Lawrence Berkeley National Laboratory

Title
Rapid generation of optimal generalized Monkhorst-Pack grids

Permalink
https://escholarship.org/uc/item/87r651ht

Authors
Wang, Yunzhe
Wisesa, Pandu
Balasubramanian, Adarsh
et al.

Publication Date
2021-02-01

DOI
10.1016/j.commatsci.2020.110100

Peer reviewed

Rapid Generation of Optimal Generalized Monkhorst-Pack Grids

Yunzhe Wang[1], Pandu Wisesa[1], Adarsh Balasubramanian[1], Shyam Dwaraknath[2] and Tim Mueller[1,]*

[1] Department of Materials Science and Engineering, Johns Hopkins University, Baltimore, Maryland

21218, USA

[2] Lawrence Berkeley National Laboratory, Berkeley, California, 94720, USA

Abstract

Computational modeling of the properties of crystalline materials has become an increasingly important

aspect of materials research, consuming hundreds of millions of CPU-hours at scientific computing

centers around the world each year, if not more. A routine operation in such calculations is the

evaluation of integrals over the Brillouin zone. We have previously demonstrated that performing such

integrals using generalized Monkhorst-Pack k-point grids can roughly double the speed of these

calculations relative to the widely-used traditional Monkhorst-Pack grids. However the generation of

optimal generalized Monkhorst-Pack grids is not implemented in most software packages due to the

computational cost and difficulty of identifying the best grids. To address this problem, we present new algorithms that allow rapid generation of optimal generalized Monkhorst-Pack grids on the fly. We demonstrate that the grids generated by these algorithms are on average significantly more efficient than those generated using existing algorithms across a range of grid densities. For grids that correspond to a real-space supercell with at least 50 angstroms between lattice points, which is sufficient to converge density functional theory calculations within 1 meV/atom for nearly all materials, our algorithm finds optimized grids in an average of 0.19 seconds on a single processing core. To facilitate the widespread adoption of this approach, we present new open-source tools including a library designed for integration with third-party software packages.

1. Introduction

Computational materials research has become increasingly vital in probing the properties of crystalline materials, especially in screening materials at a large scale to accelerate material discoveries for a wide range of applications. A routine operation for such calculations across a variety of computational methods is the evaluation of integrals over the Brillouin zone, which can be approximated by discretely sampling the Brillouin Zone at a set of points known as k-points. Many popular computational materials simulation packages generate k-points using the traditional Monkhorst-Pack scheme,[1] which creates regular k-point grids with lattice vectors that are integer fractions of a particular set of reciprocal lattice vectors. We have demonstrated in our previous work that the number of symmetrically irreducible k-points, and hence the computational cost of most methods that rely on k-point sampling, can be reduced by roughly a factor of two by generalizing the Monkhorst Pack scheme so that the grids do not need to be aligned with a particular set of reciprocal lattice vectors and selecting the optimal generalized grid.[2] The benefits of using generalized grids can be understood by considering that the set of generalized k-point grids is a superset of traditional Monkhorst Pack grids, providing far more options for selecting the optimal grid. Other researchers have since found similar results.[3, 4]

Calculating the properties of crystalline materials consumes hundreds of millions of CPU-hours at supercomputing centers around the world each year, if not much more. (A single high-throughput project, the Materials Project, spends more than 100 million CPU hours per year calculating the properties of crystalline materials.) Given that modern high-performance computing resources cost about US\$ 0.0255 per CPU hour[1] or more,[5] we conservatively estimate

---

[1] The CPU price is the latest listed value for the standard AWS machine type a1.medium with 2GB memory.

that the use of generalized Monkhorst Pack grids in place of traditional grids has the potential to save researchers millions of U.S. dollars per year in computing costs.

Some of the ideas behind the generalized k-point grids had been proposed by Froyen and Moreno and Soler decades ago,[1, 6] but they have not been widely adopted primarily due to the computational challenge of identifying the best generalized grid for a given calculation. The main challenge is that the number of possible generalized k-point grids grows rapidly with the number of k-points in the grid (Section 2 of the Supplementary Information), making it difficult to identify which grid is most efficient.[2, 7] For example, there are 54,156,102 regular grids that contain 4,000 k-points, a typical density for calculations on elemental metals. Identifying the optimal grid requires identifying which among these candidates is expected to provide a sufficiently accurate estimate of the integral with the fewest symmetrically irreducible k-points. The problem is made more challenging by the fact that it is generally necessary to search over many different k-point densities to find the optimal grid.

In our previous work we addressed these problems by creating a free, internet-accessible k-point grid server, backed by a database of pre-calculated generalized grids, that rapidly returns an efficient grid (typically the most efficient grid) for a given calculation.[2] To date, this server has delivered more than half a million grids to users outside our research group. In the years since our previous work was published there has been increasing interest in the generation and use of generalized k-point grids[4, 8-28] and how they may be used in popular software packages.[28] Yet despite the increasing interest in the use of regularized grids, most common software packages do not yet implement an efficient method for identifying highly efficient generalized grids, due largely to the lack of publicly available algorithms and tools for doing so.

To enable more widespread use of generalized Monkhorst-Pack k-point grids and fully realize their potential for accelerating computational materials research, we have developed an open-source library for grid generation, kpLib, that is designed for integration with existing software packages without significantly increasing the size of their software distribution. This library is based on novel algorithms, described in this manuscript, that greatly accelerate grid generation. These algorithms include a method for significantly reducing the number of candidate superlattices to be evaluated by transforming the problem from an enumeration of 3D superlattices to an enumeration 2D superlattices with a finite set of allowed stackings. We have also developed an open-source standalone tool for generalized k-point grid generation, the K-Point Grid Generator. This tool has the same functionality as the K-Point Grid Server, but it can be used on computing nodes that do not have network access to the K-Point Grid Server. Additional algorithms for the K-Point Grid Generator and its implementation are described in detail in section 5 and section 6.2 of the supplementary information.

To illustrate the performance of kpLib, we present benchmarks on structures randomly selected from the Inorganic Crystal Structure Database.[29] Our benchmarks demonstrate that at a grid density sufficient to converge calculated energies on nearly all crystalline materials within 1 meV / atom, kpLib identifies the optimal grid in less than half a second on average, and in under five seconds for grids that are eight times as dense. We further demonstrate that on average our algorithm finds grids with significantly fewer irreducible k-points than an alternative algorithm for generating generalized Monkhorst-Pack grids recently developed by Hart and co-workers.[30, 31]

In the following sections, a detailed explanation of the new algorithms is provided, and the implementation of kpLib is briefly discussed. Various benchmarks of the speed of the algorithms

and quality of the resulting grids are then provided. Additional comparisons between kpLib and the *K*-Point Grid Generator, along with detailed descriptions of other algorithms used by these software packages, are provided in the supplementary information.

2. Algorithms

2.1 Background and notation

Monkhorst-Pack grids are used to approximate the value of an integral over the Brillouin zone by sampling reciprocal space on a regular grid of *k*-points, where the coordinates of the *k*-points are given by

$$\mathbf{k} = \frac{n_1}{m_1}\mathbf{b}_1 + \frac{n_2}{m_2}\mathbf{b}_2 + \frac{n_3}{m_3}\mathbf{b}_3 + \mathbf{s},$$

$$n_1 = 0\ldots m_1 - 1,\ n_2 = 0\ldots m_2 - 1,\ n_3 = 0\ldots m_3 - 1$$

(1)

where m_1, m_2, and m_3 are positive integers, $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ are reciprocal lattice vectors, and $\mathbf{s}$ represents a shift vector that moves the grid away from the origin (known as the Γ point in reciprocal space). There exists a mapping between each regular *k*-point grid and a real-space superlattice that defines the Born-von Karman boundary conditions for the periodicity of the wave functions.[32, 33] The superlattice corresponding to the *k*-point grid defined by equation (1) is given by

$$\left(\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\right)^T = \mathbf{M}(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)^T$$

(2)

where $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ represent the real-space primitive lattice vectors, $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_3$ represent the lattice vectors of the superlattice, and the transformation matrix $\mathbf{M}$ is equal to

$$\mathbf{M} = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}. \tag{3}$$

The reciprocal primitive lattice vectors share an analogous relationship with those of the reciprocal superlattice. The reciprocal lattice vectors of a direct lattice are calculated by

$$[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]^T = [\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3]^{-1} \tag{4}$$

where the vectors share the same definition as in equations (1) and (2). Similarly, the primitive reciprocal lattice vectors of the superlattice can be obtained by

$$[\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]^T = [\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3]^{-1} \tag{5}$$

where $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ are the reciprocal lattice vectors corresponding to the direct superlattice. Substituting equations (4) and (5) into equation (2), the following relationship can be derived:

$$[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]^T = \mathbf{M}^T [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]^T. \tag{6}$$

The matrix multiplication order implies that the row vectors of the matrix $\mathbf{M}^T$ contain the coordinates of the vectors $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ in the basis of $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}$.

In terms of the matrix $\mathbf{M}$, equation (1) can be written as

$$
\begin{aligned}
\mathbf{k} &= (n_1, n_2, n_3) \left(\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}^{-1} \right)^T (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)^T + \mathbf{s} \\
&= (n_1, n_2, n_3) (\mathbf{M}^{-1})^T (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)^T + \mathbf{s} \\
&= (n_1, n_2, n_3) [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]^T + \mathbf{s}
\end{aligned} \tag{7}
$$

Therefore, the set of vectors $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}$ are a generating basis of the k-point grid. As shown in equation (7), the traditional Monkhorst-Pack scheme uses a diagonal matrix $\mathbf{M}$, which is

equivalent to the constraint that the *k*-point grids are aligned with the reciprocal lattice vectors. However Froyen has pointed out that this constraint is not necessary,[6] and we have previously demonstrated that much more efficient grids can be generated if the Monkhorst-Pack approach is generalized by relaxing this requirement.[2] The resulting generalized *k*-point grids, as shown by Moreno and Soler, can always be represented as standard Monkhorst-Pack grids provided a suitable set of reciprocal lattice vectors are chosen.[7] Mathematically, this is equivalent to perform a diagonal decomposition on the integer matrix $\mathbf{M}$ by unimodular matrices

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1} \tag{8}$$

and transforming the reciprocal lattice vectors to an equivalent set by plugging it into equation (6):

$$\begin{aligned}
\left[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\right]^T &= (\mathbf{U}\mathbf{D}\mathbf{U}^{-1})^T \left[\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\right]^T \\
(\mathbf{U}^T \left[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\right]^T) &= \mathbf{D}(\mathbf{U}^T \left[\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\right]^T) \\
\left[\mathbf{b}'_1, \mathbf{b}'_2, \mathbf{b}'_3\right]^T &= \mathbf{D}\left[\mathbf{d}'_1, \mathbf{d}'_2, \mathbf{d}'_3\right]^T
\end{aligned} \tag{9}$$

where $\mathbf{b}'_1$, $\mathbf{b}'_2$, and $\mathbf{b}'_3$ are the reciprocal lattice vectors that diagonalize the generating matrix. Thus generalized Monkhorst Pack *k*-point grids can be used for all of the same types of calculations that traditional Monkhorst-Pack grids are used for.

Equations (2) and (7) demonstrate that the search for optimal generalized *k*-point grids can be accomplished by an iteration over real-space superlattices, specified by the matrix $\mathbf{M}$, and shift vectors, given by the vector $\mathbf{S}$. Since the quality of *k*-point grids are determined by the number of symmetrically irreducible *k*-points, all symmetries of structures should be preserved in the grids, which transfers to the requirements that the corresponding superlattices must also be symmetry-preserving. In the following discussion, we use the symbols $r_{lattice}$, N_i, and N_T to represent, respectively, the minimum spacing between points on the a superlattice, the number of

symmetrically irreducible *k*-points, and the number of total *k*-points in the Brillouin zone. N_T is also then the number of primitive cells in a unit cell of the corresponding real-space superlattice (aka the "size" of the superlattice), and is given by the absolute value of the determinant of $\mathbf{M}$.

2.2 A New Algorithm for Dynamically Generating Generalized K-Point Grids

Although the benefits of using generalized *k*-point grids are well-established,[2-4, 24] they have not yet been widely implemented in common software packages due primarily to the challenge in implementing an algorithm for efficiently generating them. To address this problem and facilitate the generation of generalized *k*-point grids in common materials software packages, we have developed a novel algorithm for rapidly and dynamically identifying a highly efficient generalized *k*-point grid. Unlike our previous approach, this algorithm does not make use of a database, allowing us to implement it in a lightweight, open-source library designed to be integrated with third-party software packages. Although the lack of a database reduces the speed of grid generation (see section 4.1), we expect the optimized dynamic generation algorithm we present here to be sufficiently fast for most practical applications. We have also released a standalone open-source tool that provides additional functionality and makes use of a database, using algorithms described in section 3 of the supporting information.

The dynamic grid generation method starts with three parameters describing the input structure:

1. The real-space primitive lattice vectors, $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$.

2. The real-space conventional lattice vectors, $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$, where at least one of the vectors is orthogonal to the other two for all but triclinic systems.

3. The group of point symmetry operations, $\{\mathbf{R}\}$, that the k-point grid (and real-space superlattice) should preserve. These point symmetry operations can be generated by removing translation from all the operations in the real-space crystallographic space group, resulting in a symmorphic space group. If the system has time reversal symmetry, then the reciprocal-space band structure will have inversion symmetry even if the real-space crystal does not. In this case, inversion and any additional operators required to complete the group should be added if they are not already present.

The algorithm then searches for the k-point grid that minimizes N_i while satisfying the following two constraints:

1. $r_{lattice}$ for the corresponding superlattice not smaller than r_{min} (a value provided by the user),

2. N_T is greater than or equal to N_{min} (another value provided by the user).

We start by determining a lower bound for N_T, which we call, N_{lower}. It is the larger value of N_{min} and the minimum size that any superlattice can have with while satisfying $r_{lattice} \geq r_{min}$:

$$N_{lower} = \max\left(N_{min}, \left\lfloor \frac{\sqrt{2}}{2} r_{min}^{3} \middle/ V_p \right\rfloor \right) \tag{10}$$

where V_p is the volume of the primitive cell, $\frac{\sqrt{2}}{2} r_{min}^{3}$ is the volume of a unit cell in a face-centered cubic (fcc) lattice for which the distance between lattice points is r_{min}, $\lfloor x \rfloor$ is the floor operation that returns the largest integer no greater than the argument x. Equation (10) can be

justified by considering that fcc structures maximize the packing density for rigid spheres[34] and thus $\sqrt{2}/2\, r_{min}{}^3$ is the minimum unit cell volume for a superlattice for which $r_{lattice}$ is at least r_{min}.

The search for optimal superlattices starts with lattices of size N_{lower} and generates symmetry-preserving superlattices using an algorithm to be introduced in section 0. For each symmetry-preserving superlattice, the scheme checks whether $r_{lattice}$ is smaller than r_{min} and discards it if it is. When the first superlattice for which $r_{lattice} \geq r_{min}$ is found, its corresponding k-point grid is kept as the initial "best grid", and the scheme can determine an upper limit for the search, N_{upper}:

$$N_{upper} = N_i \times N_{sym} \tag{11}$$

where N_{sym} is the number of unique point symmetry operations for the system, as provided in the third input parameter listed above. Any superlattices with $N_T \geq N_{upper}$ would necessarily have more irreducible k-points than that of the initial best grid. If at some point a superlattice with N_i smaller than that of the best known grid is found, the best grid is updated to this newly found one and the value of N_{upper} is adjusted accordingly. When two k-point grids have the same N_i, the scheme favours the one with a larger $r_{lattice}$ in the corresponding superlattice. If $r_{lattice}$ of both superlattices also tie, the scheme chooses the one with a larger N_T. The search ends when the upper limit of the sizes of superlattices is reached. Figure 1 summarizes the steps of the scheme.
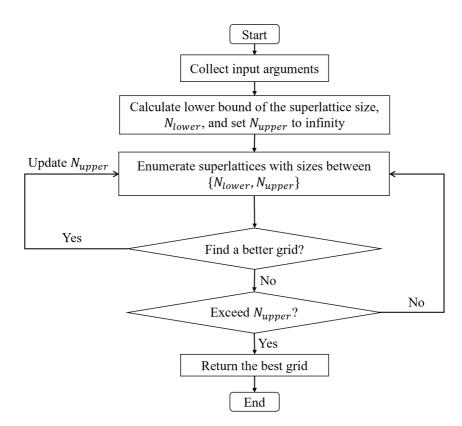
Figure 1. A diagram summarizes the workflow of the dynamic grid generation algorithm.

2.3 Algorithms for Efficient Enumeration of Symmetry-Preserving Superlattices

Enumeration of all symmetry-preserving superlattice is computationally expensive and has been identified as the main hurdle of applying generalized *k*-point grids in calculations of properties of crystalline materials.[2, 7] Morgan et al. have presented an algorithm for accelerating the enumeration of symmetry-preserving lattices for a given lattice size by expressing the primitive lattice in Niggli-reduced form.[31] For each of the 44 distinct Niggli bases, they have determined symmetry-based constraints on the entries of $\mathbf{H}$ that can be used to reduce the number of possible lattices that must be considered. We have developed an approach that similarly iterates over symmetry-preserving lattices, with two key differences: it does not rely on Niggli reduction, which reduces the complexity of the code and increases the ease of

implementation, and it is optimized for grid selection based on r_{min}, which has been shown to work well as a descriptor of k-point density both in theory[2] and in practice.[2, 4] In our benchmarks, we demonstrate that the algorithms presented here generally return more efficient grids than the those generated using the method of Morgan et al.[31]

2.3.1 Hermite normal form and symmetry-preserving lattices

It is possible for two different matrices $\mathbf{M}$ to represent the same superlattice; i.e. the rows of each matrix could represent a different choice of vectors used to represent the lattice. For the purpose of enumerating over lattices we express the transformation matrix $\mathbf{M}$ in Hermite normal form, a triangular form which uniquely defines a superlattice.[35, 36] We shall use $\mathbf{H}$ to represent the Hermite normal form of a general matrix $\mathbf{M}$.

Efficient k-point grids will generally have symmetry-preserving lattices, which are invariant with respect to the symmetry operations of the system. Hermite normal form provides a convenient way to test whether a superlattice is symmetry-preserving by generating the Hermite normal forms for the original matrix $\mathbf{M}$ and all matrices generated by applying the symmetry operations of the system to $\mathbf{M}$. If all of the generated Hermite normal forms are the same, the lattice is symmetry-preserving.

2.3.2 Enumeration Algorithm for Crystal Systems Other than Triclinic

We start by considering systems that are not triclinic. For such systems at least one of the conventional lattice vectors must, by the symmetry of the system, be perpendicular to the other two. For simplicity, our only requirement is that such a vector be listed third, as $\mathbf{c}_3$.

The key to our approach is the recognition that for systems that are not triclinic, any regular three-dimensional lattice consists of layers of identical two-dimensional lattices that are normal

to $\mathbf{c}_3$. Each two-dimensional lattice may be shifted from the one below it by a constant shift vector that is parallel to its lattice plane, and for symmetry-preserving lattices only a finite set of shift vectors are allowed. This decomposition helps quickly rule out superlattices that break symmetries without applying linear algebra to check them. For example, if there is a twofold rotational axis parallel to $\mathbf{c}_3$, then this axis may only pass through points in the two-dimensional lattice formed by linear combinations of half lattice vectors (Table 1). Any other shift would result in a lattice that is not symmetry preserving, as symmetry operations could transform lattice points to non-lattice points. Similarly, if there is a mirror plane perpendicular to $\mathbf{c}_3$, then either the mirror plane must be at the mid-point between two layers, in which case no shift is allowed, or it must pass through one of the layers, and again only the shifts shown in Table 1 are allowed. This concept is illustrated in two dimensions in Figure 2. Similar sets of shifts may be derived for three-fold rotational axes (Table 1).

A high-level summary of our algorithm for enumerating symmetry-preserving lattices is then as follows:

1. Determine all pairs of factors of the total lattice size. In each pair, the first factor represents the size of the supercell in each two-dimensional layer and the second represents the number of layers in each three-dimensional supercell.

2. For each pair of factors, enumerate all symmetry-preserving two-dimensional lattices (in Hermite normal form) with the required size.

3. Combine each two-dimensional lattice with each allowed shift to create a candidate three-dimensional lattice.

4. Verify that the three-dimensional lattice is symmetry-preserving.

Crystal System	Shift vectors in the basis of $\{\mathbf{c}_1,\mathbf{c}_2\}$ in real space	Shift vectors of the Γ point in the basis of $\{\mathbf{d}_1,\mathbf{d}_2,\mathbf{d}_3\}$ as defined in equation (5)
Cubic, Tetragonal, Orthorhombic, Monoclinic	[0.0, 0.0], [0.0, 0.5], [0.5, 0.0], [0.5, 0.5]	[0.0,0.0,0.0], [0.0,0.0,0.5], [0.0,0.5,0.0], [0.5,0.0,0.0], [0.5,0.5,0.0], [0.5,0.0,0.5],
Hexagonal, Trigonal	[0.0, 0.0], [1/3, 0.0], [0.0, 1/3], [0.0, 2/3], [2/3, 0.0], [1/3, 1/3], [2/3, 2/3], [1/3, 2/3], [2/3, 1/3]	[0.0,0.5,0.5], [0.5,0.5,0.5]

Table 1. Possible displacements of lattice planes in real space in 2 dimensions, and of the Γ point in reciprocal space in 3 dimensions.
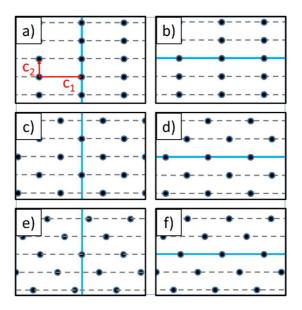

Figure 2. Two-dimensional examples of allowed and disallowed shifts. In all examples blue lines represent a mirror plane, black dots represent lattice points on real-space superlattice, and dashed lines show the different layers of lattice points that are orthogonal to $\mathbf{c}_2$. a), b), c), and d) show allowed shifts in which the mirror plane transforms every lattice point to another lattice point. In a) and b) there is zero shift, and in c) and d) the shift is half the vector $\mathbf{c}_1$. e) and f) show disallowed shifts.

This algorithm effectively reduces the problem of enumerating three-dimensional lattices to one of enumerating two-dimensional lattices, which significantly accelerates the search for symmetry-preserving lattices. Firstly, it drastically decreases the total number of 3-dimensional superlattices that need to be checked for symmetry preservation. Secondly, the symmetry groups in the 2-dimensional sublattice have fewer symmetry operations than the corresponding groups in 3 dimensions. Thirdly, a 2-dimensional matrix multiplication takes fewer elementary operations than a 3-dimensional one. We can even further accelerate the search by recognizing that if the number of layers is too small to satisfy the requirement that $r_{lattice} \geq r_{min}$, we can skip the enumeration of two-dimensional lattices and move on to the next set of factors. Similarly, if we ever determine that the $r_{lattice} < r_{min}$ for any two-dimensional layer, then we can stop evaluation of all lattices constructed from that layer and move onto the next two-dimensional lattice. We find that pre-screening the lattices for $r_{lattice}$ in this way significantly increases the speed of the algorithm when r_{min} is the limiting factor, as demonstrated by the benchmarking results in section 7.2 of the supplementary information.

The steps of the algorithm are shown in detail by the pseudocode in Figure 3. The term "maxZDistance" at line 6 defines the maximum possible length of the shortest vector parallel with $\mathbf{c}_3$ that superlattices can have while satisfying $r_{lattice} \geq r_{min}$. The function "symmetryPreserving($\mathbf{H}$, $\{\mathbf{R}\}$)" determines whether the set of symmetries is preserved in the given superlattice by checking the invariance of $\mathbf{H}$ after applying symmetries. Line 28 verifies that candidate lattices are superlattices of the primitive lattice after shifts in Table 1 are applied.

Algorithm 1 Fast Enumeration of Symmetry-Preserving Superlattices
___

Input:
  N_T - Size of superlattice. Also the total number of k-points in the grid.
  $\{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}\}$ - Conventonal lattice vectors.
  $\{\mathbf{R}\}$ - The point symmetry group to be preserved in superlattices.
  $\{\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}\}$ - Primitive lattice vectors.
  $\{\mathbf{s}\}$ - Array of possible 2D shift vectors.
  r_{min} - Minimum periodic distance specified by users.
Output:
  $\{\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}\}$ - Array of symmetry-preserving superlattices with $r_{lattice} \geq r_{min}$.
Initialization:
  $\{\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}\}$ - Empty array of symmetry-preserving superlattices.
  $\mathbf{H}$ - A 2×2 zero matrix.
  $\mathbf{M}$ - A 3×3 zero matrix.

1: find the point symmetry group, $\{\mathbf{R}^{2D}\}$, of the 2D sublattice $\{\mathbf{c_1}, \mathbf{c_2}\}$;
2: factorSets[][] ← sets of factorizations of N_T into 2 integral numbers;
3: **for** factors[] in factorSets[][] **do**
4:     primLayerSpacing = $\|\mathbf{c_3}\| \times V_p/V_c$;
5:     maxLayers = $isHexagonal()$? 3 : 2;
6:     maxZDistance = factors[2] $\times$ primLayerSpacing $\times$ maxLayers;
7:     **if** maxZDistance $< r_{min}$ **then**
8:         continue;
9:     $\{\{\mathbf{g_1}, \mathbf{g_2}\}\}$ ← initialize an empty array of 2 vectors;
10:    factorSets2D[][] ← sets of factorization of factors[1] into 2 integral numbers;
11:    **for** factors2D[] in factorSets2D[][] **do**
12:        $H_{11} = $ factors2D[1], $H_{22} = $ factors2D[2]
13:        **for** $i = 0$ **to** factors2D[1] - 1 **do**
14:            $H_{21} = i$;
15:            **if** not $symmetryPreserving(\mathbf{H}, \{\mathbf{R}^{2D}\})$ **then**
16:                continue;
17:            $\mathbf{g_1} = H_{11}\mathbf{c_1}$;
18:            $\mathbf{g_2} = H_{21}\mathbf{c_1} + H_{22}\mathbf{c_2}$;
19:            add $\{\mathbf{g_1}, \mathbf{g_2}\}$ to the array of 2D symmetry-preserving superlattices, $\{\{\mathbf{g_1}, \mathbf{g_2}\}\}$;
20:    **for** $\{\mathbf{g_1}, \mathbf{g_2}\}$ in $\{\{\mathbf{g_1}, \mathbf{g_2}\}\}$ **do**
21:        **if** $getMinDistance(\mathbf{g_1}, \mathbf{g_2}) < r_{min}$ **then**
22:            continue;
23:        **for** s in $\{\mathbf{s}\}$ **do**
24:            $\mathbf{g_3} = [s_1, s_2] \cdot [\mathbf{g_1}, \mathbf{g_2}]^T + (\text{factors}[2]/(V_c/V_p)) \cdot \mathbf{c_3}$;
25:            **if** $getMinDistance(\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}) < r_{min}$ **then**
26:                continue;
27:            $\mathbf{M} = [\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}]^T \cdot \left([\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}]^T\right)^{-1}$;
28:            **if** $\mathbf{M}$ contains non-integral elements **then**
29:                continue;
30:            **if** not $symmetryPreserving(\mathbf{M}, \{\mathbf{R}\})$ **then**
31:                continue;
32:            add $\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}$ to the array of symmetry-preserving superlattices;
33: **return** $\{\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}\}$;
___

Figure 3. Algorithm for fast enumeration of symmetry preserving superlattices for systems other than triclinic.

2.3.3 Enumeration Algorithm for the Triclinic Crystal System

The triclinic system doesn't benefit from the above algorithm since all its superlattices preserve the point symmetry operations of the primitive lattice, namely the identity operation and sometimes the inversion operation. For triclinic systems we accelerate the search for superlattices for which . . by again considering one dimension at a time. For each factor set, if $H_{11}|\mathbf{a}_1| < r_{min}$, the shortest distance between lattice points must be less than r_{min} and the factor set is not considered. Similarly, if the two dimensional lattice spanned by $H_{11}\mathbf{a}_1$ and $H_{21}\mathbf{a}_1 + H_{22}\mathbf{a}_2$ has $r_{lattice} < r_{min}$, we do not iterate over possible values of H_{31} and H_{32} as we already know the lattices will not satisfy the required constraint. The procedures are summarized as a pseudocode in Figure 4. The input lattice can be of any dimension up to three. We note that a similar approach can be used to accelerate any scheme based on iterating over lattices in HNF, such as the one developed by Morgan et al..[31]

Algorithm 2 Fast Enumeration of Symmetry-Preserving Superlattices for Triclinic Structures

---

Input:
  N_T - Size of the superlattice. Also the total number of k-points in the grid.
  r_{min} - Minimum periodic distance specified by users.
  $\{\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}\}$ - Primitive lattice vectors.
Output:
  $\{\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}\}$ - Array of symmetry-preserving superlattices with $r_{lattice} \geq r_{min}$.
Initialization:
  $\mathbf{H}$ - A 3×3 zero matrix.
  $\{\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}\}$ - Empty array of symmetry-preserving superlattices.

1: factorSets[][] $\leftarrow$ sets of integral factorizations of N_T into 3 numbers;
2: **for** $\{N_1, N_2, N_3\}$ in factorSets[][] **do**
3:     $\mathbf{H} \leftarrow$ put N_1, N_2, N_3 on diagonal positions;
4:     $\mathbf{g_1} = H_{11} \cdot \mathbf{a_1}$;
5:     **if** $\|\mathbf{g_1}\| < r_{min}$ **then**
6:         continue;
7:     **for** $H_{21} = 0$ **to** $N_1 - 1$ **do**
8:         $\mathbf{g_2} = H_{21} \cdot \mathbf{a_1} + H_{22} \cdot \mathbf{a_2}$;
9:         **if** $getMinDistance(\mathbf{g_1}, \mathbf{g_2}) < r_{min}$ **then**
10:             continue;
11:         **for** $(H_{31}, H_{32}) = (0, 0)$ **to** $(N_1 - 1, N_2 - 1)$ **do**
12:             $\mathbf{g_3} = H_{31} \cdot \mathbf{a_1} + H_{32} \cdot \mathbf{a_2} + H_{33} \cdot \mathbf{a_3}$;
13:             **if** $getMinDistance(\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}) < r_{min}$ **then**
14:                 continue;
15:             add $\{\mathbf{g_1}, \mathbf{g_2}, \mathbf{g_3}\}$ to the array of superlattices;
16: **return** the array of superlattices;

Figure 4. Algorithm for enumerating symmetry-preserving superlattices for triclinic system, accelerated by enforcing $r_{lattice} \geq r_{min}$ at each dimension.

2.4 Evaluating Shift Vectors

  K-point grids can be generated for each symmetry-preserving lattice using equation (7), where the matrix $\mathbf{H}$ can be used for $\mathbf{M}$. The only remaining unknown is the shift vector $\mathbf{S}$. When the shift vector has zero length, the *k*-point grid is called a Γ-centered grid, as it must contain the Γ point in reciprocal space as a grid point. Often the use of shift vectors with non-zero length results in more efficient grids, in part because avoiding the highly-symmetric Γ point allows for greater use of symmetry to reduce the number of symmetrically irreducible *k*-points.

For a shift to be guaranteed to result in a symmetry-preserving lattice, it must shift the origin to a point that has the full point group symmetry of the origin. For all symmorphic space groups, the only such points are located at linear combinations of full- or half-multiples of the primitive lattice vectors. Thus, we consider only the eight such unique combination of k-point grid generating vectors, $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}$, as candidate shift vectors (Table 1). In some cases (e.g. hexagonal systems), some of the shift vectors in Table 1 will not result in a symmetry-preserving grid. We identify and reject these when determining the number of irreducible k-points. As this occurs as soon as the first point that breaks symmetry is encountered, it comes with relatively little computational cost.

2.5 Algorithm for Fast Calculation of Symmetrically Irreducible K-points and K-point Weights

We select the optimal lattice based on the values of N_i, $r_{lattice}$, and N_T. The value of $r_{lattice}$ can be easily obtained from the superlattice vectors by Minkowski reduction, and N_T equals the absolute value of the determinant of the transformation matrix $\mathbf{M}$. However, calculating N_i for a k-point grid is a relatively expensive operation. An intuitive approach is to apply all the point symmetry operations to each k-point, $\mathbf{k}_i$, and compare the resulting coordinates with all the other k-points. If one of the transformed k-points, $\mathbf{k}_i'$, is translationally equivalent to one of the other k-points, $\mathbf{k}_j$, then the k-points $\mathbf{k}_i$ and $\mathbf{k}_j$ are symmetrically equivalent. However, this algorithm scales as $O(N_T^2)$, where N_T is the number of total k-points of a grid. As this operation

is applied to each of the k-point grids found by the algorithm in section 0, this intuitive but costly approach could easily become the major overhead of any k-point generation scheme.

We solve this complication by first recognizing that a unit cell in reciprocal space is a supercell of a regular k-point lattice, where the two lattices are related by equation (6). To avoid confusion with the Hermite normal form of $\mathbf{M}$, which we have labelled $\mathbf{H}$, we will refer to the Hermite normal form of the transformation matrix in reciprocal space, $\mathbf{M}^{\mathbf{T}}$, as $\mathbf{J}$ (in general, $\mathbf{J} \neq \mathbf{H}^{\mathbf{T}}$). The key to our approach is the recognition that it is possible to tessellate all of reciprocal space with supercells of size $J_{11} \times J_{22} \times J_{33}$ arranged periodically on the superlattice, where J_{11}, J_{22}, and J_{33} are the diagonal elements of $\mathbf{J}$ and each lattice point is a corner of the supercell. This is illustrated in two dimensions in Figure 5, but the same concept extends to any number of dimensions. The off-diagonal elements of $\mathbf{J}$ serve to shift each layer of supercells relative to the previous layer, so that the tessellation resembles stacked bricks. Within each of these supercells, the coordinates of a k-point can be expressed as:

$$\mathbf{r} + \left([k_1, k_2, k_3] + \mathbf{s} \right) \cdot [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]^{T} \tag{12}$$

where $\mathbf{r}$ is a lattice point on the reciprocal space lattice (blue dots in Figure 5), $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ are generating lattice vectors of the k-point lattice (also reciprocal primitive lattice vectors), k_1 is an integer from 0 to $J_{11} - 1$, k_2 is an integer from 0 to $J_{22} - 1$, and k_3 is an integer from 0 to $J_{33} - 1$. The coordinates of the k-point can then be easily transformed into any basis (such as that of the primitive lattice in reciprocal space) using linear operations. We have shared this approach for iterating over k-points with the Hart group for their work with generalized k-point grids.[30] Values for k_1, k_2, and k_3 can be quickly calculated for any k-point using integer

arithmetic, as discussed below and shown in lines 15 and 16 of Figure S6 of supplementary information.

   Given the enumeration of k-points using equation (12), we identify irreducible k-points in a way similar to that described by Hart et al..[30] We assign a unique index to each k-point in the Brillouin zone or, equivalently, to each k-point in any unit cell of the reciprocal lattice, by

$$index = 1 + k_1 + J_{11}k_2 + J_{11}J_{22}k_3.$$

(13)

The values of the index range from 1 to N_T, and translationally equivalent k-points share the same index. Linear scaling is achieved because the index for any given k-point can be calculated in constant time, as can the sublattice of k-points that have a given index. Then iteration of all k-points in a unit cell in reciprocal space, equivalent to all k-points in the Brillouin zone, is accomplished by looping over values of k_1, k_2, and k_3 in equation (13).
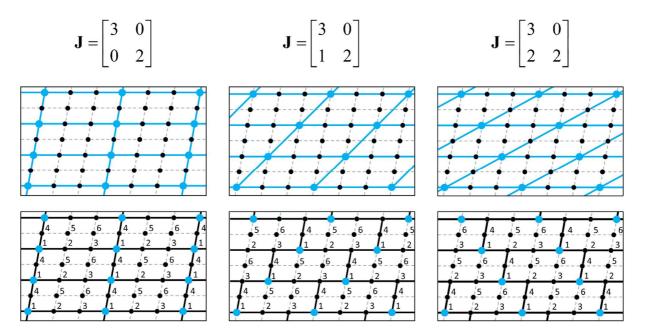

Figure 5. Two-dimensional illustrations of the concepts used for k-point enumeration and index generation. The top row provides the three possible matrices in Hermite normal form for the set

of factors (3,2). The middle row shows the three Bravais superlattices corresponding to these matrices, assuming that the generating vectors for the k-point grid, $\mathbf{d_1}$ and $\mathbf{d_2}$, are aligned with the dashed gray lines. The bottom row shows how space can be tessellated by unit cells that are 3×2 supercells of the generating lattice vectors, with k-point indices marked within each cell.

To count the number of distinct k-points, we iterate over all translationally distinct k-points as described above and apply all symmetry operations to each k-point. If an operation does not transform the k-point to another k-point, the grid is not symmetry-preserving and is rejected (this can sometimes happen if a shift of the Γ point breaks symmetry). If the index of any symmetrically equivalent k-point is less than that of the current k-point, then we have already seen a symmetrically equivalent k-point, so the counter for the number of irreducible k-points is not incremented. If there is no symmetrically equivalent k-point with an index lower than that of the current k-point, then the current k-point is the first we've seen in its orbit, so the counter for the number of irreducible k-points is incremented. A simple variation of this algorithm is used to calculate k-point weights by, for each k-point, determining the orbit of symmetrically equivalent points and then incrementing the weight of the k-point that has the lowest index in that orbit. Figure S6 in supporting information provides the pseudocode of this algorithm. The final, returned arrays contain coordinates and weights for all k-points. The symmetrically non-distinct points, however, have weights of zero. This fact is used to identify the subset of irreducible points.

3. KpLib: A Lightweight, Open-source C++ Library

To facilitate the integration of the generalized Monkhorst-Pack k-point grids in simulation packages, we implemented the presented algorithms in a lightweight library, kpLib. It is written

in C++ to make interfacing easier for as many programming languages as possible. A python module, kpGen, is also provided as a wrapper of the C++ library. The source code of kpLib only contains 1122 lines, and the API uses elementary data structures as argument types, which should be available in most programming languages and facilitate the construction of wrapping functions. We have written a demonstration application, integrated with *spglib* [37], to show how to work with the API. The library is open sourced and a documentation of the API is provided on the homepage of its public repository (https://gitlab.com/muellergroup/kplib). We note that packages that plan to integrate kpLib should ensure that the set of symmetry operations used to generate the k-point grid are used consistently in the rest of the code.

4. Benchmarks

Here we present a series of benchmarks to demonstrate the speed at which our algorithm generates k-point grids and the efficiency of the generated grids, including a comparison to the grids generated using GRkgridgen.[30, 31] All benchmarks were performed on the 102 structures randomly selected from the Inorganic Crystal Structure Database (ICSD) used in our previous work [2, 29]. Version 2019.09.17 for kpLib was used for all benchmarks.

4.1 Grid Generation Speed

We have benchmarked the speed at which kpLib generates both Γ-centered grids and grids with automatically selected shift vectors (called "auto grids" in the following text). To accelerate searches for large grids, we use an approach in which a search for small grids is performed, and then the densities of the small grids are increased in every dimension by a constant scale factor. This use of the scale factor was first introduced in section II.D of our previous work,[2] and it is also adopted in the dynamic generation approach (for a detailed discussion, see section 1 of

supplementary information). We have benchmarked grid generation speed on 102 randomly selected structures using a single core on Intel Xeon E5660 processors with a 2.80 GHz base frequency and a 48 GHZ RAM, with and without the use of the scale factor. Grid sizes are specified by r_{min}, instead of N_{min}, as the former is physically more meaningful,[2, 4] and thus we believe it is the most likely method to be used. A benchmark using N_{min} to compare the speed of the dynamic generation approach and the database look-up approach is given in section 7 of supplementary information.

Average computation time for both Γ-centered grids and auto grids are shown in Figure 6. The speed at which kpLib generates Γ-centered and auto grids is very similar. When r_{min} is 50 angstroms, which is sufficient for converging most calculations within 1 meV / atom,[2] both types of grids are generated in less than 0.2 seconds on average. For large grids, using the scale factor increases generation speed, at a slight cost of grid quality (Figure 8). When r_{min} is 100 angstroms, it takes only about 1 second to find the optimal grids using the scale factor, while the exhaustive search with scale factor switched off finishes in about 4.6 seconds. The smallest value of r_{min} at which the scale factor starts to have an effect is 55 angstroms, but not all 102 structures use the scale factor at 55 angstroms and 69 out of the 102 structures do not use the scale factor even at 100 angstroms.
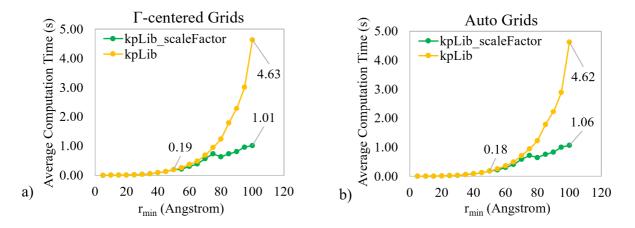
Figure 6. Average computation time of dynamic generation using kpLib with and without scale factors at various values of r_{min} for a) Γ-centered grids and b) auto grids. The computation times at $r_{min} = 100$ angstroms are labeled on the graphs.

The dynamic generation approach used by kpLib is more lightweight than the database approach used by the *K*-Point Grid Generator, which includes a 7.3 MB database containing 428,632 pre-generated grids. However the database lookup method (section 3 of supplementary information) is generally faster (Figure 7). Database searching is much faster than dynamic grid generation for Γ-centered grids over a wide range of densities. The difference between the two approaches is smaller when shifted grids are included, but the database is still two times as fast at the largest r_{min}. This difference in relative performance for shifted grids can be attributed to the fact that dynamic grid generation loops over N_T, and the database search loops over N_i. When searching for shifted grids rather than only Γ-centered grids, the upper bound for the loop over N_T is more rapidly reduced due to the larger number of candidate grids (Figure 1), whereas the upper bound for the loop over N_i is not (Figure S3 of the supplementary information).
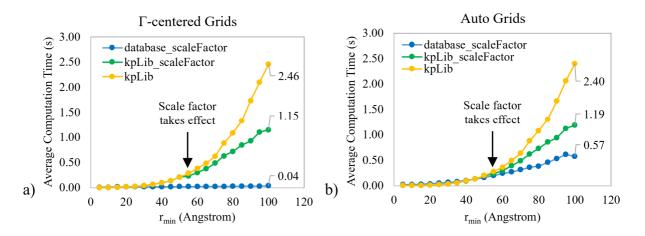
Figure 7. Comparison of computation time between database lookup method used by the *K*-Point Grid Generator and the dynamic generation approach used by kpLib. This benchmark did not include monoclinic and triclinic structures, as both the *K*-Point Grid Generator and kpLib use dynamic grid generation for these.

4.2 Grid Quality Comparison between KpLib and GRkgridgen

We compared our dynamic grid generation method with GRkgridgen, another software package which can generate generalized Monkhorst Pack grids.[31] As the options for grid generation differ between the two packages, we used the following settings to make a fair comparison: both applications were instructed to select the grid with minimal N_i (a natural measure of the efficiency of a grid that meets user-provided constraints), and the required *k*-point density was specified by providing a value for N_{min} (defined as MINTOTALKPOINTS in kpLib and NKPTS in GRkgridgen). In the version we tested, 0.7.5, GRkgridgen doesn't guarantee that the real-space superlattices corresponding to the returned grids satisfy $r_{lattice} \geq r_{min}$, but it does take $r_{lattice}$ into account when generating grids based on N_{min}. As kpLib only accounts for $r_{lattice}$ if r_{min} is provided by the user, to ensure a fair comparison we have constrained the grids generated by

kpLib, to have $r_{lattice}$ which is at least as large as that of the grid generated by GRkgridgen at the same N_{min} and for the same structure. The same 102 structures were used and both Γ-centered grids and auto grids were compared. For kpLib without a scale factor, N_{min} values ranged from 1 to 5623, while for kpLib using scale factor, the range is increased to 15,848 to better demonstrate the effect of scale factor for large grids.
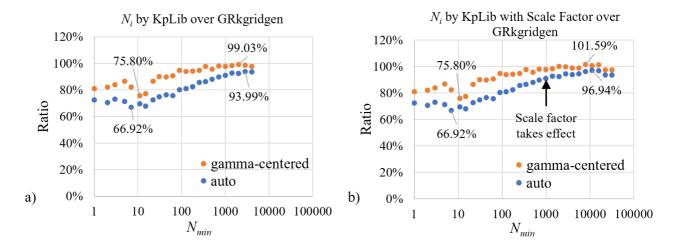

Figure 8. Ratios of average number of symmetrically irreducible k-points from the dynamic search by a) kpLib, b) kpLib with the scale factor, to grids generated using GRkgridgen, for both Γ-centered grids and auto grids. Both the maximal and minimal ratios are labeled for both types of grids. Part b) has a larger range of N_{min} (from 1 to 15,848), to better demonstrate the effect of the scale factor on grid quality.

We use the number of irreducible k-points in the generated grid as a metric of grid efficiency, as the computational cost of most calculations that use k-points scales linearly with the number of irreducible k-points. The scale factor makes little difference in the number of irreducible k-points for grids generated below $N_{min}=5623$ (Figure 8). For auto grids at all values of N_{min},

including those generated using the scale factor, grids from kpLib consistently have fewer

irreducible k-point than the grids from GRkgridgen on average. The same is true for Γ-centered

grids generated without using the scale factor, although for very dense grids when the scale

factor is used GRkgridgen may return grids that are 1-2% more efficient on average. The

difference between kpLib and GRkgridgen is much larger for auto grids than Γ-centered grids,

and it is larger for small N_T than large ones. We note that the gain in performance for relatively

small values of N_T can be particularly beneficial as calculations with such small grids often have

large supercells and are thus computationally demanding. For auto grids, which we expect to be

the most commonly used mode, the expected increase in calculation speed using the grids

generated by kpLib ranges from 3% to 37%.

5. Conclusion

The widespread use of generalized Monkhorst-Pack k-point grids has been limited by the lack

of algorithms and tools for rapidly generating highly efficient grids. By effectively reducing the

problem of generating optimal 3-dimenstional generalized Monkhorst-Pack k-point grids to that

of enumerating over 2-dimensional lattices, along with several other algorithmic innovations, we

have demonstrated that is possible to very rapidly identify optimal generalized Monkhorst-Pack

k-point grids for a given material, given user constraints on the spacing of the real-space grid

points and/or the minimum total required k-points. For commonly-used grid densities, the grids

generated by the algorithms presented in this paper are on average significantly more efficient

than those generated using previously developed algorithms. Given the demonstrated benefits of

using generalized Monkhorst-Pack k-point grids[2-4, 24], we conservatively estimate that

widespread adoption of these algorithms could save computational materials researchers more

than a hundred million CPU hours, worth millions of US dollars, each year. To facilitate this widespread use, we have implemented our algorithms for grid generation in kpLib, a lightweight open source library with only 1122 lines of code for integration with third-party software algorithms, and we have developed a standalone open-source tool, the *K*-Point Grid Generator, for rapidly generating generalized Monkhorst-Pack grids.

Corresponding Author

Tim Mueller – Department of Materials Science and Engineering, Johns Hopkins University, Baltimore, MD 21218, United States. E-mail: tmueller@jhu.edu

Data Availability

The source code of kpLib and *K*-Point Grid Generator are freely available online at https://gitlab.com/muellergroup/kplib and https://gitlab.com/muellergroup/k-pointGridGenerator. The raw data required to reproduce these findings are available to download from https://gitlab.com/muellergroup/kplib/-/blob/master/doc/paper_data/Raw_Data.xlsx.

REFERENCES

1.   Monkhorst, H. J.; Pack, J. D., Special Points for Brillouin-Zone Integrations. *Physical Review B* **1976,** *13* (12), 5188-5192.

2.   Wisesa, P.; McGill, K. A.; Mueller, T., Efficient generation of generalized Monkhorst-Pack grids through the use of informatics. *Physical Review B* **2016,** *93* (15).

3.   Morgan, W. S.; Jorgensen, J. J.; Hess, B. C.; Hart, G. L. W., Efficiency of Generalized Regular k-point grids. *Comp Mater Sci* **2018,** *153*, 424-430.

4.   Choudhary, K.; Tavazza, F., Convergence and machine learning predictions of Monkhorst-Pack k-points and plane-wave cut-off in high-throughput DFT calculations. *Comp Mater Sci* **2019,** *161*, 300-308.

5.   Amazon Web Services Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/ (accessed June 19 2020).

6.   Froyen, S., Brillouin-zone integration by Fourier quadrature: Special points for superlattice and supercell calculations. *Phys Rev B* **1989,** *39* (5), 3168-3172.

7.   Moreno, J.; Soler, J. M., Optimal Meshes for Integrals in Real-Space and Reciprocal-Space Unit Cells. *Physical Review B* **1992,** *45* (24), 13891-13898.

8.   Mundet, B.; Hartman, S. T. T.; Guzman, R.; Idrobo, J. C. C.; Obradors, X.; Puig, T.; Mishra, R.; Gazquez, J., Local strain-driven migration of oxygen vacancies to apical sites in YBa2Cu3O7-x. *Nanoscale* **2020,** *12* (10), 5922-5931.

9.   Chowdhury, T.; Kim, J.; Sadler, E. C.; Li, C.; Lee, S. W.; Jo, K.; Xu, W.; Gracias, D. H.; Drichko, N. V.; Jariwala, D.; Brintlinger, T. H.; Mueller, T.; Park, H.-G.; Kempa, T. J.,

Substrate-directed synthesis of MoS2 nanocrystals with tunable dimensionality and optical properties. *Nature Nanotechnology* **2020,** *15* (1), 29-34.

10.   Wisesa, P.; Li, C.; Mueller, T., Materials with the CrVO4 structure type as candidate superprotonic conductors. **2019**.

11.   Williams, L.; Kioupakis, E., BAlGaN alloys nearly lattice-matched to AlN for efficient UV LEDs. *Appl Phys Lett* **2019,** *115* (23), 231103.

12.   Wang, Y.; Cao, L.; Libretto, N. J.; Li, X.; Li, C.; Wan, Y.; He, C.; Lee, J.; Gregg, J.; Zong, H.; Su, D.; Miller, J. T.; Mueller, T.; Wang, C., Ensemble Effect in Bimetallic Electrocatalysts for CO2 Reduction. *J Am Chem Soc* **2019,** *141* (42), 16635-16642.

13.   Rosenbrock, C. W.; Gubaev, K.; Shapeev, A. V.; Pártay, L. B.; Bernstein, N.; Hart, G. L., Machine-learned Interatomic Potentials for Alloys and Alloy Phase Diagrams. *arXiv preprint arXiv:1906.07816* **2019**.

14.   Nyshadham, C.; Rupp, M.; Bekker, B.; Shapeev, A. V.; Mueller, T.; Rosenbrock, C. W.; Csányi, G.; Wingate, D. W.; Hart, G. L., Machine-learned multi-system surrogate models for materials prediction. *Npj Comput Mater* **2019,** *5* (1), 51.

15.   Li, C.; Gao, H.; Wan, W.; Mueller, T., Mechanisms for hydrogen evolution on transition metal phosphide catalysts and a comparison to Pt(111). *Phys Chem Chem Phys* **2019,** *21* (44), 24489-24498.

16.   Kratzer, P.; Neugebauer, J., The basics of electronic structure theory for periodic systems. *Frontiers in chemistry* **2019,** *7*.

17.   Hernandez, A.; Balasubramanian, A.; Yuan, F.; Mason, S. A. M.; Mueller, T., Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *Npj Comput Mater* **2019,** *5* (1), 112.

18.   Greenman, K.; Williams, L.; Kioupakis, E., Lattice-constant and band-gap tuning in wurtzite and zincblende BInGaN alloys. *Journal of Applied Physics* **2019,** *126* (5), 055702.

19.   Cao, L.; Zhao, Z.; Liu, Z.; Gao, W.; Dai, S.; Gha, J.; Xue, W.; Sun, H.; Duan, X.; Pan, X.;   Mueller, T.; Huang, Y., Differential Surface Elemental Distribution Leads to Significantly Enhanced Stability of PtNi-Based ORR Catalysts. *Matter* **2019,** *1* (6), 1567-1580.

20.   Cao, L.; Niu, L.; Mueller, T., Computationally generated maps of surface structures and catalytic activities for alloy phase diagrams. *Proceedings of the National Academy of Sciences of the United States of America* **2019,** *116* (44), 22044-22051.

21.   Liu, Y.; Zhang, H.; Behara, P. K.; Wang, X.; Zhu, D.; Ding, S.; Ganesh, S. P.; Dupuis, M.;   Wu, G.; Swihart, M. T., Synthesis and Anisotropic Electrocatalytic Activity of Covellite Nanoplatelets with Fixed Thickness and Tunable Diameter. *ACS applied materials & interfaces* **2018,** *10* (49), 42417-42426.

22.   Li, C. Y.; Raciti, D.; Pu, T. C.; Cao, L.; He, C.; Wang, C.; Mueller, T., Improved Prediction of Nanoalloy Structures by the Explicit Inclusion of Adsorbates in Cluster Expansions. *J Phys Chem C* **2018,** *122* (31), 18040-18047.

23.   Ding, Y.; Wang, Y., Tunable Electronic Structures of Hydrogenated Zigzag and Armchair Dumbbell Silicene Nanosheets: A Computational Study. *The Journal of Physical Chemistry C* **2018,** *122* (40), 23208-23216.

24.   Wolloch, M.; Suess, D.; Mohn, P., Influence of antisite defects and stacking faults on the magnetocrystalline anisotropy of FePt. *Physical Review B* **2017,** *96* (10), 104408.

25.   Williams, L.; Kioupakis, E., BInGaN alloys nearly lattice-matched to GaN for high-power high-efficiency visible LEDs. *Appl Phys Lett* **2017,** *111* (21), 211107.

26.   Raciti, D.; Cao, L.; Liv, K. J. T.; Rottmann, P. F.; Tang, X.; Li, C. Y.; Hicks, Z.; Bowen, K. H.; Hemker, K. J.; Mueller, T.; Wang, C., Low-Overpotential Electroreduction of Carbon Monoxide Using Copper Nanowires. *Acs Catalysis* **2017,** *7* (7), 4467-4472.

27.   Cao, L.; Raciti, D.; Li, C. Y.; Livi, K. J. T.; Rottmann, P. F.; Hemker, K. J.; Mueller, T.; Wang, C., Mechanistic Insights for Low-Overpotential Electroreduction of CO2 to CO on Copper Nanowires. *Acs Catalysis* **2017,** *7* (12), 8578-8587.

28.   Vienna   Ab   initio   Simulation   Package   KPOINTS   in   VASP   Wiki. https://www.vasp.at/wiki/index.php/KPOINTS.

29.   Inorganic Crystal Structure Database. Fiz Karlsruhe: 2016.

30.   Hart, G. L. W.; Jorgensen, J. J.; Morgan, W. S.; Forcade, R. W., A robust algorithm for k-point grid generation and symmetry reduction. *Journal of Physics Communications* **2019,** *3* (6), 065009.

31.   Morgan, W. S.; Christensen, J. E.; Hamilton, P. K.; Jorgensen, J. J.; Campbell, B. J.; Hart, G. L. W.; Forcade, R. W., Generalized regular k-point grid generation on the fly. *Comp Mater Sci* **2020,** *173*.

32.   Ashcroft, N. W.; Mermin, N. D., *Solid State Physics*. Brooks/Cole: Belmont, USA, 1976; p 132-143.

33.   Tinkham, M., *Group Theory and Quantum Mechanics*. McGraw-Hill Book Company: New York, 1964; p 279-281.

34.   Giacovazzo, C.; Monaco, H. L.; Artioli, G.; Viterbo, D.; Milanesio, M.; Ferraris, G.; Gilli, G.; Gilli, P.; Zanotti, G.; Catti, M., *Fundamentals of Crystallography*. third ed.; Oxford University Press: Oxford, 2002; p 842.

35.   Mueller, T. Computational studies of hydrogen storage materials and the development of related methods. Massachusetts Institute of Technology, Boston, Massachusetts, 2007.

36.   Hart, G. L. W.; Forcade, R. W., Algorithm for generating derivative structures. *Physical Review B* **2008,** *77* (22).

37.   Togo, A.; Tanaka, I. Spglib: a software library for crystal symmetry search *arXiv e-prints* [Online], 2018. https://ui.adsabs.harvard.edu/abs/2018arXiv180801590T (accessed August 01, 2018).

.