for generating probing vectors assumes each color splits into the same number of colors at a given level. This is the case with our methods in Section 2.4. With an arbitrary coloring method that does not assign the same number of sublattices to each color (e.g., the 3-coloring method on a lattice with a length not divisible by three), a different way to generate probing vectors is needed. A simple but not as efficient solution is to create the required canonical probing vectors, and then orthogonalize them against previous vectors in the sublattice as well as each other with Gram-Schmidt. We introduce a more efficient and elegant method that works with uneven color splits and thus generalizes probing to any arbitrary matrix with hierarchical coloring. Although this situation does not arise in lattices, it may in algorithms used for more general matrices, such as the heuristic that we introduce later.

The method is described better through an example. Consider a graph with seven nodes (each node could be generalized to be a subgraph). Suppose at level 0 the entire graph is assigned three colors. After the corresponding permutation, the first color contains nodes [1,2,3], the second color nodes [4,5], and the third color nodes [6,7]. To probe at level 0 we use the following three probing vectors(each probing vector is a column vector of $\mathcal{Z}_{(1,0)}$), which is a variation of $\mathcal{F}_3$ in (15) and (17) to allow for different number of nodes per color,

$$\mathcal{Z}_{(1,0)} = \begin{bmatrix} \mathcal{F}_3(1,:) \otimes \mathbf{1}_3 \\ \mathcal{F}_3(2,:) \otimes \mathbf{1}_2 \\ \mathcal{F}_3(3,:) \otimes \mathbf{1}_2 \end{bmatrix} \in \mathbb{C}^{7\times 3}. \tag{18}$$

The second index in $\mathcal{Z}_{(1,0)}$ shows the level and the first shows the color block at the previous level that is split (at level 0 there is only 1 block, the entire graph). Suppose now that at the next level, 1, the first color splits into three colors and the others into two. Clearly, the next level of probing vectors cannot be created by (16) because of uneven splitting. Each color block of the first level has to be probed independently. Thus, we could probe the first block using $\mathcal{F}_3$ for the elements inside the first block (with zeros everywhere else in the probing vector). Similarly for the last two blocks, but using $\mathcal{F}_2$. These seven probing vectors, grouped as three blocks, are shown in (19) —note that $\mathbf{0}_k = \text{zeros}(k,1)$,

$$\begin{bmatrix}\mathcal{F}_3(:,1)\\ \mathbf{0}_2\\ \mathbf{0}_2\end{bmatrix}\begin{bmatrix}\mathcal{F}_3(:,2)\\ \mathbf{0}_2\\ \mathbf{0}_2\end{bmatrix}\begin{bmatrix}\mathcal{F}_3(:,3)\\ \mathbf{0}_2\\ \mathbf{0}_2\end{bmatrix},\begin{bmatrix}\mathbf{0}_3\\ \mathcal{F}_2(:,1)\\ \mathbf{0}_2\end{bmatrix}\begin{bmatrix}\mathbf{0}_3\\ \mathcal{F}_2(:,2)\\ \mathbf{0}_2\end{bmatrix},\begin{bmatrix}\mathbf{0}_3\\ \mathbf{0}_2\\ \mathcal{F}_2(:,1)\end{bmatrix}\begin{bmatrix}\mathbf{0}_3\\ \mathbf{0}_2\\ \mathcal{F}_2(:,2)\end{bmatrix}. \quad (19)$$

The problem is that using seven vectors would be wasting the solutions of linear systems with the three probing vectors (18) in the first step.

The key to remedying this problem is to note that the three first vectors of the new color blocks,

$$\mathcal{I} = \begin{bmatrix}\mathcal{F}_3(:,1) & \mathbf{0}_3 & \mathbf{0}_3\\ \mathbf{0}_2 & \mathcal{F}_2(:,1) & \mathbf{0}_2\\ \mathbf{0}_2 & \mathbf{0}_2 & \mathcal{F}_2(:,1)\end{bmatrix} = \begin{bmatrix}\mathbf{1}_3 & \mathbf{0}_3 & \mathbf{0}_3\\ \mathbf{0}_2 & \mathbf{1}_2 & \mathbf{0}_2\\ \mathbf{0}_2 & \mathbf{0}_2 & \mathbf{1}_2\end{bmatrix}$$

are spanned by the vectors of $\mathcal{Z}_{(1,0)}$, since $\mathcal{F}_3$ is a basis of $\mathbb{C}^3$. More formally, if $\mathbf{a} \in \mathbb{C}^{3\times 3}$, from (18) and basic properties of the Kronecker product we have that the following matrix equation

$$\mathcal{Z}_{(1,0)}\mathbf{a} = \begin{bmatrix}\mathcal{F}_3(1,:)\mathbf{a}\otimes\mathbf{1}_3\\ \mathcal{F}_3(2,:)\mathbf{a}\otimes\mathbf{1}_2\\ \mathcal{F}_3(3,:)\mathbf{a}\otimes\mathbf{1}_2\end{bmatrix} = \mathcal{I},$$

is equivalent to $\mathcal{F}_3\mathbf{a} = I_3$, which has a unique solution $\mathbf{a} = \text{ifft}(I_3)$, i.e., the inverse Fourier transform of the identity. Therefore, if we saved $P = A^{-1}\mathcal{Z}_{(1,0)}$, we can recover the probing result for the vectors in $\mathcal{I}$ as $A^{-1}\mathcal{I} = P\mathbf{a}$. Thus, we only need to apply A^{-1} on the remaining four probing vectors, exactly as in our HP method on the lattice. Finally, if each node represents a subgraph, each color block in (19) involves Kronecker products of the rows of its Fourier matrix with columns of ones, each sized to the cardinality of the subgraph. Thus, each block has the same form as (18) and the idea can be applied recursively.

To generalize we need the following definitions. First, assume a hierarchical coloring at levels $i = 0, 1, \ldots$, and let l_i be the number of colors at level i, with the convention $l_{-1} = 1$. The nodes belonging to the same color at level i are called together a "block" at the next level $i+1$. There are l_{i-1} blocks at the i-th level. Let $s(j,i)$ be the number of colors the j-th block splits into at level i. Thus, $\sum_{j=1}^{l_{i-1}} s(j,i) = l_i$. For each color in block j, let $n(j,i,k)$, $k = 1, \ldots, s(j,i)$ be the number of nodes in that color. Thus, $\sum_{k=1}^{s(j,i)} n(j,i,k)$ is the number of nodes in the j-th block. For each $j = 1, \ldots, l_{i-1}$ block, define the Fourier transform $\mathcal{F}_{s(j,i)} = \text{fft}(I_{s(j,i)})$, and $\mathcal{Z}_{(j,i)}$ the set of probing vectors as

$$\mathcal{Z}_{(j,i)} = \begin{bmatrix}\mathbf{0}\\ \mathcal{F}_m(1,:)\otimes\mathbf{1}_{n(j,i,1)}\\ \vdots\\ \mathcal{F}_m(m,:)\otimes\mathbf{1}_{n(j,i,m)}\\ \mathbf{0}\end{bmatrix}, \text{ where } m = s(j,i). \quad (20)$$

The $\mathbf{0}$ zero matrices have m columns and rows that overlap with all other blocks. At level 0, there is only one block ($l_{-1} = 1$, with $s(1,0)$ colors), so the $\mathbf{0}$ matrices are empty. In the previous example with 7 nodes, the single block at level 0 was denoted as $\mathcal{Z}_{(1,0)}$ and (19) had three blocks $\mathcal{Z}_{(1,1)}$, $\mathcal{Z}_{(2,1)}$, $\mathcal{Z}_{(3,1)}$ with 3,2,2 vectors respectively. Define also as $\mathcal{Z}_{(i)} = [\mathcal{Z}_{(1,i)} \ldots \mathcal{Z}_{(l_{i-1},i)}]$ all the probing vectors at level i.

Assume that the results of the inversions at level $i-1$ have been saved $P_{i-1} = A^{-1}\mathcal{Z}_{(j,i-1)}$ for all blocks $j = 1, \ldots, l_{i-2}$. At the i-th level, probing with the first vector of each block can be determined as follows:

$$\mathbf{a} = \text{ifft}(I_{l_{i-1}}), \tag{21}$$

$$A^{-1}\mathcal{Z}_{(j,i)}(:,1) = P_{i-1}\mathbf{a}(:,j),\ j = 1, \ldots, l_{i-1}, \tag{22}$$

or equivalently note that

$$P_{i-1}\mathbf{a} = \text{ifft}(P_{i-1}^H)^T. \tag{23}$$

So we can pick the appropriate columns of $P_{i-1}\mathbf{a}$, which we have previously used to probe with as our solutions while systems for the remaining $l_i - l_{i-1}$ probing vectors in $\mathcal{Z}_{(i)}$ are solved explicitly. At the end of level i, we have inversions for all the probing vectors $\mathcal{Z}_{(i)} = [\mathcal{Z}_{(1,i)} \ldots \mathcal{Z}_{(l_{i-1},i)}]$. The process continues recursively as described in Algorithm 7. We emphasize that our new method fuses the generation of probing vectors and the solution of linear systems needed for the trace computation. However, Algorithm 7 depicts only the generation of the vectors.

Our method requires storage for P_{i-1} at level $i-1$. To compute P_i at the next level, Algorithm 7 first permutes the implicitly computed vectors $P_{i-1}\mathbf{a}$ in their new positions and then solves the linear systems for the remaining P_i vectors. Because of the tree structure, the total storage is $l_{imax-1} < \min_j s(j, l_{imax-1})$, which is always less than half of the final number of probing vectors at level l_{imax}.

Computationally, at level i, because we have created a set of probing vectors that allows us to reuse all our previous work, so we have avoided the solution of l_{i-1} additional solutions of systems of equations. We do however require an additional l_{i-1} inverse FFTs in (23), or a $\mathcal{O}(Nl_{i-1}\log l_{i-1})$ cost. Moreover, this is more elegant and less expensive than a brute-force Gram-Schmidt which costs $\mathcal{O}(Nl_{i-1}^2)$ at level i. Finally we remind the reader that this method works for hierarchical colorings on arbitrary graphs.

3. Hierarchical Probing for Symmetric Matrices with General Graph Structure.

The hierarchical coloring algorithms of the previous sections take advantage of a-priori information about the distances between nodes of a lattice. Clearly, this information is unavailable for arbitrary matrices. Classical probing obtains the distances between nodes directly by computing successive powers of the adjacency matrix, a memory and often computationally intensive operation. To avoid this and to ensure hierarchical colorings we propose a multi-level approach where, at each level, neighboring nodes and their neighborhoods are merged and the smaller graph is colored. Distances between merged nodes shrink as the graph becomes smaller. This ensures that nodes that are further away on the original graph are eventually forced to be in different colors in a hierarchical manner, since their merged nodes will eventually become close. The color of a node in relation to the fine level graph is determined by its colors at each level.

A key component to such an algorithm is the merging (or aggregation) strategy. Consider the 2-dimensional lattice in Fig. 6 and assume that at every level we only

Algorithm 7 GenerateAndPerformProbingVector_general(s, l, n, i_{max})
% Input: $s(j,i)$: number of colors the j-th block splits into at the i-th level,
% $n_{(j,i,k)}$: the number of nodes in the color k subgraph of block j
% l_{i-1}: the number of colors at level $i-1$, also the number of blocks at level i
% i_{max}: maximum desired level
% Output: The probing vectors $\mathcal{Z}$ at level i_{max}.

```
1: $\mathcal{Z} \leftarrow [\ ]$
2: $\mathcal{F}_{s(1,0)} \leftarrow \text{fft}(I_{s(1,0)})$
3: Build $\mathcal{Z}_{(1,0)}$ using (20) and the coloring permutation      % Level 0
4: $P \leftarrow [\ A^{-1}\mathcal{Z}_{(1,0)}]$
5: for $i = 1 \rightarrow i_{max}$ do                                    % Level $i$
6:     $P \leftarrow \text{ifft}(P^H)^T$
7:     $newpos(1) = 1$
8:     for $j = 2 \rightarrow l_{i-1}$ do                                % block $j$
9:         $newpos(j) = newpos(j-1) + s(j-1,i)$ % new positions of $P\mathbf{a}$ at level $i$
10:    end for
11:    $P(:, newpos) = P(:, 1 : l_{i-1})$                                % Permute to new positions
12:    for $j = 1 \rightarrow l_{i-1}$ do                                % block $j$
13:        $\mathcal{F}_{s(j,i)} \leftarrow \text{fft}(I_{s(j,i)})$
14:        Build $\mathcal{Z}_{(j,i)}$ using (20) and the coloring permutation
15:        for $k = 2 \rightarrow s(j,i)$ do                             % color $k$ in block $j$
16:            $P(:,k) \leftarrow A^{-1}\mathcal{Z}_{(j,i)}(:,k)]$
17:        end for
18:    end for
19: end for
```

merge nodes along the horizontal dimension. Regardless of whether we decide to merge nearest neighbors or neighbors that are farther away, the two circled nodes remain distance-1 apart at all levels. Even though these nodes should receive different colorings at some level, using the merging strategy shown in Figure 6 ensures that this never happens, and their strong link will not be eliminated by such a coloring. On lattices, we could alternate merge directions to avoid this problem but there is no such analog for arbitrary graphs.

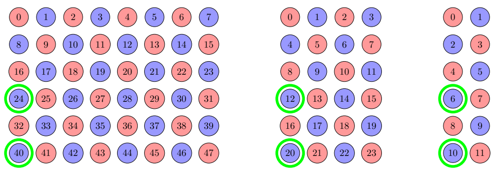


Fig. 6: 2-D multi-level lattices using a naive merging strategy only along the horizontal dimension. After three merges the circled nodes still receive the same color, even though they are only one hop away from each other.

Our solution is to provide appropriate ways to merge not only nodes but also their neighborhoods, producing a coarse representation of the fine level graph. Let G_i be the graph at level i. To produce the coarse graph at level $i+1$, we select a yet unmerged node $x \in G_i$ and one of its unmerged (distance-1) neighbors, y. If no such neighbor exists, x remains a node at the coarse graph G_{i+1} and keeps its neighborhood. Otherwise, we merge x and y as one node in G_{i+1} and set its neighborhood to the union of the distance-2 neighborhoods of x and y. This guarantees that all nodes that are distance-2 apart in G_i are neighbors in G_{i+1} and thus at level i we have eliminated distances at least up to 2^i in the fine level graph. This is comparable to our HP algorithm for lattices. The process repeats until there are no unmerged nodes in G_i.

At each level, we apply a greedy coloring algorithm on G_i. The color given to a node $x \in G_i$ is associated with all the nodes that are merged into x at finer $G_k, k < i$, and serves as a digit in a mixed-radix representation of the final color of each fine level node, as in the original hierarchical coloring approach.

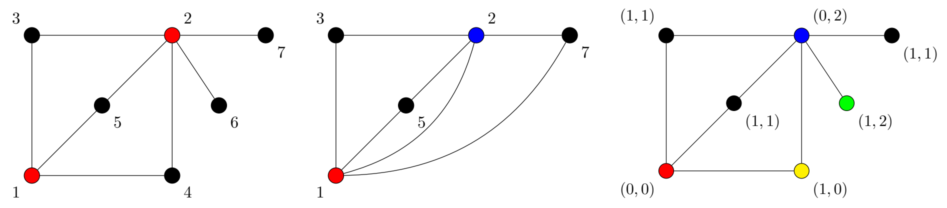
However, the color numbers this representation gives to the fine level nodes, although unique, may have gaps. In other words, not all consecutive color numbers are assigned to nodes and thus some nodes may have colors larger than N. The reason is that the coloring at level i produces more colors than required for eliminating all links up to distance-2^i in G_0. This is mainly because of the aggregate nature of the coarse graphs but also because of the greedy algorithm used for coloring. To avoid this issue, we simply examine the colors produced by the mixed-radix representation and renumber them to remove any gaps, which also keeps the total number of colors no more than N. We show an example of this later in Fig. 7. Finally, after the coloring has been created, we can use the same probing vector generation of Algorithm 7.

Our new method is shown in Algorithm 8. We first color the current graph and if every node has a different color, the algorithm terminates. Otherwise we continue to generate a coarser version of the graph. We first obtain the neighborhood of each node and check if it has an unmerged neighbor. If so, we merge those nodes and their distance-2 neighborhoods (which includes the nearest neighbors). Finally we continue the recursion on the resultant coarser graph. We note that although Algorithm 8 produces the distance-2 neighborhood of each node using breath first search (computing A^2), we could also have saved memory by computing the neighbourhood of each node one at a time and then coloring, instead of computing A^2 all at once.

Algorithm 8 Coloring = MultilevelColoring(A,c)
% Compute a Multilevel coloring of A
% Input: A,c the matrix to be colored, the coloring from the last level
% Output: The coloring at each level

```
 1: $newc \leftarrow$ Color($A$)
 2: if max($newc$) == size($A$) then          % If graph is dense, end recursion
        return [$newc$, $c$]
 3: end if
 4: $A_2 \leftarrow A^2$     % Find the dist-2 adjacency matrix
 5: $A_c \leftarrow \emptyset$     % Initialize coarser graph for next level
 6: for $v \in A$ and $v$ unmerged do       % For each yet unmerged node in the graph
 7:     $w \leftarrow$ UnmergedNeighbor($v$)       % Find an unmerged neighbor of $v$ in $A$
 8:     $z = \{v, w\}$ ; $A_c \leftarrow A_c \cup z$          % Merge nodes and add to coarser graph
 9:     Neigh($A_c, z$) $\leftarrow$ Neigh($A_2, v$) $\cup$ Neigh($A_2, w$)
10: end for
        return MultilevelColoring($A_c$,[$c$,$newc$]) % Recursion on coarser grid
```

Fig. 7 shows an example of our algorithm producing a hierarchical coloring for a graph where Algorithm 5 cannot be used. At the fine level, the graph can be colored using a red-black coloring. The algorithm then merges node 1 with 4 and their distance-2 neighborhoods, thus linking the new node with nodes 2, 6 and 7. Next we merge nodes 6 and 2, repeating the process. At this point the remaining nodes have no more unmerged neighbors so this is the next level graph that the algorithm colors (see Fig. 7(b)). This graph can be colored with 3 colors (shown as red, black, and blue in the figure). To produce the hierarchical coloring for this second level we take the color at each level and generate a mixed-radix integer representation (Fig. 7(c)). As in our previous algorithms the first color applied should be the most significant digit of our mixed-radix representation, which each subsequent level coloring becoming the next most significant digit of the final color. As an example, node 6 has color 1 at level 0 and color 2 at level 1 (since it is in the merged node labeled 2 in Fig. 7(b)). In mixed-radix basis this is color number (1,2). Since the radix for level 1 was 3, the fine level node 6 gets color $1 * 3 + 2 = 5$. Similarly, node 2 with radix representation (0,2) takes $0 * 3 + 2 = 2$. The total list of numbers produced in this fashion are $(0, 0) = 0, (0, 2) = 2, (1, 0) = 3, (1, 1) = 4, (1, 2) = 5$. Note that there is no mixed-radix number (0,1) in this list, and thus the color 1 in the final list of colorings does not occur, so the color list must be adjusted for gaps as described previously.



(a) Coloring of the original graph

(b) Coloring after merging (1,4), (2,6)

(c) Colors based on coloring at both levels

Fig. 7: Multi-level coloring algorithm.

4. Performance Testing. We examine the performance of our HP algorithms first for general lattices and then for several more general matrices.

For lattices of arbitrary sizes we examine the performance of Algorithm 5 in terms of cost and effectiveness. First, we confirm that the increased cost of the algorithmic extensions is not excessive. Second, we show that the probing vectors produced by our algorithm provide better trace estimation than simply using a set of Hadamard vectors after the original algorithm in [13] exhausts all factors of 2.

We ran Algorithm 5 on a set of lattices with sizes that are powers of 2 so that our original HP algorithm can also be used. We forced Algorithm 5 to use the general method and not to revert to the original binary method for factors of 2. The results shown in Table 1 indicate that the increased computation over the original algorithm is reasonable, given the short running times involved even for very large lattices. In practice, many lattices only have a couple of factors that are not powers of 2 so the timings in Table 1 represent a worst case scenario. At the same time, the algorithm is embarrassingly parallel, since each lattice point can be reordered independently. Given the low runtimes compared to the cost of solving the linear systems during probing, we have not investigated this option. Finally, we observe that the dimensionality of

	Original Method	Extended Method	
Lattice	**Time(sec)**	**Time(sec)**	**Slowdown**
8^4	0.002	0.005	2.5
16^4	0.027	0.069	2.6
32^4	0.330	1.082	3.8
4096^2	5.493	14.048	2.6
256^3	4.782	15.413	3.2
64^4	5.290	19.561	3.7

Table 1: Table showing run times for Algorithm 5 applied to all nodes in the shown lattices, compared to the original HP method. Results obtained from implementing algorithm in C on an MacBook Pro i9 clocked at 2.9 GHz.

the lattice does not impact the performance of the algorithm.

Next, we examine the effectiveness of our algorithm on estimating the trace for a model 2D lattice problem with each dimension of size $2^2 3^2 5$. Since this matrix is not invertible, we shift it by adding in a term of $0.1\mathcal{I}$. We compare it with our multi-level merge algorithm, as well as Hadamard. When using Hadamard we use our original hierarchical probing method until the powers of two factors are exhausted. At this point we begin using Hadamard vectors in their natural ordering inside each color block, with zero support in other colors. Although these vectors continue to remove diagonals from each block, they do not do this in a way that completely eliminates certain distances. Still this is often an improvement to simply using random vectors in each block.

As we can see in Fig. 8, the new algorithm does offer a significant performance increase, providing a much better trace estimate than Hadamard. Note that the circles for the Hadamard method start at color 48 since the two methods are identical until that point. We also note some interesting results from our multi-level algorithm. This 2D lattice is a difficult case for the multi-level approach since Hierarchical Probing knows a-priori all the connections at every distance, while the multi-level algorithm must compute them. Despite this, our algorithm performs reasonably well, providing acceptable results at every level. This is encouraging because it suggests that when we proceed to general matrices, our algorithm may be effective.

We examine the effectiveness of our multi-level algorithm for arbitrary matrices by comparing its accuracy versus classical probing [14] and versus statistical methods [1]. The two test cases we have selected are the synthetic covariance matrix of [14] and the synthetic uncertainty quantification matrix of [15]. To examine the cost we report the memory usage of our method versus classical probing. Classical probing needs to store two matrices, A^k and A^{2k}, of the same size N and increasing density. The alternative of using depth first search on A to produce the distance-k coloring is computationally prohibitive for large k. Our method requires the storage of two matrices, A and A^2, at any level but memory between levels can be reused. As levels increase, the size of these matrices decreases. However, since we merge distance-2 neighborhoods the density of these matrices increases. Finally, we can adjust the implementation of our method to avoid storing A^2 by using depth first search to find the distance-2 neighborhoods and build A_c at a slightly larger cost. In this case, the method needs both A and A_c. Thus, we compare the total memory usage of the three methods using as a measure the total number of non-zero elements stored at any

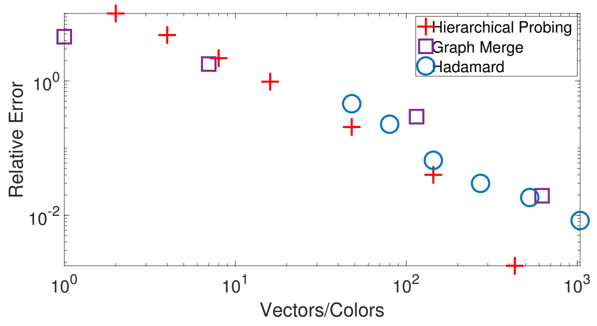


Fig. 8: Comparison of our extended hierarchical probing method against our multi-level merging algorithm and Hadamard. We run our test on a 2x2x3x3x5 two dimensional lattice which has been shifted by $0.1\mathcal{I}$ to make the matrix invertible. We apply Hadamard after the powers of two have been exhausted by our original algorithm. We report the trace estimate whenever a color completes. For the common factors of two, the methods are the same, but once these are exhausted the improved method has much lower error. As noted in section 2, the number of additional colors increases as $\prod_i \mathbf{F}_i^d$, where $\mathbf{F}$ are the common factors and d is the dimension of the lattice.

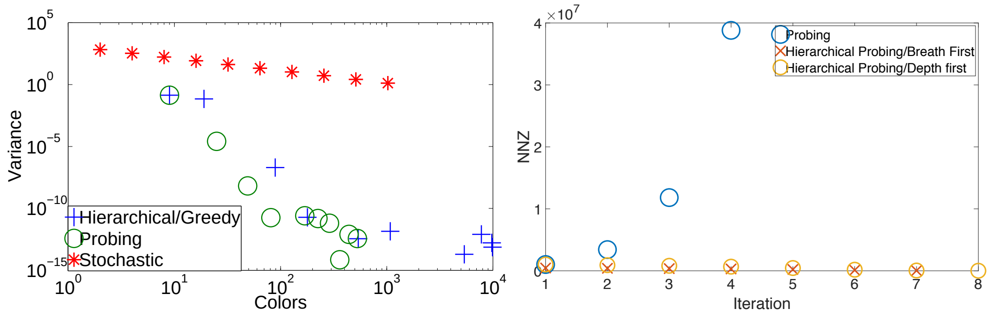
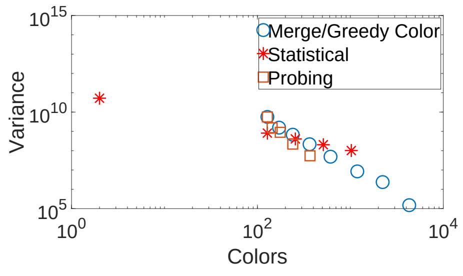


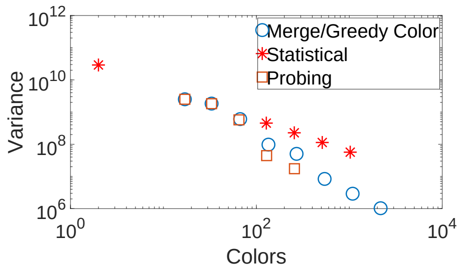
Fig. 9: Experiments on the covariance matrix of [14]. Left: Comparison of hierarchical probing against classical probing and statistical approaches. Right: Comparing memory usage of classical probing, explicitly forming A^{2k}, against two implementations of our algorithm at each iteration (level k).

point during execution. Within a factor this also determines their relative runtime performance.

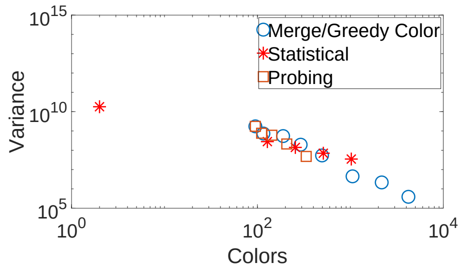
We start by examining the covariance matrix of [14], using the same parameters suggested in [14]. Classical probing works extremely well on this matrix so here we hope simply to be close to probing in performance while being more memory efficient. In the left graph of Fig. 9 we compare the variance of classical probing, our general HP probing method, and the Hutchinson Monte Carlo method. For this test case the performance of HP is similar to that of classical probing. In the right graph of Fig. 9 we compare the memory usage of the three probing variants, observing that both of HP variants require less memory than classical probing for the same level k (iteration). Note that all algorithms annihilate connections of distance at least $2k$.



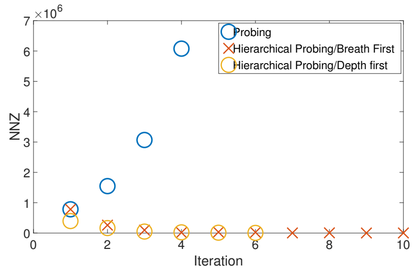
(a) $\lambda = .1$



(b) $\lambda = 1$



(c) $\lambda = 10$

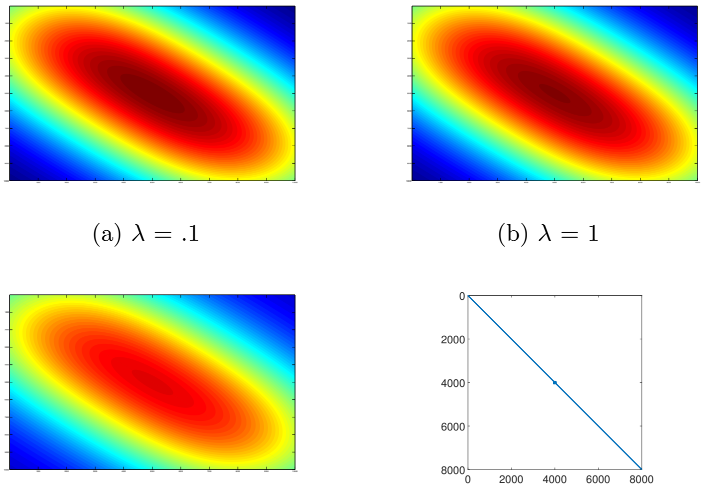


(d) Memory usage

Fig. 10: Experiments on the uncertainty quantification matrix. (a)(b)(c): Variance reduction with hierarchical probing, classical probing, and statistical methods for different values of λ. (d): Memory usage of various methods at each level (iteration).

Finally, we examine the synthetic uncertainty quantification matrix of [15]. This matrix is generated by the equation $A(A^tA+\lambda^2D^tD)^{-1}A^t$ and is of dimension 8000. Here D is a finite-difference operator, and we take A to be a Gaussian kernel $\propto e^{\frac{-(x-y)}{2\gamma^2}}$. We fix $\gamma = .08$ as in [15], and vary λ. Since this is a dense matrix, we need to create a sparse pattern which we can use to create the probing vectors. While an optimal solution is unclear, we have developed a heuristic which has yielded good results in practice. We note two major features of the matrix as seen in Fig. 11. There is a clear banded structure to the matrix, as well as a concentration of large values close to the center. To capture both features of this structure, we drop all but a small percent of the largest values of the matrix. This leaves a disconnect matrix, that we cause to be connected by adding in a banded matrix. This structure can be seen in Fig. 11(d). Here we obtain the central structure of the matrix by dropping all but the smallest .1 percent of the matrix, and the banded structure by creating a matrix with a bandwidth of 32. We would expect it to be difficult to outperform classical probing on this type of matrix, so our goal is to remain competitive with classical probing, while significantly reducing our memory usage. As can be seen in Fig. 10, this is indeed the case. While we do not surpass the variance reduction achieved by probing, we do remain competitive while achieving substantial memory reductions at the higher hierarchical levels that are needed to surpass the statistical methods.

5. Conclusion. We have provided several extensions to the algorithm for hierarchically coloring and probing lattices. By formalizing the use of sublattices in the



(a) $\lambda = .1$

(b) $\lambda = 1$

(c) $\lambda = 10$

(d) Sparsified structure

Fig. 11: Structure and magnitude of the elements of the uncertainty quantification matrix as λ increases. The sparsified matrix is close to a narrow banded matrix.

algorithm, we have made the algorithm easier to reason about. This allowed us to improve its flexibility, enabling the algorithm to handle lattices with dimensions of arbitrary sizes, as long as these sizes share common prime factors. These improvements come at minimal computational cost, and retain the ease of parallelization that was an attractive feature of the original algorithm. Moreover, we have introduced a method for generating probing vectors regardless of whether the colors split evenly in the hierarchy. This method applies to any color hierarchy produced from lattices or for a general matrix. With this tool in hand, we have provided an algorithm for producing such a hierarchical coloring for general graphs. Our tests show that our method provides accuracy that is comparable to classical probing, while using far less memory and thus operations.

6. Appendix.

Lemma 2.1

Proof. Since $\cup_{\mathbf{c}}\mathcal{L}(b\mathbf{I})_{\mathbf{c}} = \mathcal{L}(\mathbf{I})$, $\mathbf{p} = b\mathbf{x} + \mathbf{c}$. Then $\mathit{color}(b\mathbf{x} + \mathbf{c} \bmod b) = \mathit{color}(\mathbf{c})$, and since $\mathbf{p},\ \mathbf{c} \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}}$, both have the same color. □

Lemma 2.3

Proof. Let $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{L}(\mathbf{I})$, with $\|\mathbf{p}_1 - \mathbf{p}_2\|_1 = 1$. This means they share all but one coordinate, say the i-th. If their connection is not due to the toroidal connection on the boundary (wrap-around connection), their i-th coordinate will differ by one. Thus, $|\mathbf{p}_1(i) - \mathbf{p}_2(i)| \bmod b = 1$, implying that $\mathit{color}(\mathbf{p}_1) \neq \mathit{color}(\mathbf{p}_2)$. If both points lie on a boundary and connect via the toroidal property, then $|\mathbf{p}_1(i) - \mathbf{p}_2(i)| = (b - 1) - 0 \bmod b \not\equiv 0 \bmod b$, and therefore $\mathit{color}(\mathbf{p}_1) \neq \mathit{color}(\mathbf{p}_2)$. Since these are no other cases possible, the result holds. □

Lemma 2.5

Proof. Making use of the definition of mod for positive integers

$$k \bmod b = k - b\lfloor \frac{k}{b} \rfloor, \tag{24}$$

we proceed by induction on d. For the base case, $c_{1,b} = \{0 \dots b-1\}$, and $c_{2,b} = \{c_{1,b}, c_{1,b} + 1 \bmod b, \dots, c_{1,b} + b - 1 \bmod b\}$. Then by construction, every $c_{2,b}(ib) = 0 + i = c_{1,b}(i)$. Assume that $c_{d-1,b}(ib) = c_{d-2,b}(ib/b) = c_{d-2,b}(i)$. Then,

$$\begin{aligned}
c_{d,b}(ib) &= c_{d-1,b}(ib \bmod b^{d-1}) + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{ by 13})\\
&= c_{d-2,b}(\tfrac{ib \bmod b^{d-1}}{b}) + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by the I.H.})\\
&= c_{d-2,b}(\tfrac{ib - \lfloor \frac{ib}{b^{d-1}} \rfloor b^{d-1}}{b}) + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by 24})\\
&= c_{d-2,b}(i \bmod b^{d-2}) + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by 24})\\
&= c_{d-1,b(i)} - \lfloor \tfrac{i}{b^{d-2}} \rfloor + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by 14})\\
&= c_{d-1,b(i)} - \lfloor \tfrac{i}{b^{d-2}} \rfloor + \lfloor \tfrac{i}{b^{d-2}} \rfloor \bmod b & &\\
&= c_{d-1,b(i)}.
\end{aligned}$$

□

Lemma 2.6

Proof. We proceed by induction on d. For the base case, by construction we have $c_{2,b}(ib+q) \bmod b = c_{1,b}(i) + q \bmod b$. Then, $c_{2,b}(ib+q) - q \bmod b = (c_{1,b}(i) + q \bmod b - q) \bmod b = c_{1,b}(i) \bmod b = c_{1,b}(i) = c_{2,b}(ib)$ (by Lemma 2.5). We now assume $c_{d-1,b}(ib) = c_{d-1,b}(ib + q \bmod b^{d-1}) - q \bmod b$. Then,

$$\begin{aligned}
c_{d,b}(ib) &= c_{d-1,b}(ib \bmod b^{d-1}) + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by 13})\\
&= c_{d-1,b}(ib + q \bmod b^{d-1}) - q + \lfloor \tfrac{ib}{b^{d-1}} \rfloor \bmod b & &(\text{by the I.H.})\\
&= c_{d,b}(ib + q \bmod b^{d}) - q \bmod b & &(\text{by 14})\\
&= c_{d,b}(ib + q) - q \bmod b. & &(\text{since } ib + q < b^{d})
\end{aligned}$$

□

Lemma 2.7

Proof. Because of Lemma 2.6 when any b-tuplet of indices $(bi, bi+1, \dots, bi+b-1)$, is considered, the number of nodes in every color increases by 1. Since Algorithm 3 will send the b-th color to the b^{d-1}-th section, the equation holds. □

Theorem 2.8

Proof. Since $i \le b^m$, we can apply Lemma 2.5 recursively,

$$c_{d,b}(ib^{d-m}) = c_{d-1,b}(ib^{d-m-1}) = c_{d-2,b}(ib^{d-m-2}) = \dots = c_{m,b}(i).$$

Using this and Lemma 2.7 we have

$$\left\lfloor \frac{\text{Perm}_d(ib^{d-m})}{b^{d-m}} \right\rfloor = \left\lfloor \frac{\lfloor \frac{ib^{d-m}}{b} \rfloor + c_{d,b}(ib^{d-m})b^{d-1}}{b^{d-m}} \right\rfloor = \left\lfloor \frac{i}{b} + c_{m,b}(i)b^{m-1} \right\rfloor$$
$$= \lfloor \frac{i}{b} \rfloor + c_{m,b}(i)b^{m-1} = \text{Perm}_m(i).$$

□

REFERENCES

[1] H. Avron and S. Toledo, *Randomized algorithms for estimating the trace of an implicit symmetric positive semi-denite matrix*, Journal of the ACM, 58 (2011), p. Article 8.

[2] Z. Bai, M. Fahey, G. H. Golub, *Some large-scale matrix computation problems*, J. of Comput. and Appl. Math., 74 (1-2), 1996, p. 71–89.

[3] C. Bekas, E. Kokiopoulou, and Y. Saad, *An estimator for the diagonal of a matrix*, Appl. Numer. Math., 57 (2007), pp. 1214-1229.

[4] M. Benzi, P. Boito, and N. Razouk, *Decay properties of spectral projectors with applications to electronic structure*, SIAM Rev., 55 (2013), pp. 364

[5] S. Bernardson, P. McCarty, and C. Thron, *Monte Carlo methods for estimating linear combinations of inverse matrix entries in lattice QCD*, Comput. Phys. Commun., 78 (1994), pp. 256–264.

[6] E. Estrada, N. Hatano, *Statistical-mechanical approach to subgraph centrality in complex networks*, Chemical Physics Letters, 439, (5/07), pp. 247-251.

[7] A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.

[8] D. Goldfarb, Ph. L. Tont, *Optimal Estimation of Jacobian and Hessian Matrices That Arise in Finite Difference Calculations*, Mathematics of Computation, Vol. 43, No. 167 (Jul., 1984), pp. 69-88

[9] S. W. Golomb and L. D. Baumert, *The search for Hadamard matrices*, Amer. Math. Monthly, 70 (1963) pp. 12-17.

[10] M. F. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, J. Commun. Statist. Simula., 19 (1990), pp. 433–450.

[11] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third Edition. Addison-Wesley, (1997), pp. 65-66, 208-209, and 290.

[12] F. Rouet, *Calcul partiel de l'inverse d'une matrice creuse de grande taille - application en astrophysique*, Master's Thesis (10/09), Institut National Polytechnique de Toulouse, 2009.

[13] A. Stathopoulos, J. Laeuchli, and K. Orginos, *Hierarchical probing for estimating the trace of the matrix inverse on toroidal lattices* , SIAM J. Sci. Comput.,35(5) (2013), pp. 299–322.

[14] J.M. Tang and Y. Saad, *A probing method for computing the diagonal of a matrix inverse*, Numerical Linear Algebra and its Applications, 19, 3, (2012), 485–501.

[15] L. Tenorio, F. Andersson, M. V.De Hoop, and P. Ma, *Data Analysis for uncertainty quantification of inverse problems*, Proceedings of the Project Review, Geo-Mathematical Imaging Group (1) (2011) pp.1-20