RSC Advances

PAPER

View Article Online
View Journal | View Issue

Cite this: *RSC Adv.*, 2019, **9**, 33310

New scaling relations to compute atom-in-material polarizabilities and dispersion coefficients: part 2. Linear-scaling computational algorithms and parallelization†

Thomas A. Manz * and Taoyi Chen

We present two algorithms to compute system-specific polarizabilities and dispersion coefficients such that required memory and computational time scale linearly with increasing number of atoms in the unit cell for large systems. The first algorithm computes the atom-in-material (AIM) static polarizability tensors, force-field polarizabilities, and C_6, C_8, C_9, C_{10} dispersion coefficients using the MCLF method. The second algorithm computes the AIM polarizability tensors and C_6 coefficients using the TS-SCS method. Linear-scaling computational cost is achieved using a dipole interaction cutoff length function combined with iterative methods that avoid large dense matrix multiplies and large matrix inversions. For MCLF, Richardson extrapolation of the screening increments is used. For TS-SCS, a failproof conjugate residual (FCR) algorithm is introduced that solves any linear equation system having Hermitian coefficients matrix. These algorithms have mathematically provable stable convergence that resists round-off errors. We parallelized these methods to provide rapid computation on multi-core computers. Excellent parallelization efficiencies were obtained, and adding parallel processors does not significantly increase memory requirements. This enables system-specific polarizabilities and dispersion coefficients to be readily computed for materials containing millions of atoms in the unit cell. The largest example studied herein is an ice crystal containing >2 million atoms in the unit cell. For this material, the FCR algorithm solved a linear equation system containing >6 million rows, 7.57 billion interacting atom pairs, 45.4 billion stored non-negligible matrix components used in each large matrix-vector multiplication, and ~19 million unknowns per frequency point (>300 million total unknowns).

Received 14th March 2019
Accepted 23rd September 2019

DOI: 10.1039/c9ra01983a

rsc.li/rsc-advances

1. Introduction

In the first part of this series, we introduced the MCLF method (acronym from authors' last initials) to compute atom-in-material (AIM) polarizabilities and dispersion coefficients.[1] We compared chemical performance of MCLF to the Tkatchenko–Scheffler method with self-consistent screening (TS-SCS).[1] Computed polarizabilities and/or dispersion coefficients were compared to experimental and/or high-level computational benchmark data for isolated atoms, diatomic molecules, small polyatomic molecules, fullerenes, polyacenes, and solids.[1] For HIV reverse transcriptase complexed with an inhibitor, computed MCLF and TS-SCS AIM polarizabilities and dispersion coefficients were compared to the OPLS biomolecular force-field.[1]

In this article, we introduce computationally efficient algorithms that extend the MCLF and TS-SCS methods to materials having large numbers of atoms in the unit cell. For sufficiently large systems, both the required memory and computational time scale linearly with increasing number of atoms in the unit cell. Our methods can easily be applied to materials containing millions of atoms in the unit cell. This is orders of magnitude larger than unit cells for materials previously studied with MCLF and TS-SCS methods. For small unit cells, our methods are still faster and require less memory than direct matrix inversion with negligible difference in computational precision.

The TS-SCS method was introduced in 2012.[2] TS-SCS can be combined with multibody dispersion (MBD), a damping function, and density functional theory (DFT) to give a DFT + dispersion method.[2] Ambrosetti *et al.* introduced range-

Department of Chemical & Materials Engineering, New Mexico State University, Las Cruces, New Mexico, 88003-8001, USA. E-mail: tmanz@nmsu.edu

† Electronic supplementary information (ESI) available: A pdf file containing: a proof that $\alpha^{\text{force-field}}$, $\alpha^{\text{non-dir}}(u)$, $\alpha^{\text{low\_freq}}$, and $\alpha^{\text{screened}}(u)$ are ≥ 0 (S1); derivations and proofs related to the failproof conjugate residual (FCR) algorithm (S2); a proof the wp lookup table method yields error proportional to the increment squared (S3); linear-scaling algorithm for setting up the lists of interacting atom pairs (S4). A zip format archive containing: Romberg integration and Richardson extrapolation coefficients; a Fortran file containing a function that computes C_6^{total} using the wp lookup table method; Fortran modules containing sample code to perform atom image pair array initialization. See DOI: 10.1039/c9ra01983a

separated dipole interaction tensors to avoid (or minimize) double-counting dispersion interactions in the combined MBD@rsSCS framework.[3] The TS-SCS method requires atom-in-material $\langle r^3 \rangle$ radial moments as inputs.[2,4] Several TS-SCS variants used different partitioning methods to compute the $\langle r^3 \rangle$ moments: TS-SCS/Hirshfeld,[2,4,5] TS-SCS/iterative Hirshfeld,[6,7] and TS-SCS/DDEC6.[1] Normally, the TS method assumes a constant polarizability to $\langle r^3 \rangle$ moment ratio for a specific chemical element irrespective of its charge states in a material.[2,4] Recently, Gould *et al.* introduced a Fractionally Ionic (FI) method that adjusts the polarizability to $\langle r^3 \rangle$ moment ratio using partial atomic charges.[8] This requires a library of reference ion polarizabilities and C_6 coefficients for free atoms in different charge states.[8,9] However, these reference polarizabilities and C_6 coefficients are hard to compute for anions such as O^{2-} that contain unbound electrons[10] in their free state.

In the TS-SCS method, the $C_{6,A}$ dispersion coefficient is computed using the Casimir–Polder integral (eqn (8)), which requires AIM polarizabilities as a function of imaginary frequency (aka imfreq). These AIM polarizabilities are computed by solving the following linear equation system

$$\sum_{t=1}^{3} \sum_{B=1}^{N\text{atoms}} \left(\frac{\delta_{st}}{\alpha_A^{\text{unscreened}}(u)} - \tau_{st}{}^{AB}(u) \right) \overrightarrow{\overrightarrow{\alpha}}_B^{\text{SCS}}(u) = \delta_{sk} \qquad (1)$$

at each imfreq point u, where $\tau_{st}{}^{AB}(u)$ is the dipole–dipole interaction tensor, s and t are spatial indices (*e.g.*, x, y, or z), δ_{st} is the Kronecker delta, $\alpha_A^{\text{unscreened}}(u)$ is the isotropic AIM unscreened polarizability as a function of u, and $\overrightarrow{\overrightarrow{\alpha}}_B^{\text{SCS}}(u)$ is the self-consistent screened AIM polarizability tensor as a function of u.[2] A dipole interaction cutoff length function turns off dipole interactions between atoms separated by a distance larger than this cutoff length (*e.g.*, 50 bohr). This makes the coefficients matrix sparse when the material's unit cell is much larger than this cutoff length. Because the dipole interaction tensor is symmetric in both atomic indices and spatial indices, developing a computationally efficient TS-SCS algorithm is functionally equivalent to developing a computationally efficient algorithm to solve a set of real-valued symmetric sparse linear equations.

At its heart, therefore, this is a fundamental linear algebra problem. Specifically, to solve a linear equation system $\mathbf{M}y = \boldsymbol{W}$ having a sparse real-valued symmetric coefficient matrix $\mathbf{M}$. A naïve approach would invert the matrix $\mathbf{M}$ to compute $y = (\text{inverse}(\mathbf{M}))\boldsymbol{W}$. However, this is computationally infeasible for matrices containing a large number of rows (Nrows). First, direct matrix inversion algorithms (*e.g.*, Gaussian elimination with partial pivoting (GEPP) and the more advanced Strassen algorithm) have computational costs scaling between $(N\text{rows})^2$ and $(N\text{rows})^3$.[11] Second, inverse($\mathbf{M}$) may be dense (or at least considerably less sparse than $\mathbf{M}$) leading to high matrix multiplication costs. The cost of multiplying two dense square matrices is between $(N\text{rows})^2$ and $(N\text{rows})^3$,[11] and multiplying a dense square matrix times a dense vector would cost $(N\text{rows})^2$ if using conventional multiplication. Therefore, the primary goal is to solve this linear equation system without any large dense matrix multiplies or large matrix inversions. Here, a new conjugate residual algorithm is introduced to achieve this with a computational cost proportional to Nrows. This new conjugate residual algorithm resists round-off errors.

As described in the first article of this series,[1] the MCLF method includes numerous important innovations. It uses a conduction limit upper bound to ensure assigned AIM polarizabilities do not exceed those of a perfect conductor. It uses m-scaling to smoothly transition between the scaling behaviors of surface and buried atoms. It uses new scaling relationships to describe changes in the polarizability-to-$\langle r^3 \rangle$ moment ratio as a function of the atomic charge state without requiring quantum mechanical (QM) computed reference polarizabilities and C_6 values for charged atoms. This is an important advantage, because some isolated charged atoms are unstable (*e.g.*, O^{2-} as discussed above). To compute several different kinds of AIM polarizabilities, MCLF separates directional from non-directional dipole interaction tensor contributions. This includes: (a) non-directionally screened polarizabilities to be used as force-field input parameters, (b) fluctuating polarizabilities to compute the C_6 dispersion coefficients, and (c) static polarizabilities containing long-range dipole alignment due to a constant externally applied electric field. A new polarizability partition and iterative polarizability screening ensure assigned AIM polarizabilities are non-negative. A proof that MCLF $\alpha^{\text{force-field}}$, $\alpha^{\text{non-dir}}(u)$, $\alpha^{\text{low\_freq}}$, and $\alpha^{\text{screened}}(u)$ are ≥ 0 is provided in the ESI.† MCLF uses a multibody screening function to capture the fluctuating dipole alignment at short distances and disorder at long distances. This leads to more accurate C_6 coefficients. Quantum Drude oscillator (QDO) parameters yield higher-order AIM dispersion coefficients (*e.g.*, C_8, C_9, C_{10}) and associated mixing rules.

The MCLF method includes polarizability screening using a much different approach than TS-SCS.[1] While TS-SCS solves a linear equation system (eqn (1)), MCLF uses an incremental polarizability screening.[1] Accordingly, our linear-scaling MCLF algorithm is vastly different in mathematical approach than our linear-scaling TS-SCS algorithm. As explained in Section 2 below, we use Richardson extrapolation of the screening increments to achieve high computational efficiency and precision for MCLF. Both our linear-scaling MCLF and linear-scaling TS-SCS algorithms avoid large dense matrix multiplications and large matrix inversions.

These computational methods should find widespread use beyond the specific applications studied herein. This new conjugate residual algorithm solves any linear equation system having Hermitian coefficients matrix. (Conditioning is required for ill-conditioned matrices.) Because it has mathematically provable stable convergence that resists round-off errors, this conjugate residual algorithm should find widespread applications to solve a plethora of scientific computing problems involving large sparse linear equation systems. Potential applications for such a method are vast.

The remainder of this article is organized as follows. Section 2 presents the linear-scaling MCLF algorithm. The failsafe

conjugate residual and linear-scaling TS-SCS algorithms are introduced in Section 3. Section 4 contains the performance results: required computational times, required memory, and parallelization efficiencies. All computational timing and memory results reported in this paper are for the MCLF or TS-SCS analysis; they do not include the prior quantum chemistry calculation (if any) and prior atomic population analysis used to provide the input data. Section 5 concludes. The ESI† contains mathematical derivations and proofs pertaining to these methods.

Finally, we comment on the choice of atomic population analysis method (APAM) used to provide input data to the MCLF and TS-SCS methods. The memory requirements for MCLF and TS-SCS analysis do not depend on which APAM provides the input data. The required computational time for MCLF analysis does not depend on which APAM provides the input data. The required computational time for TS-SCS analysis using the FCR algorithm only depends on the number of FCR iterations, which could potentially be weakly affected by the choice of APAM used to provide the input data. In this article, we used DDEC6 (ref. 12 and 13) atomic population analysis to generate the input data for MCLF and TS-SCS analysis. (Other stockholder partitioning methods could potentially be used.) Currently, DDEC6 is the most recent generation of the Density Derived Electrostatic and Chemical (DDEC) methods that are optimized to quantify important electrostatic, magnetic, and chemical properties across diverse materials.[12–18] All DDEC6 calculations were performed using the Chargemol program.[12]

The overall sequence is: QM calculation → APAM → MCLF or TS-SCS analysis. DDEC6 and many other stockholder partitioning methods can be made strictly linear-scaling by using a cutoff radius around each atom in a material.[12] The electron and spin densities assigned to an atom in the material are zero outside this cutoff radius. This yields linear-scaling computational time and memory, because the number of integration points per atom does not increase as *N*atoms increases.[12] *N*atoms is the number of atoms in the unit cell.

QM computation of the material's electron and spin densities is the rate-limiting step in this overall sequence. Significant progress has been made developing linear-scaling DFT codes that have applications to studying unit cells containing ~10 000 atoms.[62–69] Orbital-free DFT calculation (which is presently most suitable for treating metallic conductors[66]) was performed for a million atom system.[66] Further improvements are needed to enable DFT calculations on millions and billions of atoms. Even without DFT calculations of such large systems, the MCLF (or TS-SCS) method could still find applications to materials containing millions of atoms. For example, by using DFT calculations and DDEC6 analysis on smaller clusters to parameterize information for atom types, followed by MCLF (or TS-SCS) analysis for the full material. This strategy would make sense, because DDEC6 analysis is more localized (*i.e.*, cutoff radius = 5 Å) compared to MCLF (or TS-SCS) analysis (*e.g.*, dipole interaction cutoff length = 50 bohr).[1,13]

2. Linear-scaling MCLF algorithm

2.1 How strict linear-scaling is achieved

Strict linear-scaling means that each and every part of the MCLF program scales no worse than linear in computational time and memory as *N*atoms increases when *N*atoms is sufficiently large. First, no allocatable arrays having multiple dimensions of size proportional to *N*atoms (*e.g.*, *My_array*(*N*atoms, *N*atoms)) are allocated for *N*atoms greater than a threshold. Second, no nested DO loops having two indices of ranges proportional to *N*atoms (*e.g.*, requiring *Order*(*N*atoms^2) iterations) are executed for *N*atoms greater than a threshold. This is achieved by making sure the program is written such that all sets of nested DO loops and allocatable arrays (that are operational for *N*atoms greater than a threshold) have at most one index whose range is proportional to *N*atoms.

This is physically enabled by using a dipole interaction cutoff length such that any two atoms farther apart have no direct interactions. As explained in Section 2.2 below, the program constructs lists of interacting atom pairs and uses these in subsequent calculations. Because each atom in the material only directly interacts with a limited number of other atoms, for sufficiently large *N*atoms the total number of symmetry unique interacting atom pairs is proportional to *N*atoms. This is true even if the material is periodic and extends forever. This is conceptually equivalent to a sparse matrix algebra.

Other mathematical innovations are also employed. First, a special lookup table method (see Section 2.4) is used to compute the total dispersion coefficient

$$C_6^{\text{total}} = \sum_A \sum_{B>A} (2C_{6,AB}) + \sum_A C_{6,A} \quad (2)$$

without having to directly itemize this summation over all pairs of atoms in the unit cell. Second, directly inverting a large matrix (even if it is sparse) can potentially require more operations than *Order*(*N*rows). As explained in Section 2.5, a linear-scaling algorithm was developed that avoids large matrix inversion. This algorithm is linear-scaling, because it requires a fixed number of large matrix-vector multiplications in which the matrix is sparse. This matrix is the dipole interaction tensor, whose non-zero components correspond to the interacting atom pairs. Hence, each large matrix-vector multiplication corresponds to looping over the lists of symmetry unique interacting atom pairs.

As explained in Section 2.6, all of these linear-scaling innovations were parallelized to enable fast multi-core computing. Moreover, all of these innovations were programmed in a general way that handles 0, 1, 2, or 3 periodic boundary conditions (PBC). For input files of materials having periodic unit cells, these MCLF and TS-SCS programs include the option to use or ignore the input file PBC. For example, a molecule can be simulated in a planewave code (such as VASP) by placing it near the center of a large cube with PBC. In this case, to yield AIM polarizabilities and dispersion coefficients for the isolated molecule, the MCLF or TS-SCS program should be set to ignore these PBC; this turns off all interactions between periodic images.

2.2 Lists of interacting atom pairs

We use a capital letter (*e.g.*, *A*, *B*) to represent an atom in the reference unit cell. A small letter represents a translated atomic image that is not necessarily located in the reference unit cell. For example, $b = (B, L_1, L_2, L_3)$ represents the image of atom *B* that is translated L_1 times the first lattice vector $\vec{\boldsymbol{v}}_1$, L_2 times the second lattice vector $\vec{\boldsymbol{v}}_2$, and L_3 times the third lattice vector $\vec{\boldsymbol{v}}_3$.

$$\vec{\boldsymbol{R}}_b = \vec{\boldsymbol{R}}_B + L_1\vec{\boldsymbol{v}}_1 + L_2\vec{\boldsymbol{v}}_2 + L_3\vec{\boldsymbol{v}}_3 \tag{3}$$

is the nuclear position of this image, where $\vec{\boldsymbol{R}}_B$ is the nuclear position of the parent atom in the reference unit cell.

During MCLF analysis, two separate lists of interacting atom pairs are prepared after the unscreened calculation and before the screened calculation. For convenience, we refer to these as the 'small' and 'large' lists, where 'small' and 'large' refer to the interaction cutoff distance. Without loss of generality, the first atom in each pair can be considered to reside within the reference unit cell. The second atom is located somewhere within the dipole interaction cutoff length of the first atom. Only translation symmetry unique atom pairs need to be included in the loops over atom pairs. As explained in the earlier bond order article, an atom image pair is translation symmetry unique if and only if at least one of the following criteria is satisfied: (i) the index numbers of the atoms are different (*e.g.*, atom 210 and atom 1056), (ii) $L_1 > 0$, (iii) $L_1 = 0$ and $L_2 > 0$, or (iv) $L_1 = L_2 = 0$ and $L_3 > 0$.[18]

The small lists all atom pairs having 'overlapping' Gaussian dipole model densities as defined by the cutoff criterion

$$\upsilon_{Ab}^{\text{unscreened}} = \frac{d_{Ab}}{\sigma_{AB}^{\text{unscreened}}(u = N\text{imfreqs})} \leq \upsilon_{\text{cutoff}} \tag{4}$$

where

$$d_{Ab} = \|\vec{\boldsymbol{R}}_A - \vec{\boldsymbol{R}}_b\| \tag{5}$$

is the distance from atom *A* to image *b*, and $\sigma_{AB}^{\text{unscreened}}(u = N\text{imfreqs})$ is the static attenuation length. This criterion ensures $\text{erfc}(\upsilon_{Ab}) \approx \exp(-(\upsilon_{Ab}{}^2)) \approx 0$ for any atom pair not included in the list. The value $\upsilon_{\text{cutoff}} = 5^{4/3}$ was chosen such that even if the (partially) screened polarizability is larger than the unscreened polarizability by up to a factor of five, then $\text{erfc}(\upsilon_{Ab}^{\text{screened}}) \leq \text{erfc}(5) = 1.5 \times 10^{-12}$ and $\exp(-(\upsilon_{Ab}^{\text{screened}})^2) \leq \exp(-25) = 1.4 \times 10^{-11}$. The following information was saved in the small list array for each included atom pair: the index numbers of the first and second atoms, the translation integers for the second atom, the atomic number of each atom, d_{Ab}, the value of the smooth cutoff function, the value of the multibody screening function times the smooth cutoff function, and the following six tensor components: $\eta_{1,1} = 3(\Delta x)^2/d_{Ab}{}^2 - 1$, $\eta_{1,2} = \eta_{2,1} = 3\Delta x\Delta y/d_{Ab}{}^2$, $\eta_{1,3} = \eta_{3,1} = 3\Delta x\Delta z/d_{Ab}{}^2$, $\eta_{22} = 3(\Delta y)^2/d_{Ab}{}^2 - 1$, $\eta_{2,3} = \eta_{3,2} = 3\Delta y\Delta z/d_{Ab}{}^2$, $\eta_{3,3} = 3(\Delta z)^2/d_{Ab}{}^2 - 1$. Here, Δx, Δy, and Δz are the Cartesian components of $\vec{\boldsymbol{R}}_A - \vec{\boldsymbol{R}}_b$.

The large lists all atom pairs (*A*, *B*) in the reference unit cell for which atom *A* (first atom) is located a distance less than dipole interaction cutoff length to at least one image of atom *B* (second atom). The self-pair (*A*, *A*) is included if and only if one of the non-trivially translated images of atom *A* is located a distance less than dipole interaction cutoff length to atom *A* in the reference unit cell. The number of atom pairs in the large list is always $\leq N\text{atoms}(N\text{atoms} + 1)/2$. For example, a NaCl crystal containing two atoms in the reference unit cell would contain three pairs in the large list: Na–Na, Na–Cl, and Cl–Cl. For each pair (*A*, *B*) in the large list, a loop is performed over all atom *b* images located within dipole interaction cutoff length of atom *A*, and the following sums are accumulated and stored:

$$\vec{\boldsymbol{g}}^{AB} = \sum_L f_{\text{cutoff}}(d_{Ab})\vec{\boldsymbol{\eta}}^{Ab} \tag{6}$$

$$\vec{\boldsymbol{h}}^{AB} = \sum_L f_{\text{cutoff}}(d_{Ab}) f_{\text{MBS}}(d_{Ab})\vec{\boldsymbol{\eta}}^{Ab} \tag{7}$$

Here, f_{cutoff} and f_{MBS} are the smooth cutoff function and multibody screening function, respectively, described in the prior article.[1] Since $\vec{\boldsymbol{g}}$ and $\vec{\boldsymbol{h}}$ are symmetric with respect to exchange of the spatial indices, only six components need to be computed and stored for each.

Our algorithm for constructing these small and large lists has time and memory requirements scaling linearly with increasing *N*atoms. Section S4 of the ESI† explains this algorithm's details. Fig. S1 of the ESI† is a flow diagram summarizing this process. The ESI† also contains Fortran modules that can be examined for coding details. Data is grouped to enable fast computation by avoiding all array searches. For example, information is ordered such that arrays do not have to be searched to identify which atoms belong in each spatial region. Also, each array allocation is performed once, rather than continuously appending arrays (which would be extremely slow). This is accomplished by first performing a 'dry run' code block that executes a sequence to count up the required array size, followed by array allocation, followed by a code block that writes data to the allocated array. The four key steps to construct these lists are:

1. *Define basis vectors and unit cell parallelepiped:* A parallelepiped of non-zero volume is constructed to enclose the system's unit cell. Three basis vectors correspond to this parallelepiped's non-collinear edges. For a periodic direction, the basis vector is the corresponding periodic lattice vector. For a non-periodic direction, the basis vector is chosen to be of a non-zero length that fully encloses all the nuclear positions. Periodic basis vectors can be non-perpendicular to each other (*e.g.*, triclinic unit cells), but each non-periodic basis vector is chosen to be perpendicular to the other basis vectors.

2. *Divide the unit cell parallelepiped into spatial regions:* This unit cell parallelepiped is divided into a whole number of spatial regions along each basis vector. A periodic direction produces an infinite number of periodic images of each region, while a non-periodic direction has only the reference image. Atoms in the reference unit cell are classified by region, and a sorted list is prepared such that atoms of the same region are adjacent in this sorted list. Because each spatial region is defined such that its volume is less than that of a sphere of dipole interaction cutoff length radius, the number of atoms in

each region is always below a threshold. The code ignores regions that do not contain any atoms. Because empty regions are skipped, having a few atoms in the center of an enormous unit cell would execute quickly.

3. *Construct arrays listing interacting region pair images:* Two spatial region images interact if the minimum distance between inter-region points is ≤the dipole interaction cutoff length. Because the spatial regions and their images are coordinate system indexed, a list of interacting region pair images is constructed without having to construct a double summation over all region pairs. Thus, even for an extremely large unit cell (*e.g.*, containing billions of atoms) divided into many regions (*e.g.*, millions), the list of interacting region pair images is constructed in time and memory scaling linearly with increasing unit cell size. Different regions can interact with different periodic images. For example, in an extremely large unit cell, a region near the center would interact only with nearby regions in the reference unit cell, while a region not too far from the left edge would interact with some regions in the reference unit cell and some other region images in the left-translated unit cell. Thus, a first array is constructed listing pairs of regions having any interacting images, and a second array is prepared that lists which specific images of each particular region pair interact. Region pairs that do not interact are not included in these two arrays.

4. *Construct two lists of interacting atom pairs:* Because interacting atom pairs must be contained in interacting region pair images, the code identifies the interacting atom pairs by executing an outer loop over the interacting region pair images and inner loops over the atoms in these regions (along with tests for inclusion criteria). Using the list of atoms sorted by region makes this process cache access friendly. For each such atom pair, tests are performed to determine if it meets the small and large list inclusion criteria. If so, its information is added to the small and/or large lists. Because the number of interacting region pair images scales linearly with large *N*atoms and the number of atoms in each region is below a threshold, these small and large lists are constructed in linear-scaling computational time and memory for large *N*atoms.

Having separate small and large lists provides the following computational efficiencies during the subsequent MCLF polarizability screening. First, non-directional screening only needs to be performed over the 'overlapping' atoms contained in the small list. Second, the computationally expensive erfc function only needs to be evaluated for atom pairs in the small list. This provides computational savings during directional screening that first loops over all pairs in the small list and then over all pairs in the large list. Third, the large list contains precomputed sums over all periodic images of interacting atom pairs. For non-overlapping atoms, this avoids re-computing any sums over periodic images at each screening increment and each frequency point.

These two lists of interacting atom pairs are used as follows. For each screening increment of each frequency point, non-directional screening loops over all atom pairs in the small list to compile the necessary dipole–dipole interaction terms. For each screening increment of each frequency point, directional screening first loops over all atom pairs in the small list and then over all atom pairs in the large list to compile the associated dipole–dipole interaction terms. More details are provided in Sections 2.5 and 2.6 below.

Table 1 studies the effect of dipole interaction cutoff length on computed precision. Graphene was chosen as a test system, because it has strong long-range dipole–dipole coupling. As shown in Table 1, α^{static} was the most sensitive to the dipole interaction cutoff length, and $\alpha^{\text{force-field}}$ was the least sensitive. For 2-dimensional sheets such as graphene, doubling the dipole interaction cutoff length increases the computational cost by approximately four-fold. For dense materials with large unit cells, doubling the dipole interaction cutoff length increases the computational cost by approximately eight-fold. We selected dipole interaction cutoff length = 50 bohr as a good compromise between computational cost and precision. This value was used for all other results in this article.

The graphene primitive unit cell was optimized in VASP[19–21] using the PBE[22] functional, a 400 eV planewave cutoff, and the projector augmented wave (PAW[23,24]) method. The *k*-point mesh and grid spacing followed previous recommendations.[12] This yielded a nearest neighbor C–C distance of 1.42 Å. Our MCLF and TS-SCS calculations of graphene were performed using 2-D not 3-D periodic boundary conditions (*i.e.*, the spacing between graphene layers was set to infinite at the start of MCLF or TS-SCS calculation).

2.3 Integration over imaginary frequencies

The Casimir–Polder integral relates the $C_{6,AB}$ dispersion coefficient between two subsystems *A* and *B* to the product of their fluctuating polarizabilities at imfreq $\sqrt{-1}\omega$ integrated over all imfreqs:[25]

$$C_{6,AB} = \frac{3}{\pi}\int_0^\infty \alpha_A\left(\sqrt{-1}\,\omega\right)\alpha_B\left(\sqrt{-1}\,\omega\right)\mathrm{d}\omega \tag{8}$$

As shown in eqn (8), the integration limits are zero to infinity. For convenience, we used the substitution of variables

$$u = \frac{N\text{imfreqs}}{1+\omega} \quad \text{and} \quad \omega(u) = \frac{N\text{imfreqs}}{u} - 1 \tag{9}$$

to make both integration limits finite. Differentiating eqn (9) yields

$$\mathrm{d}\omega = -\frac{N\text{imfreqs}}{u^2}\mathrm{d}u \tag{10}$$

Table 1 Effect of dipole interaction cutoff length on the MCLF computed properties of graphene. Results (in atomic units) are per carbon atom

	10 bohr	25 bohr	50 bohr	75 bohr	100 bohr
C_6	37.91	50.33	**53.36**	53.80	53.90
α^{static}	11.42	18.13	**22.73**	24.82	26.01
$\alpha^{\text{low\_freq}}$	9.75	11.81	**12.28**	12.35	12.36
$\alpha^{\text{force-field}}$	7.03	7.03	**7.03**	7.03	7.03

Table 2 Effect of Romberg integration order (G) and number of integration points (Nimfreqs $= 2^G$) on the computed atom-in-material C_6 coefficients in atomic units

$G \rightarrow$	3	4	5
Nimfreqs $\rightarrow$	8	**16**	32
Graphene	53.33	**53.36**	53.36
K bcc solid	444.29	**448.35**	447.00
NaF solid	11.69 (Na)	**11.69 (Na)**	11.69 (Na)
	45.13 (F)	**45.26 (F)**	45.26 (F)
C_{60}	29.57	**29.67**	29.67

which upon substitution into eqn (8) for $A = B$ gives

$$C_{6,A} = \frac{3}{\pi}\int_0^{N\text{imfreqs}} \frac{(\alpha_A(u))^2 N\text{imfreqs}}{u^2}\,\mathrm{d}u \tag{11}$$

$u = 0$ corresponds to infinite imfreq. Near infinite imfreq, the polarizability becomes inversely proportional to ω^2.[26] This is incorporated into the TS-SCS[2,4] and MCLF[1] methods using a Padé approximation[27] for the dynamic unscreened polarizability:

$$\alpha_A^{\text{unscreened}}(u) = \frac{\alpha^{\text{unscreened}}}{1 + (\omega(u)/\text{wp})^2} \tag{12}$$

At this limit, the integrand in eqn (11) simplifies to

$$\lim_{u\to 0}\frac{(\alpha_A(u))^2 N\text{imfreqs}}{u^2} \approx \lim_{u\to 0}\frac{(\text{constant}/\omega^2)^2 N\text{imfreqs}}{u^2} \tag{13}$$

Substituting eqn (9) into (13) gives

$$\lim_{u\to 0}\frac{(\text{constant})^2 N\text{imfreqs}}{u^2\left(\frac{N\text{imfreqs}}{u}-1\right)^4} = \lim_{u\to 0}\frac{(\text{constant})^2 N\text{imfreqs}}{\left(\frac{N\text{imfreqs}}{u}-1\right)^2 (N\text{imfreqs}-u)^2} = 0 \tag{14}$$

Therefore, the $u = 0$ point contributes nothing to the integral.

We numerically integrated using Richardson extrapolation (*i.e.*, Romberg integration).[28–30] Dividing the (0, 1) interval into 2^G segments and performing Romberg integration of order (G, G) yields an integration error of the order $2^{-G(2G+2)}$.[28] Normally, Romberg integration of 2^G segments corresponds to $2^G + 1$ integration points. Since the $u = 0$ point contributes nothing to the integral, this leaves only Nimfreqs $= 2^G$ nontrivial integration points. The ESI† contains the Romberg integration weights $c_{G,u}^{\text{Romberg}}$ of these 2^G integration points for $G = 1$ to 5. The integral is computed using the following sum:

$$C_{6,A} = \sum_{u=1}^{N\text{imfreqs}} c_{G,u}^{\text{Romberg}}\left(\frac{N\text{imfreqs}\cdot\alpha_A(u)}{u}\right)^2 \tag{15}$$

where

$$\sum_{u=0}^{N\text{imfreqs}} c_{G,u}^{\text{Romberg}} = 1 \tag{16}$$

Table 2 shows computed AIM C_6 coefficients for four materials. We used the same geometries and electron densities for K solid, NaF solid, and C_{60} fullerene as in the companion article.[1] The K bcc solid showed a small difference (0.3%) in the C_6 coefficient between 16 and 32 integration points. All other results were virtually identical for 16 and 32 integration points. Moreover, all results for 8 integration points differed by <1% from the higher integration points. This shows the results are highly converged for 16 integration points, which is the value we chose.

2.4 Lookup table for computing $C_6 = \text{sum}(C_{6,AB})$

Although the mixed $C_{6,AB}$ dispersion coefficients could in principle be computed from the Casimir–Polder integral using $\alpha_A(u)$ and $\alpha_B(u)$, this would involve many integrations for unit cells containing thousands or more atoms. Therefore, we used the following mixing formula which is consistent with both Padé approximation[27] and QDO[31] models:

$$C_{6,AB} = \frac{2\alpha_A^{\text{low\_freq}}\alpha_B^{\text{low\_freq}} C_{6,A} C_{6,B}}{\left(\alpha_B^{\text{low\_freq}}\right)^2 C_{6,A} + \left(\alpha_A^{\text{low\_freq}}\right)^2 C_{6,B}} \tag{17}$$

Table 3 Accuracy of wp lookup table computation of $C_6^{\text{unscreened}}$ and C_6^{screened} compared to directly itemized computation. Num_lookup $= 10^5$. Unsigned relative errors (UREs) are listed in parts per trillion (ppt $= 10^{-12}$). Computational times (in seconds) for serial execution on the Comet cluster are listed. The column labeled PBC is the number of periodic boundary conditions. Natoms is the number of atoms per unit cell

			Unscreened			Screened		
Material	PBC	Natoms	URE (ppt)	Itemized (s)	Lookup (s)	URE (ppt)	Itemized (s)	Lookup (s)
C_{60}	0	60	0.002	0.000	0.005	0.002	0.000	0.005
$C_{50}H_{24}$	0	74	0.02	0.000	0.005	0.1	0.000	0.005
B-DNA	3	733	13.6	0.003	0.012	14.5	0.003	0.012
KUCDIW	3	1104	0.7	0.008	0.006	0.9	0.007	0.006
Graphene	2	2	0.001	0.000	0.005	0.003	0.000	0.005
Graphene	2	20 000	1.2	2.6	0.007	0.6	2.6	0.006
Ice	3	12	0.5	0.000	0.005	0.7	0.000	0.005
Ice	3	165 888	0.4	177	0.30	0.7	176	0.27
Ice	3	263 424	0.5	447	0.87	0.7	444	0.77
Ice	3	1 053 696	0.3	7859	3.7	0.6	7929	3.4
Ice	3	2 107 392	1.4	31 368	7.5	4.0	31 719	6.7

Table 4 Effect of the number of Richardson extrapolations steps (RES) applied to the MCLF screening increments. Graphene was chosen as a test system, because it has strong long-range dipole–dipole coupling. For (5,7) the results for a 20 000 atom supercell are shown in parentheses. Results are per atom

	(5,5)[a] RES	(6,6)[a] RES	(7,7)[a] RES	(5,7)[a] RES	(7,7)[a] GEPP[b]
α^{static}	22.62	22.72	22.73	**22.73 (22.73)**	22.73
$\alpha^{\text{low\_freq}}$	12.28	12.28	12.28	**12.28 (12.28)**	12.28
C_6	53.36	53.36	53.36	**53.36 (53.36)**	53.36
$\alpha^{\text{force-field}}$	7.03	7.03	7.03	**7.03 (7.03)**	7.03
Relative computational cost	17	34	68	**20**[c]	Cubic

[a] The first number in parentheses refers to the number of RES used for non-directional screening and short-range directional screening to compute $\alpha_A^{\text{non-dir}}(u)$ and $\alpha_A^{\text{screened}}(u)$, respectively. The second number in parentheses is the number of RES used for long-range directional screening to compute $\overrightarrow{\overrightarrow{\alpha}}_A^{\text{static}}$. [b] Explicit large matrix construction and inversion using Gaussian elimination with partial pivoting. [c] Nominal computational cost for (5,7) algorithm includes 16 frequency points at 5 RES plus the static polarizability at 7 RES, which gives 16(1) + 1(4) = 20 compared to 17 for the (5,5) algorithm.

In our method, the polarizabilities appearing in eqn (17) must be $\alpha^{\text{low\_freq}}$, because these are the polarizabilities associated with the dispersion interaction. Of note, the TS and TS-SCS methods use similar mixing formula, except the polarizabilities appearing in the mixing formula are α^{TS} and $\alpha^{\text{TS-SCS}}$,[2,4] because those methods do not yield $\alpha^{\text{low\_freq}}$.

When using the mixing rule of eqn (17), itemized computation of the C_6 dispersion coefficient per unit cell requires a total

Begin input file reading
Read the following quantities needed to perform the unscreened scaling calculations: Natoms; atomic numbers $\{\text{Element}_A\}$ or element symbols; net atomic charges $\{N_A\}$, Rcubed, Rfourth, and weighted_Rfourth atom-in-material radial moments
Read the following quantities needed to perform the screening calculations: periodic lattice vectors (if any), (X, Y, Z) coordinates of each atom, atom-in-material volumes $\{V_A\}$
If element symbols were read, then convert to atomic numbers $\{\text{Element}_A\}$
Calculate the polarizability upper bound for every atom: $\alpha_A^{\text{upper\_bound}} = V_A/(2\pi)$
Calculate the number of electrons for every atom: $N_A = \text{Element}_A - q_A$
End input file reading

Begin unscreened scaling
Loop over atoms in the unit cell A = 1, 2, … Natoms
Calculate $m = \text{weighted\_Rfourth\_moment}(A)/\text{Rfourth\_moment}(A)$
Calculate $\alpha^{\text{unscreened}}(A)$ using the m-scaling model equation for polarizability
Calculate $\text{wp}(A)$ using the m-scaling model equation for wp
Calculate $\text{unscreened\_C6}(A) = 0.75\left(\alpha^{\text{unscreened}}(A)\right)^2 \text{wp}(A)$
Calculate $\text{unscreened\_Rdamp}(A) = \text{Rdamp\_ref}\left(\text{Element}(A)\right)\left(\frac{\text{unscreened\_C6}(A)}{\text{C6\_ref}\left(\text{Element}(A)\right)}\right)^{1/9}$
End loop
$\text{total\_unscreened\_polarizability} = \text{SUM}(\alpha^{\text{unscreened}})$
IF (Natoms < threshold_wp_lookup) THEN
Initialize $\text{total\_unscreened\_C6} = \text{SUM}(\text{unscreened\_C6})$
Loop over atoms in the unit cell A = 1, 2, … Natoms (this loop index is parallelized)
Loop over atoms in the unit cell B = (A+1), (A+2), … Natoms
$\text{total\_unscreened\_C6} = \text{total\_unscreened\_C6} + \frac{4\left(\text{unscreened\_C6}(A)\right)\left(\text{unscreened\_C6}(B)\right)\left(\alpha^{\text{unscreened}}(A)\right)\left(\alpha^{\text{unscreened}}(B)\right)}{\left(\alpha^{\text{unscreened}}(B)\right)^2 \text{unscreened\_C6}(A) + \left(\alpha^{\text{unscreened}}(A)\right)^2 \text{unscreened\_C6}(B)}$
End loop
End loop
ELSE
$\text{total\_unscreened\_C6} = \text{C6tot}\left(\alpha^{\text{unscreened}}, \text{wp}, \text{Natoms}, \text{Num\_lookup}\right)$
END IF
End unscreened scaling

Fig. 1 Pseudocode for the input file reading and unscreened calculation using *m*-scaling. The *m*-scaling model equations for polarizability and wp are given in the companion article.[1]

Note: This algorithm starts with the unscreened m-scaled results (i.e., $\{\alpha^{\text{unscreened}}(A)\}$ *and* $\{\text{wp}(A)\}$ *) and small list array as inputs, with* $\Lambda = 2^{1/6}\pi^{-1/6}3^{-1/3}$ and $\gamma = 4/(3\sqrt{\pi})$ already assigned.

Begin non-directional screening
  Initialize $\{\alpha^{\text{non-dir}}(A,u)\} = 0$
  Loop over the imfreq points u = 1, 2, … Nimfreqs
    Calculate $\omega(u) = (\text{Nimfreqs}/u) - 1$
    For all atoms, initialize $\alpha^{\text{partial}}(A) = \alpha^{\text{unscreened}}(A)/\left(1+(\omega(u)/\text{wp}(A))^2\right)$
    For each atom image pair (A,b) in the small list, calculate and store in the small list array:
      $\text{pair\_ratio}_{Ab} = \alpha^{\text{partial}}(A)/(\alpha^{\text{partial}}(A)+\alpha^{\text{partial}}(B))$
    Loop over the Richardson extrapolation steps s = 0, 1, … K
      Calculate $\Delta = 1/2^s$
      For all atoms, initialize $\alpha^{\text{partial}}(A) = \alpha^{\text{unscreened}}(A)/\left(1+(\omega(u)/\text{wp}(A))^2\right)$
      Loop over the screening increments j=1, 2, … 2^s
        Initialize temp_vector(:) = 0, where length(temp_vector) = Natoms
        For all atoms, calculate $\sigma(A) = (\alpha^{\text{partial}}(A))^{1/3}\Lambda$
        Loop over all atom image pairs $\{(A,b)\}$ in the 'small' list (this loop index is parallelized)
          Read atom numbers for A & B, distance between atom A & image b, f^{cutoff}, and pair_ratio from the small list array
          Calculate $\sigma^{\text{pair}} = \sqrt{(\sigma(A))^2+(\sigma(B))^2}$
          Calculate $x^{\text{pair}} = \text{distance}/\sigma^{\text{pair}}$
          If $x^{\text{pair}} > 5$, then cycle
          Calculate $\text{temp} = (2f^{\text{cutoff}}\gamma\Delta)\exp\left(-\left((x^{\text{pair}})^2\right)\right)\alpha^{\text{partial}}(A)\alpha^{\text{partial}}(B)/\left((\sigma^{\text{pair}})^3\right)$
          Calculate $\text{temp\_vector}(A) = \text{temp\_vector}(A) + (\text{pair\_ratio})(\text{temp})$
          Calculate $\text{temp\_vector}(B) = \text{temp\_vector}(B) + (1-\text{pair\_ratio})(\text{temp})$
        End loop
        For all atoms: $\alpha^{\text{partial}}(A) = \alpha^{\text{partial}}(A) - \text{temp\_vector}(A)$
      End loop
      For all atoms: $\alpha^{\text{non-dir}}(A,u) = \alpha^{\text{non-dir}}(A,u) + c^{\text{extrap}}(K+1,s+1)\alpha^{\text{partial}}(A)$
    End loop
  End loop
  Apply conduction limit upper bound for all atoms and imfreq points:
    $\alpha^{\text{non-dir}}(A,u) = \text{smooth\_min}\left(\alpha^{\text{upperbound}}(A)/\left(1+(\omega(u)/\text{wp}(A))^2\right), \alpha^{\text{non-dir}}(A,u)\right)$
  Assign $\alpha^{\text{force-field}}(A) = \alpha^{\text{non-dir}}(A,\text{Nimfreqs})$
End non-directional screening

Fig. 2 Pseudo-code for non-directional screening to compute $\{\alpha_A^{\text{non-dir}}(u)\}$ and the force-field polarizabilities $\{\alpha_A^{\text{force-field}}\}$ using inverse-free algorithm and Richardson extrapolation.

of (*N*atoms)(*N*atoms − 1)/2 individual $C_{6,AB}$ evaluations. This itemized approach to computing C_6^{total} would be computationally expensive for unit cells containing a billion or more atoms. Atomistic (*e.g.*, molecular dynamics or Monte Carlo) simulations employing an interaction cutoff distance require only a small subset of $\{C_{6,AB}\}$ for large unit cells, because atoms outside the cutoff distance do not interact. Therefore, it would be extremely wasteful to compute and print all the individual $C_{6,AB}$ values for large unit cells. It is preferred to compute and print $\{\alpha_A^{\text{low\_freq}}\}$ and $\{C_{6,A}\}$ that can be inserted into eqn (17) to generate selected $C_{6,AB}$ whenever needed. For a billion atoms, storing $\{\alpha_A^{\text{low\_freq}}\}$ and $\{C_{6,A}\}$ requires only 16 gigabytes (GB), while storing $\{C_{6,AB}\}$ would require 4 exabytes. Therefore, it is unnecessary for the program to compute, store, or print the full $\{C_{6,AB}\}$.

A linear-scaling algorithm to compute C_6^{total} is now introduced to address this issue. Defining

$$\text{wp}_A^{\text{screened}} = \frac{4C_{6,A}}{3\left(\alpha_A^{\text{low\_freq}}\right)^2} \tag{18}$$

allows eqn (17) to be re-written as the following variant of the London formula

$$C_{6,AB} = \frac{3}{2}\alpha_A^{\text{low\_freq}}\alpha_B^{\text{low\_freq}}\frac{\text{wp}_A^{\text{screened}}\text{wp}_B^{\text{screened}}}{\text{wp}_A^{\text{screened}}+\text{wp}_B^{\text{screened}}} \tag{19}$$

and analogously for the unscreened values

$$C_{6,AB}^{\text{unscreened}} = \frac{3}{2}\alpha_A^{\text{unscreened}}\alpha_B^{\text{unscreened}}\frac{\text{wp}_A\text{wp}_B}{\text{wp}_A+\text{wp}_B} \tag{20}$$

Linear-scaling computation is achieved by constructing a lookup table that uniformly spaces ln(wp_values):

$$x_1 = \ln(\min(\text{wp\_values})) - 10^{-2} \tag{21}$$

Note 1: This algorithm starts with the non-directional screened results (i.e., $\{\alpha^{\text{non-dir}}(A,u)\}$), small list array, and large list array as inputs, with $\Lambda = 2^{1/6}\pi^{-1/6}3^{-1/3}$ and $\gamma = 4/(3\sqrt{\pi})$ already assigned. Note 2: Because each tensor is symmetric, only six components are unique and required to be stored: xx, xy=yx, xz=zx, yy, yz=zy, zz .

Begin imfreq directional screening

Initialize $\{\alpha^{\text{screened}}(A,u)\} = 0$ (See note 2 above.)

Loop over the imfreq points u = 1, 2, … Nimfreqs

For each atom image pair (A,b) in the small list, calculate and store in the small list array:

$$\text{pair\_ratio}_{Ab} = \alpha^{\text{non-dir}}(A,u)/\left(\alpha^{\text{non-dir}}(A,u)+\alpha^{\text{non-dir}}(B,u)\right)$$

For each atom pair (A,B) in the large list, calculate and store in the large list array:

$$\text{pair\_ratio}_{AB} = \alpha^{\text{non-dir}}(A,u)/\left(\alpha^{\text{non-dir}}(A,u)+\alpha^{\text{non-dir}}(B,u)\right)$$

Loop over the Richardson extrapolation steps s = 0, 1, … K

Calculate $\Delta = 1/2^s$

For all atoms, initialize $\alpha^{\text{partial}}(A) = \alpha^{\text{non-dir}}(A,u)$,

$$\vec{\vec{\alpha}}^{\text{partial}}(A) = \text{diag}\left(\alpha^{\text{partial}}(A),\alpha^{\text{partial}}(A),\alpha^{\text{partial}}(A)\right)$$

Loop over the screening increments j=1, 2, … 2^s

For all atoms, initialize $\vec{\vec{t}}(A) = 0$ (See note 2 above.)

For all atoms, calculate $\sigma(A) = \left(\alpha^{\text{partial}}(A)\right)^{1/3}\Lambda$

Loop over all atom image pairs $\{(A,b)\}$ in the 'small' list (this loop index is parallelized)

Read atom numbers for A & B, distance between atom A & image b, f^{cutoff}, f^{MBS}, pair_ratio, and $\vec{\vec{\eta}}^{AB}$ from the small list array

Calculate $\sigma^{\text{pair}} = \sqrt{(\sigma(A))^2+(\sigma(B))^2}$

Calculate $x^{\text{pair}} = \text{distance}/\sigma^{\text{pair}}$

If $x^{\text{pair}} > 5$, then cycle

Calculate:

$$\text{temp} = \left(f^{\text{cutoff}} f^{\text{MBS}}\Delta\right)\left(\frac{\text{erfc}\left(x^{\text{pair}}\right)}{\text{distance}^3}+(\gamma)\left(\frac{3}{2\sigma^{\text{pair}}\text{distance}^2}+\frac{1}{\left(\sigma^{\text{pair}}\right)^3}\right)\exp\left(-\left(\left(x^{\text{pair}}\right)^2\right)\right)\right)$$

Calculate $\vec{\vec{w}} = \vec{\vec{\alpha}}^{\text{partial}}(A)\bullet\vec{\vec{\eta}}^{AB}\bullet\vec{\vec{\alpha}}^{\text{partial}}(B)$

Calculate $\vec{\vec{w}} = \left(\vec{\vec{w}}+\text{transpose}\left(\vec{\vec{w}}\right)\right)\text{temp}$

Calculate $\vec{\vec{t}}(A) = \vec{\vec{t}}(A)+(\text{pair\_ratio})\vec{\vec{w}}$

Calculate $\vec{\vec{t}}(B) = \vec{\vec{t}}(B)+(1-\text{pair\_ratio})\vec{\vec{w}}$

End loop

Loop over all atom pairs $\{(A,B)\}$ in the 'large' list (this loop index is parallelized)

Read atom numbers for A & B, pair_ratio, and $\vec{\vec{h}}^{AB}$ from the large list array

Calculate $\vec{\vec{w}} = \vec{\vec{\alpha}}^{\text{partial}}(A)\bullet\vec{\vec{h}}^{AB}\bullet\vec{\vec{\alpha}}^{\text{partial}}(B)$

Calculate $\vec{\vec{w}} = \left(\vec{\vec{w}}+\text{transpose}\left(\vec{\vec{w}}\right)\right)\Delta$

Calculate $\vec{\vec{t}}(A) = \vec{\vec{t}}(A)+(\text{pair\_ratio})\vec{\vec{w}}$

Calculate $\vec{\vec{t}}(B) = \vec{\vec{t}}(B)+(1-\text{pair\_ratio})\vec{\vec{w}}$

End loop

For all atoms: $\vec{\vec{\alpha}}^{\text{partial}}(A) = \vec{\vec{\alpha}}^{\text{partial}}(A)-\vec{\vec{t}}(A)$

For all atoms: $\alpha^{\text{partial}}(A) = \text{trace}\left(\vec{\vec{\alpha}}^{\text{partial}}(A)\right)/3$

End loop

For all atoms: $\alpha^{\text{screened}}(A,u) = \alpha^{\text{screened}}(A,u)+c^{\text{extrap}}(K+1,s+1)\alpha^{\text{partial}}(A)$

End loop

End loop

For all atoms: $\alpha^{\text{low-freq}}(A) = \alpha^{\text{screened}}(A,\text{Nimfreqs})$

End imfreq directional screening

Fig. 3 Pseudocode for imfreq directional screening to compute $\{\alpha_A^{\text{screened}}(u)\}$ and $\{\alpha_A^{\text{low\_freq}}\}$ using inverse-free algorithm and Richardson extrapolation.

Note 1: This algorithm starts with the non-directional screened results (i.e., $\{\alpha^{\text{force-field}}(A)\}$), small list array, and large list array as inputs, with $\Lambda = 2^{1/6}\pi^{-1/6}3^{-1/3}$ and $\gamma = 4/(3\sqrt{\pi})$ already assigned. Note 2: Because each tensor is symmetric, only six components are unique and required to be stored: xx, xy=yx, xz=zx, yy, yz=zy, zz .

Begin directional static screening

Initialize $\{\vec{\vec{\alpha}}^{\text{static}}(A)\} = 0$ (See note 2 above.)

For each atom image pair (A,b) in the small list, calculate and store in the small list array:

$$\text{pair\_ratio}_{Ab} = \alpha^{\text{force-field}}(A)/\left(\alpha^{\text{force-field}}(A)+\alpha^{\text{force-field}}(B)\right)$$

For each atom pair (A,B) in the large list, calculate and store in the large list array:

$$\text{pair\_ratio}_{AB} = \alpha^{\text{force-field}}(A)/\left(\alpha^{\text{force-field}}(A)+\alpha^{\text{force-field}}(B)\right)$$

Loop over the Richardson extrapolation steps s = 0, 1, … K

Calculate $\Delta = 1/2^{s}$

For all atoms, initialize $\alpha^{\text{partial}}(A) = \alpha^{\text{force-field}}(A)$,

$$\vec{\vec{\alpha}}^{\text{partial}}(A) = \text{diag}\left(\alpha^{\text{partial}}(A),\alpha^{\text{partial}}(A),\alpha^{\text{partial}}(A)\right)$$

Loop over the screening increments j=1, 2, … 2^{s}

For all atoms, initialize $\vec{\vec{t}}(A) = 0$ (See note 2 above.)

For all atoms, calculate $\sigma(A) = \left(\alpha^{\text{partial}}(A)\right)^{1/3}\Lambda$

Loop over all atom image pairs $\{(A,b)\}$ in the 'small' list (this loop index is parallelized)

Read atom numbers for A & B, distance between atom A & image b, f^{cutoff}, pair_ratio, and $\vec{\vec{\eta}}^{AB}$ from the small list array

Calculate $\sigma^{\text{pair}} = \sqrt{(\sigma(A))^2+(\sigma(B))^2}$

Calculate $x^{\text{pair}} = \text{distance}/\sigma^{\text{pair}}$

If $x^{\text{pair}} > 5$, then cycle

Calculate

$$\text{temp} = \left(f^{\text{cutoff}}\Delta\right)\left(\frac{\text{erfc}\left(x^{\text{pair}}\right)}{\text{distance}^3}+(\gamma)\left(\frac{3}{2\sigma^{\text{pair}}\text{distance}^2}+\frac{1}{\left(\sigma^{\text{pair}}\right)^3}\right)\exp\left(-\left(\left(x^{\text{pair}}\right)^2\right)\right)\right)$$

Calculate $\vec{\vec{w}} = \vec{\vec{\alpha}}^{\text{partial}}(A)\bullet\vec{\vec{\eta}}^{AB}\bullet\vec{\vec{\alpha}}^{\text{partial}}(B)$

Calculate $\vec{\vec{w}} = \left(\vec{\vec{w}}+\text{transpose}\left(\vec{\vec{w}}\right)\right)\text{temp}$

Calculate $\vec{\vec{t}}(A) = \vec{\vec{t}}(A)+(\text{pair\_ratio})\vec{\vec{w}}$

Calculate $\vec{\vec{t}}(B) = \vec{\vec{t}}(B)+(1-\text{pair\_ratio})\vec{\vec{w}}$

End loop

Loop over all atom pairs $\{(A,B)\}$ in the 'large' list (this loop index is parallelized)

Read atom numbers for A & B, pair_ratio, and $\vec{\vec{g}}^{AB}$ from the large list array

Calculate $\vec{\vec{w}} = \vec{\vec{\alpha}}^{\text{partial}}(A)\bullet\vec{\vec{g}}^{AB}\bullet\vec{\vec{\alpha}}^{\text{partial}}(B)$

Calculate $\vec{\vec{w}} = \left(\vec{\vec{w}}+\text{transpose}\left(\vec{\vec{w}}\right)\right)\Delta$

Calculate $\vec{\vec{t}}(A) = \vec{\vec{t}}(A)+(\text{pair\_ratio})\vec{\vec{w}}$

Calculate $\vec{\vec{t}}(B) = \vec{\vec{t}}(B)+(1-\text{pair\_ratio})\vec{\vec{w}}$

End loop

For all atoms: $\vec{\vec{\alpha}}^{\text{partial}}(A) = \vec{\vec{\alpha}}^{\text{partial}}(A)-\vec{\vec{t}}(A)$

For all atoms: $\alpha^{\text{partial}}(A) = \text{trace}\left(\vec{\vec{\alpha}}^{\text{partial}}(A)\right)/3$

End loop

For all atoms: $\vec{\vec{\alpha}}^{\text{static}}(A) = \vec{\vec{\alpha}}^{\text{static}}(A)+c^{\text{extrap}}(K+1,s+1)\vec{\vec{\alpha}}^{\text{partial}}(A)$ (See note 2 above.)

End loop

For all atoms: $\alpha^{\text{static}}(A) = \text{trace}\left(\vec{\vec{\alpha}}^{\text{static}}(A)\right)/3$

Perform the anisotropic polarizability correction: $\vec{\vec{\alpha}}^{\text{static}}(A) = (1-\text{C.F.})\vec{\vec{\alpha}}^{\text{static}}(A)+(\text{C.F.})\alpha^{\text{static}}(A)\vec{\vec{I}}$

Calculate unit cell polarizability: $\vec{\vec{\alpha}}^{\text{unit\_cell}} = \sum_{A}\vec{\vec{\alpha}}^{\text{static}}(A)$ and $\alpha^{\text{unit\_cell}} = \sum_{A}\alpha^{\text{static}}(A)$

Calculate eigenvalues of $\{\vec{\vec{\alpha}}^{\text{static}}(A)\}$ and $\vec{\vec{\alpha}}^{\text{unit\_cell}}$

End directional static screening

Fig. 4 Pseudocode for directional static screening to compute the static polarizability tensors $\{\vec{\vec{\alpha}}_A^{\text{static}}\}$ using inverse-free algorithm and Richardson extrapolation.

Note: The Romberg integration order is set such that Nimfreqs=2^{Rom_order}.

Begin screened dispersion coefficients and QDO parameters calculations

Initialize $\mathrm{screened\_C6(A)} = 0$ and $\mathrm{nondir\_C6(A)} = 0$

Loop over the imfreq points u = 1, 2, … Nimfreqs

Calculate $\mathrm{temp} = (\mathrm{Nimfreqs}/\mathrm{u})^2\,(3/\pi)\,\mathrm{Rom\_coeff}(\mathrm{Rom\_order}, \mathrm{u})$

Loop over atoms in the unit cell A = 1, 2, … Natoms

Calculate: $\mathrm{screened\_C6(A)} = \mathrm{screened\_C6(A)} + \left(\alpha^{\mathrm{screened}}(\mathrm{A},\mathrm{u})\right)^2 \mathrm{temp}$

Calculate: $\mathrm{nondir\_C6(A)} = \mathrm{nondir\_C6(A)} + \left(\alpha^{\mathrm{non-dir}}(\mathrm{A},\mathrm{u})\right)^2 \mathrm{temp}$

End loop

End loop

Loop over atoms in the unit cell A = 1, 2, … Natoms

Calculate $\mathrm{C8(A)} = \mathrm{e}^{1.7327}\left(\mathrm{nondir\_C6(A)}\right)^{3.8305/3}\left(\dfrac{\mathrm{Rfourth\_moment(A)}}{\mathrm{Rcubed\_moment(A)}}\right)^{0.339}$

Calculate $\mathrm{C10(A)} = \left(\dfrac{49}{40}\right)\dfrac{(\mathrm{C8(A)})^2}{\mathrm{nondir\_C6(A)}}$

Calculate $\mathrm{wp\_QDO(A)} = \dfrac{4\mathrm{nondir\_C6(A)}}{3\left(\alpha^{\mathrm{force-field}}(\mathrm{A})\right)^2}$

Calculate $\mathrm{m\_QDO(A)} = \dfrac{15\left(\alpha^{\mathrm{force-field}}(\mathrm{A})\right)^2}{4\mathrm{C8(A)}}$

Calculate $\mathrm{q\_QDO(A)} = -\sqrt{\dfrac{20\left(\mathrm{nondir\_C6(A)}\right)^2}{3\alpha^{\mathrm{force-field}}(\mathrm{A})\mathrm{C8(A)}}}$

Calculate $\mathrm{wp\_screened(A)} = \dfrac{4\mathrm{screened\_C6(A)}}{3\left(\alpha^{\mathrm{low\_freq}}(\mathrm{A})\right)^2}$

End loop

IF (Natoms < threshold_wp_lookup) THEN

Initialize $\mathrm{total\_screened\_C6} = \mathrm{SUM(screened\_C6)}$

Loop over atoms in the unit cell A = 1, 2, … Natoms (this loop index is parallelized)

Loop over atoms in the unit cell B = (A+1), (A+2), … Natoms

$$\mathrm{total\_screened\_C6} = \mathrm{total\_screened\_C6} + \frac{4\left(\mathrm{screened\_C6(A)}\right)\left(\mathrm{screened\_C6(B)}\right)\left(\alpha^{\mathrm{low\_freq}}(\mathrm{A})\right)\left(\alpha^{\mathrm{low\_freq}}(\mathrm{B})\right)}{\left(\alpha^{\mathrm{low\_freq}}(\mathrm{B})\right)^2 \mathrm{screened\_C6(A)} + \left(\alpha^{\mathrm{low\_freq}}(\mathrm{A})\right)^2 \mathrm{screened\_C6(B)}}$$

End loop

End loop

ELSE

$\mathrm{total\_screened\_C6} = \mathrm{C6tot}\left(\alpha^{\mathrm{low\_freq}}, \mathrm{wp\_screened}, \mathrm{Natoms}, \mathrm{Num\_lookup}\right)$

END IF

End screened dispersion coefficients and QDO parameters calculations

Fig. 5 Pseudocode for computing the screened dispersion coefficients and QDO parameters.

$$x_{\mathrm{Num\_lookup}} = \ln(\max(\mathrm{wp\_values})) + 10^{-2} \quad (22)$$

$$\mathrm{Interval} = x_i - x_{i-1} = \frac{x_{\mathrm{Num\_lookup}} - x_1}{\mathrm{Num\_lookup} - 1} \quad (23)$$

$$\mathrm{wp\_table}_i = \exp(x_i) \quad (24)$$

The $\pm 10^{-2}$ in eqn (21) and (22) ensures the interval always remains non-zero even if max(wp_values) = min(wp_values), and it also ensures the floor operation in eqn (25) will not produce an index j < 1 or > Num_lookup even in the presence of fixed precision floating point round-off. An array alpha_table having Num_lookup components is initialized to zero. A loop is then performed over all atoms in the unit cell. Each atom A's polarizability α_A contributes to two adjacent array values:

$$j = \mathrm{floor}\left(\frac{\ln(\mathrm{wp\_values}_A) - x_1}{\mathrm{interval}} + 1\right) \quad (25)$$

Table 5 Effect of unit cell size on the calculated results and number of FCR iterations required to converge TS-SCS/DDEC6 calculation. The convergence threshold was 10^{-5} as the maximum absolute value of any conditioned residual component. The listed number of FCR iterations to converge was the maximum for any one direction at a single imfreq point. The total FCR iterations and large matrix-vector multiplies are summed over all imfreq points and directions. GEPP results are also listed for the primitive unit cells. The polarizabilities and C_6 dispersion coefficients are in atomic units: (a) per carbon atom for graphene and (b) per water molecule for ice

	Graphene			Ice		
	2 atom primitive cell		20 000 atom supercell	12 atom primitive cell		2 107 392 atom supercell
Algorithm	GEPP	FCR	FCR	GEPP	FCR	FCR
α^{static}	20.12	20.12	20.12	8.86	8.86	8.86
C_6	108.66	108.66	108.66	42.48	42.48	42.49
FCR iterations to converge	—	1	1	—	6	6
Total FCR iterations	—	48	48	—	181	181
Large matrix-vector multiplies	—	192	192	—	724	724

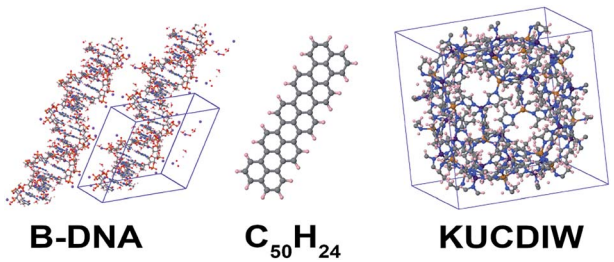

Fig. 6 Three diverse chemical structures used to study the TS-SCS convergence properties: the B-DNA decamer, the $C_{50}H_{24}$ polyacene, and the KUCDIW metal–organic framework. The atoms are colored by chemical element: grey (C), pink (H), blue (N), purple (Na), red (O), orange (P), copper (Cu), indigo (B).

$$c_{j+1} = \frac{\text{wp\_values}_A - \text{wp\_table}_j}{\text{wp\_table}_{j+1} - \text{wp\_table}_j} \quad (26)$$

$$c_j = 1 - c_{j+1} \quad (27)$$

$$\text{alpha\_table}_j \leftarrow \text{alpha\_table}_j + c_j\alpha_A \quad (28)$$

$$\text{alpha\_table}_{j+1} \leftarrow \text{alpha\_table}_{j+1} + c_{j+1}\alpha_A \quad (29)$$

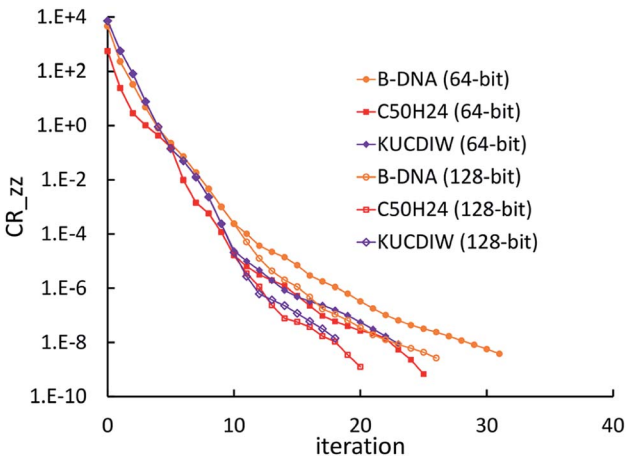

Fig. 7 Plot of the conditioned residual norm (*i.e.*, CR_zz) *versus* the FCR iteration number for three diverse materials: the B-DNA decamer, the $C_{50}H_{24}$ polyacene, and the KUCDIW metal–organic framework. The conditioned residual norm decreased rapidly and monotonically with increasing iteration number. The size of the real numbers (64-bit or 128-bit) is indicated.

After this loop completes, C_6^{total} is computed by the following double summation (that involves an outer loop over lookup table entries and an inner loop over atoms in the unit cell):

Table 6 TS-SCS(FCR) convergence performance for three diverse materials: (a) the B-DNA decamer, (b) the $C_{50}H_{24}$ polyacene, and (c) the KUCDIW metal–organic framework. The convergence threshold was 10^{-5} as the maximum absolute value of any conditioned residual component. The listed number of FCR iterations to converge was the maximum for any one direction at a single imfreq point. The total FCR iterations and large matrix-vector multiplies are summed over all imfreq points and directions. The total computational time is for a serial (*i.e.*, one computing core) run on the Comet cluster. Data for 64-bit (128-bit) reals is shown outside (inside) parentheses

	B-DNA decamer	$C_{50}H_{24}$ polyacene	KUCDIW metal–organic framework
Atoms per unit cell	733	74	1104
FCR iterations to converge	31 (26)	25 (20)	23 (18)
Total FCR iterations	483 (417)	360 (323)	390 (347)
Large matrix-vector multiplies	1932 (1668)	1440 (1292)	1560 (1388)
Total computational time in seconds	58 (104)	0.57 (0.97)	107 (195)

```
!$omp parallel default(none) &
!$omp private( . . . list of private variables . . . ) &
!$omp shared( . . . list of shared variables . . . ) &
!$omp reduction(+:MMp,MMq)
!$omp do schedule(static)
   DO i=1,num_sym_unique_atom_image_pairs_small
        .
        .
        .
   END DO
!$omp end do
!$omp do schedule(static)
   DO i=1,num_sym_unique_atom_image_pairs_large
        .
        .
        .
   END DO
!$omp end do
!$omp end parallel
```

Fig. 8 Excerpt of OpenMP parallelized Fortran code that performs the operations $\mathbf{MM}p(:,1) = \mathbf{M} \times \mathbf{M}p(:)$ and $\mathbf{MM}q(:,1) = \mathbf{M} \times \mathbf{M}q(:)$. Threads are created by the *parallel* directive. The directive *default(none)* specifies that each variable used in the parallel region must be declared as either private or shared. Reductions are performed over $\mathbf{MM}p$ and $\mathbf{MM}q$. The *do schedule(static)* directive assigns loop values to different threads in a round-robin manner.

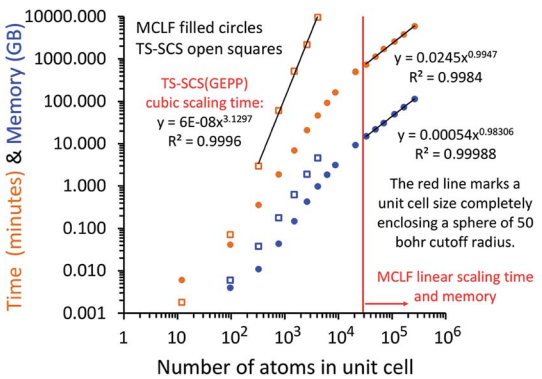

Fig. 9 Plot of required computational time and random access memory (RAM) to perform MCLF analysis on ice crystals containing different numbers of atoms in the periodic unit cell: 12, 96, 324, 768, 1500, 2592, 4116, 6144, 8748, 20 736, 32 928, 49 152, 69 984, 111 132, 165 888, and 263 424. For 12 atoms, the required RAM was <1 MB. Beyond a certain system size (governed by the dipole interaction cutoff length), the required computational time and memory scale linearly with increasing system size. TS-SCS results using GEPP for 12 to 4116 atoms per unit cell are also shown for comparison. Because of direct matrix inversion, the TS-SCS method using GEPP has nearly cubic scaling computational cost, which makes it infeasible for large unit cells. Each calculation time is the average of three runs in serial mode.

$$C_6^{\text{total}} = \sum_{i=1}^{\text{Num\_lookup}} \left(\frac{3}{2} (\text{alpha\_table}_i) \sum_{k=1}^{N\text{atoms}} \left[(\alpha_k) \frac{\text{wp\_table}_i \text{wp\_values}_k}{\text{wp\_table}_i + \text{wp\_values}_k} \right] \right) \quad (30)$$

The code skips the inner loop over atoms whenever $\text{alpha\_table}_i = 0$.

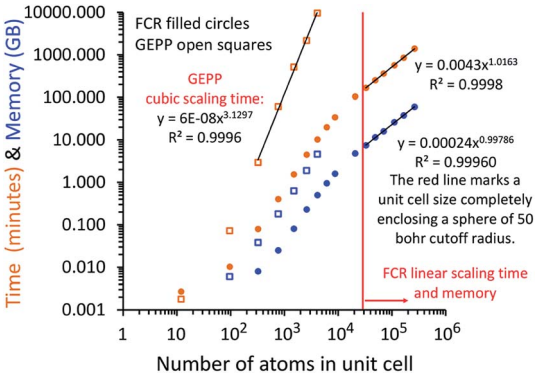

Fig. 10 Comparison of failproof conjugate residual (FCR) and Gaussian elimination with partial pivoting (GEPP) algorithms for performing TS-SCS analysis. The same ice supercells were studied as listed in the caption of Fig. 9. The required RAM was <1 MB for 12 atoms (GEPP and FCR) and 96 atoms (FCR). Beyond a certain system size (governed by the dipole interaction cutoff length), the required computational time and memory for the FCR algorithm scale linearly with increasing system size. In contrast, the GEPP algorithm has nearly cubic scaling computational cost, which makes it infeasible for large unit cells. Each calculation time is the average of three runs in serial mode.

Since each atom contributes to exactly two alpha_table components (eqn (28) and (29)), the total number of loop iterations performed is always $\leq \min(2(N\text{atoms}^2), (N\text{atoms} \times \text{Num\_lookup}))$ (eqn (30)) + Natoms (eqn (25)–(29)) + Num_lookup (eqn (24)). For Natoms > Num_lookup this method exhibits linear-scaling computational cost, because as Natoms increases only the size of the inner loop in eqn (30) increases linearly while the outer loop over lookup table entries has fixed size. For systems containing a small number of symmetry distinct atoms, each symmetry identical atom contributes to the same two alpha_table components, so the number of non-zero alpha_table components never exceeds twice the number of symmetry distinct atoms, which results in super acceleration for computations on large supercells comprised of many symmetry equivalent primitive cells (*e.g.*, see graphene and ice in Table 3).

The ESI† contains a Fortran file that implements this algorithm (eqn (21)–(30)) as a function C_6^{total}(alpha_values, wp_values, Natoms, Num_lookup). This function parallelizes eqn (30) using OpenMP. Using appropriate arguments, this same function can be called to compute either $C_6^{\text{unscreened}}$ or C_6^{screened}.

This method has astonishingly high precision. As proved in Section S3 of the ESI,† the unsigned relative error (URE) in C_6^{total} is always less than interval^2/16, which is $\leq \sim 4 \times \text{Num\_lookup}^{-2}$. Thus, with Num_lookup $= 10^5$, the URE is usually $\leq \sim 4 \times 10^{-10}$. Table 3 confirms this high computational precision for several example materials.

All real variables were 64-bit, except C_6^{total} (both eqn (2) and (30)), alpha_table, and a temporary variable to accumulate the inner sum of eqn (30)) were 128-bit reals. The need for a 128-bit real (aka 'quadruple precision') for the C_6^{total} variable is straightforward to demonstrate. For a unit cell containing one billion similar atoms, $C_6^{\text{total}} \approx 10^{18} C_{6,A}$. Since 64-bit reals have

Table 7 Calculation time breakdown in seconds for MCLF analysis of ice crystals. Each time is the average of three runs. The standard deviation is based on the total time per run

Atoms in unit cell →	324	1500	4116	8748	20 736	49 152	111 132	263 424
Input file reading	0.01	0.3	0.4	0.1	0.2	0.8	1.6	3.3
Unscreened α & C_6	0.1	0.1	0.3	0.9	3.4	16.6	79.5	12.8
Atom image pair matrix initialization	0.4	2.3	8.6	31.0	115.3	394.1	1174.0	2610.9
Non-directional screening	0.4	1.9	5.3	12.8	28.0	66.1	164.3	360.2
Fluctuating screening	16.2	329.7	2230.2	7878.4	23 807.5	54 089.5	123 403.6	281 602.7
Static α screening	4.1	82.4	557.3	1923.0	5929.8	13 226.3	29 717.1	69 835.3
Total	21.7	417.1	2798.0	9848.4	29 890.5	67 815.5	154 627.4	355 216.5
Standard deviation	0.9%	2.7%	0.1%	0.2%	0.5%	1.1%	1.4%	0.7%

approximately 15 significant base-ten digits, attempting to accumulate C_6^{total} in the traditional manner by adding $C_{6,AB}$ one-at-a-time would cause the accumulation to stall at $C_6^{\text{total}} \approx 10^{15} C_{6,A}$ which would be off by a factor of $\sim 10^3$. For a unit cell containing one million atoms, accumulating C_6^{total} one atom pair at a time using a 64-bit real number will not stall, but the final working capacity will be reduced to $\sim$3 precision digits: 15 (digits for 64-bit reals) − 12 digits (for $C_6^{\text{total}}/C_{6,A} \approx 10^{12}$) = $\sim$3 remaining digits of precision for individual $C_{6,AB}$ addition. 128-Bit reals have approximately 34 significant base-ten digits, which is adequate. Although using 128-bit reals for alpha_table and the temporary variable are not required, this helps preserve quadruple precision during C_6^{total} computation.

Computational times in Table 3 refer to the time required to call and compute the C_6^{total} function (*i.e.*, eqn (2) and (17) or (21)–(30)) and do not include times required to compute function inputs. As shown in Table 3, the wp lookup table computation can sometimes be slower than directly itemized computation when *N*atoms is small but is always faster when *N*atoms is large. Therefore, the program is normally configured to perform directly itemized computation of C_6^{total} when *N*atoms < threshold_wp_lookup, and wp lookup table computation otherwise. For all calculations in this paper except those listed in Table 3, we set threshold_wp_lookup = 2 × Num_lookup. When *N*atoms > $\sim$2 × Num_lookup, the lookup table algorithm always requires fewer iterations than directly itemized computation.

This same wp lookup table method for computing C_6^{total} is also employed in the TS-SCS(FCR) method described in Section 3 below. The same Num_lookup = 10^5 and threshold_wp_lookup = 2 × Num_lookup are used there. The same function described above is used, but called with the alpha_values and wp_values appropriate for computing C_6^{TS} or $C_6^{\text{TS-SCS}}$ for the TS-SCS(FCR) method.

Table 9 Parallelization efficiency for MCLF analysis of ice crystals

Atoms in unit cell	Number of processors				
	1	2	4	8	16
324	113%	112%	107%	88%	73%
1500	113%	111%	109%	104%	100%
4116	108%	107%	105%	98%	103%
8748	121%	112%	117%	111%	114%
20 736	110%	118%	116%	107%	100%
49 152	111%	113%	112%	102%	109%
111 132	115%	105%	111%	106%	108%
263 424	112%	111%	111%	108%	104%

Table 10 Parallelization efficiency for TS-SCS analysis of ice crystals using FCR algorithm

Atoms in unit cell	Number of processors				
	1	2	4	8	16
324	98%	97%	93%	74%	58%
1500	103%	99%	97%	91%	89%
4116	97%	97%	96%	95%	94%
8748	101%	99%	97%	92%	95%
20 736	98%	98%	95%	98%	98%
49 152	100%	100%	93%	93%	96%
111 132	97%	100%	97%	96%	94%
263 424	101%	102%	101%	99%	97%

Table 8 Calculation time breakdown in seconds for TS-SCS analysis of ice crystals. Each time is the average of three runs. The standard deviation is based on the total time per run

Atoms in unit cell →	324	1500	4116	8748	20 736	49 152	111 132	263 424
Input file reading	0.07	0.01	0.04	0.03	0.07	0.8	0.3	0.6
Unscreened α & C_6	0.1	0.1	0.3	0.8	3.4	16.7	75.7	11.6
Atom image pair matrix initialization	0.2	1.7	6.8	25.1	106.1	365.1	1033.3	2534.6
TS-SCS screening	4.4	91.0	608.6	2027.7	6170.3	14 779.4	32 851.3	81 153.8
Total	4.8	92.7	615.8	2053.7	6279.9	15 162.0	33 960.5	83 700.7
Standard deviation	0.5%	2.3%	0.2%	1.6%	1.3%	0.4%	0.4%	2.9%

Table 11 Comparison of RAM required in GB for serial and parallel program execution. Results are listed for both MCLF and TS-SCS(FCR) analysis of ice crystals

Atoms in unit cell	MCLF		TS-SCS(FCR)	
	Serial	8 core	Serial	8 core
324	0.011	0.012	0.008	0.008
1500	0.15	0.15	0.081	0.082
4116	0.98	0.98	0.50	0.53
8748	3.2	3.2	1.6	1.6
20 736	9.3	9.3	4.8	4.8
49 152	22	22	11.5	11.5
111 132	50	50	26	26
263 424	115	115	60	60

2.5 Avoiding direct inversion of large matrices

Direct inversion of large matrices is extremely computationally expensive. Gaussian elimination with partial pivoting (GEPP) is a common matrix inversion algorithm that exhibits numerical instability for some matrices but is usually stable in practice.[32,33] GEPP, QR factorization, Cholesky and LDL decomposition (for positive definite Hermitian matrices), and other common matrix inversion algorithms have computational costs scaling proportional to the number of rows (*N*rows) cubed.[11,33] The more complicated Strassen algorithm has scaling proportional to $N\text{rows}^{\log(7)/\log(2)} = N\text{rows}^{2.807\ldots}$.[11]

A direct matrix inversion algorithm would first construct the dipole interaction tensor, then invert it to get the multibody polarizability matrix, and then contract the multibody polarizability matrix to get the AIM polarizability tensors.[34] Consider a material containing 1 million atoms in the unit cell. For this material, the dipole interaction tensor has 3 million rows and an equal number of columns. The computational cost of the matrix inversion step would be on the order of $N\text{rows}^3 = (3 \times 10^6)^3 = \sim 2.7 \times 10^{19}$ floating point operations (*i.e.*, ~27 exaflop). This corresponds to ~7500 computational hours on a teraflop computer or ~7.5 computational hours on a petaflop computer for each imfreq point. Since the dipole interaction tensor would need to be inverted at several (*e.g.*, 16) imfreq points, a TS-SCS or MCLF calculation on this material would be computationally prohibitive when using direct matrix inversion. Moreover, storing the full multi-body polarizability matrix in double-precision arithmetic would take (8 bytes per double precision real) × (3 × 10^6 rows) × (3 × 10^6 columns) = 7.2 × 10^{13} bytes = 72 terabytes of random access memory (RAM). This is a huge amount of memory.

To address this problem, we developed a new computational algorithm that converges to the same solution without requiring any matrix inversions. This inverse-free algorithm is conceptually related to the iterative Schulz method for matrix inversion. In the Schulz method, an estimate $\mathbf{P}^{(i-1)}$ for the inverse of matrix $\mathbf{Q}$ is iteratively refined by[35]

$$\mathbf{P}^{(i)} = 2\mathbf{P}^{(i-1)} - \mathbf{P}^{(i-1)}\mathbf{Q}\mathbf{P}^{(i-1)} \tag{31}$$

where index i is the Schulz iteration. The key difference between our inverse-free algorithm and the Schulz matrix inversion method is that we exploit the particular structures of the dipole interaction tensor and multi-body polarizability matrix contraction to enable us to work with *N*atoms scalars for the non-directional screening and *N*atoms 3 × 3 matrices for the directional screening instead of working with *N*atoms × *N*atoms and 3*N*atoms × 3*N*atoms matrices for matrix inversions using GEPP or Schulz method. This allows us to reduce the computational cost from cubic scaling (for GEPP or Schulz method using conventional matrix multiplications) to linear in *N*atoms as the unit cell becomes sufficiently large.

Substituting $\mathbf{Q}_{j+1} = \mathbf{D}_j + \Delta_j\tau_j$ (see companion article[1] for specific definitions of $\mathbf{D}_j$ and τ_j) into (31) with $i = 1$ gives the first Schulz iteration as

$$\mathbf{P}_{j+1}{}^{(1)} = 2\mathbf{P}_{j+1}^{(0)} - \mathbf{P}_{j+1}^{(0)}(\mathbf{D}_j + \Delta_j\tau_j)\mathbf{P}_{j+1}^{(0)} \tag{32}$$

where

$$\mathbf{P}_{j+1}^{(0)} = \text{inv}(\mathbf{D}_j) \tag{33}$$

is the initial estimate for inv($\mathbf{Q}_{j+1}$). The subscript $j + 1$ is the screening iteration and should not be confused with the Schulz iteration (superscript i). Δ_j is the screening increment. Substituting eqn (33) into (32) and simplifying yields

$$\mathbf{P}_{j+1}{}^{(1)} = \text{inv}(\mathbf{D}_j) - \text{inv}(\mathbf{D}_j)(\Delta_j\tau_j)\text{inv}(\mathbf{D}_j) \tag{34}$$

Note that: (1) τ_j includes non-zero blocks only for interacting atom pairs (*i.e.*, atom pairs in the 'small' and/or 'large' lists), (2) $\mathbf{D}_j$ and inv($\mathbf{D}_j$) are block-diagonal matrices, (3) the non-zero blocks of inv($\mathbf{D}_j$) are the partially screened atomic polarizabilities $\{\alpha_j^A\}$ (which are scalar for non-directional screening and tensor for fluctuating and static screening), and (4) the multibody polarizability matrix $\mathbf{P}$ is only needed in its contracted form as partially screened atomic polarizabilities $\{\alpha_j^A\}$.

Three different types of atomic polarizabilities are computed using this inverse-free algorithm. As explained in the companion article,[1] atoms with larger pre-screened polarizability get a proportionally larger piece of the screening mixed polarizability contribution:

$$\alpha_A^{\text{non-dir}}(u) = \sum_B \mathbf{P}^{AB}(u)\left(\frac{2\alpha_A^{\text{unscreened}}(u)}{\alpha_A^{\text{unscreened}}(u) + \alpha_B^{\text{unscreened}}(u)}\right) \tag{35}$$

$$\mathbf{P}_{st}{}^{A}(u) = \sum_B \left(\frac{\alpha_A^{\text{non-dir}}(u)}{\alpha_A^{\text{non-dir}}(u) + \alpha_B^{\text{non-dir}}(u)}\left(\mathbf{P}_{st}{}^{AB}(u) + \mathbf{P}_{ts}{}^{AB}(u)\right)\right) \tag{36}$$

$$\mathbf{P}_{st}{}^{A} = \sum_B \left(\frac{\alpha_A^{\text{force-field}}}{\alpha_A^{\text{force-field}} + \alpha_B^{\text{force-field}}}\left(\mathbf{P}_{st}{}^{AB} + \mathbf{P}_{ts}{}^{AB}\right)\right) \tag{37}$$

where s and t represent spatial directions $\in \{x, y, z\}$. Eqn (35), (36), and (37) correspond to the non-directional, fluctuating, and static polarizabilities, respectively. Using these to simplify eqn (34) yields

$$\alpha_{j+1}{}^{A} = \alpha_j{}^{A} - \sum_b \left[\left(\frac{\alpha_{\text{in}}{}^{A}}{\alpha_{\text{in}}{}^{A} + \alpha_{\text{in}}{}^{B}} \right) \left(\alpha_j{}^{A} \left(\Delta_j \tau_j{}^{Ab} \right) \alpha_j{}^{B} + \alpha_j{}^{B} \left(\Delta_j \tau_j{}^{Ab} \right) \alpha_j{}^{A} \right) \right] \quad (38)$$

For non-directional screening to compute $\alpha_A^{\text{non-dir}}(u)$, eqn (38) becomes

$$\alpha_{j+1}{}^{A}(u) = \alpha_j{}^{A}(u) - \sum_{b \in \text{small\_list}} \left[\left(\frac{2\alpha_A^{\text{unscreened}}(u)}{\alpha_A^{\text{unscreened}}(u) + \alpha_B^{\text{unscreened}}(u)} \right) \times \left(\alpha_j{}^{A}(u) \left(\Delta_j \tau_{\text{non-dir},j}{}^{Ab}(u) \right) \alpha_j{}^{B}(u) \right) \right] \quad (39)$$

$\tau_{\text{non-dir},j}{}^{Ab}(u)$ is defined in the companion article,[1] where the attenuation length for the pair of atoms A and B

$$\sigma_{AB,j}(u) = \sqrt{\sigma_{A,j}{}^{2}(u) + \sigma_{B,j}{}^{2}(u)} \quad (40)$$

is updated at the start of screening iteration $j + 1$ using $\alpha_j{}^{A}(u)$ and $\alpha_j{}^{B}(u)$ to compute the spherical Gaussian dipole width

$$\sigma_{A,j}(u) = \left(\sqrt{\frac{2}{\pi}} \ \frac{\alpha_j{}^{A}(u)}{3} \right)^{1/3} \quad (41)$$

The process starts with

$$\alpha_{j=1}{}^{A}(u) = \alpha_A^{\text{unscreened}}(u) \quad (42)$$

and ends with $\alpha_A^{\text{non-dir}}(u)$ as the value on the left-side of eqn (39) after the last screening increment finishes.

For directional screening to compute $\alpha_A^{\text{screened}}(u)$, eqn (38) becomes

$$\mathbf{M1} = \left(\Delta_j f_{\text{MBS}}{}^{Ab} f_{\text{cutoff}}{}^{Ab} \varphi_j{}^{Ab}(u) \right) \left(\vec{\vec{\alpha}}_j{}^{A}(u) \cdot \left(\vec{\vec{\eta}}{}^{Ab}(u) \right) \cdot \vec{\vec{\alpha}}_j{}^{B}(u) \right) \quad (43)$$

$$\mathbf{M2} = \Delta_j \vec{\vec{\alpha}}_j{}^{A}(u) \cdot \vec{\vec{h}}{}^{AB}(u) \cdot \vec{\vec{\alpha}}_j{}^{B}(u) \quad (44)$$

$$\vec{\vec{\alpha}}_{j+1}{}^{A}(u) = \vec{\vec{\alpha}}_j{}^{A}(u) - \sum_{b \in \text{small\_list}} \left[\left(\frac{\alpha_A^{\text{non-dir}}(u)}{\alpha_A^{\text{non-dir}}(u) + \alpha_B^{\text{non-dir}}(u)} \right) (\mathbf{M1} + \mathbf{M1}^{\text{T}}) \right] - \sum_{B \in \text{large\_list}} \left[\left(\frac{\alpha_A^{\text{non-dir}}(u)}{\alpha_A^{\text{non-dir}}(u) + \alpha_B^{\text{non-dir}}(u)} \right) (\mathbf{M2} + \mathbf{M2}^{\text{T}}) \right] \quad (45)$$

where **M1** and **M2** are square matrices with 3 rows and

$$\varphi_j{}^{Ab}(u) = -\left(\frac{1}{(d_{Ab})^3} \text{erfc}\left(\frac{d_{Ab}}{\sigma_{AB,j}(u)} \right) + \left(\frac{2}{\sqrt{\pi}} \right) \left(\frac{1}{(d_{Ab})^2 \sigma_{AB,j}(u)} + \frac{2}{3(\sigma_{AB,j}(u))^3} \right) \exp\left(-\left(\frac{d_{Ab}}{\sigma_{AB,j}(u)} \right)^2 \right) \right) \quad (46)$$

$$\alpha_j{}^{A}(u) = \text{trace}\left(\vec{\vec{\alpha}}_j{}^{A}(u) \right) / 3 \quad (47)$$

In eqn (46), the attenuation length $\sigma_{AB,j}(u)$ is updated at the start of screening iteration $j + 1$ using $\alpha_j{}^{A}(u)$ and $\alpha_j{}^{B}(u)$ from eqn (47) inserted into eqn (40) and (41). The process starts with

$$\vec{\vec{\alpha}}_{j=1}{}^{A}(u) = \begin{bmatrix} \alpha_A^{\text{non-dir}}(u) & 0 & 0 \\ 0 & \alpha_A^{\text{non-dir}}(u) & 0 \\ 0 & 0 & \alpha_A^{\text{non-dir}}(u) \end{bmatrix} \quad (48)$$

and ends with $\alpha_A^{\text{screened}}(u)$ as one-third the trace of the tensor on the left-side of eqn (45) after the last screening increment finishes.

For directional screening to compute $\vec{\vec{\alpha}}_A^{\text{static}}$, eqn (38) becomes

$$\mathbf{M1} = \left(\Delta_j f_{\text{cutoff}}{}^{Ab} \varphi_j{}^{Ab} \right) \left(\vec{\vec{\alpha}}_j{}^{A} \cdot \left(\vec{\vec{\eta}}{}^{Ab} \right) \cdot \vec{\vec{\alpha}}_j{}^{B} \right) \quad (49)$$

$$\mathbf{M2} = \Delta_j \vec{\vec{\alpha}}_j{}^{A} \cdot \vec{\vec{g}}{}^{AB} \cdot \vec{\vec{\alpha}}_j{}^{B} \quad (50)$$

$$\vec{\vec{\alpha}}_{j+1}{}^{A} = \vec{\vec{\alpha}}_j{}^{A} - \sum_{b \in \text{small\_list}} \left[\left(\frac{\alpha_A^{\text{force-field}}}{\alpha_A^{\text{force-field}} + \alpha_B^{\text{force-field}}} \right) (\mathbf{M1} + \mathbf{M1}^{\text{T}}) \right] - \sum_{B \in \text{large\_list}} \left[\left(\frac{\alpha_A^{\text{force-field}}}{\alpha_A^{\text{force-field}} + \alpha_B^{\text{force-field}}} \right) (\mathbf{M2} + \mathbf{M2}^{\text{T}}) \right] \quad (51)$$

where $\varphi_j{}^{Ab}$ and $\alpha_j{}^{A}$ are defined analogous to eqn (46) and (47) except based on $\vec{\vec{\alpha}}_j{}^{A}$ from eqn (51). The attenuation length $\sigma_{AB,j}(u = N\text{imfreqs})$ is updated at the start of screening iteration $j + 1$ using $\alpha_j{}^{A}$ and $\alpha_j{}^{B}$ inserted into eqn (40) and (41). The process starts using eqn (48) and ends with $\vec{\vec{\alpha}}_A^{\text{pre-corrected}}$ as the tensor on the left-side of eqn (51) after the last screening increment finishes. As explained in the companion article,[1] the following anisotropic polarizability correction is then applied to get the static polarizability tensors

$$\vec{\vec{\alpha}}_A^{\text{static}} = (1 - \text{C.F.}) \vec{\vec{\alpha}}_A^{\text{pre-corrected}} + (\text{C.F.}) \alpha_A^{\text{static}} \quad (52)$$

$$\alpha_A^{\text{static}} = \text{trace}\left(\vec{\vec{\alpha}}_A^{\text{pre-corrected}} \right) / 3 = \text{trace}\left(\vec{\vec{\alpha}}_A^{\text{static}} \right) / 3 \quad (53)$$

using a correction factor (C.F.) = 0.2. Summing $\vec{\vec{\alpha}}_A^{\text{static}}$ over all atoms in the unit cell gives the unit cell (or molecular) polarizability tensor. Taking one-third the trace of the AIM or molecular static polarizability tensors yields the corresponding isotropic static polarizabilities.

The order of the error residual is now derived. Since $\mathbf{P}_{j+1}{}^{(1)}$ is a calculated estimate of inv($\mathbf{Q}_{j+1}$), we begin by defining the error residual as

$$\text{Residual} = \| \mathbf{I} - \mathbf{Q}_{j+1} \mathbf{P}_{j+1}{}^{(1)} \| \quad (54)$$

where $\|\|$ is any desired matrix norm and **I** is the identity matrix. Multiplying eqn (34) by $\mathbf{Q}_{j+1} = \mathbf{D}_j + \Delta_j \tau_j$ gives

$$\mathbf{Q}_{j+1} \mathbf{P}_{j+1}{}^{(1)} = \mathbf{I} - \Delta_j \tau_j \text{inv}(\mathbf{D}_j)(\Delta_j \tau_j) \text{inv}(\mathbf{D}_j) \quad (55)$$

Multiplying each side of eqn (55) by −1, adding **I**, and taking the norm gives

$$\|\mathbf{I} - \mathbf{Q}_{j+1}\mathbf{P}_{j+1}{}^{(1)}\| = \|\Delta_j\tau_j\text{inv}(\mathbf{D}_j)\Delta_j\tau_j\text{inv}(\mathbf{D}_j)\| = (\Delta_j)^2\|\tau_j\text{inv}(\mathbf{D}_j)\,\tau_j\text{inv}(\mathbf{D}_j)\| \quad (56)$$

The right-most side follows from the theorem that norm(scalar × matrix) = scalar × norm(matrix).[36] For convenience, we used fixed size screening increments: $\Delta_j = \Delta$. Eqn (56) shows that at each screening iteration, the difference between this inverse-free algorithm and direct matrix inversion will be on the order of Δ^2, where Δ is the screening increment. Since the total number of screening iterations is $1/\Delta$, the total difference between this inverse-free algorithm and direct matrix inversion will be on the order of Δ.

By taking the limit $\Delta \to 0$, the inverse-free and direct matrix inversion algorithms converge to the identical solution. We used Richardson extrapolation[30,37] to evaluate the limit $\Delta \to 0$. As explained in the prior paragraph, without Richardson extrapolation the overall error using screening increment Δ is of *Order*(Δ). Each Richardson extrapolation step removes a successive power of Δ in the error. After K Richardson extrapolation steps (RES), the remaining error will thus be of *Order*(Δ^{K+1}). We extrapolated using screening increments of 1, 2^{-1}, 2^{-2}, …2^{-K}. During each such screening process, the screening increments sum to 1

$$\sum_j \Delta_j = 1 \quad (57)$$

Therefore, this extrapolation corresponds to extrapolating from results computed with 1, 2, 2^2, …2^K screening points. The final $\Delta = 2^{-K}$ undergoes K RES leading to a residual error of *Order*($2^{-K(K+1)}$). Note that 7 RES is approximately twice as expensive as 6 RES and four times as expensive as 5 RES.

Richardson extrapolation was applied to the following atomic polarizabilities:

Non-directional screening:

$$\alpha_A^{\text{non-dir}}(u) = \sum_{\xi=1}^{K+1} c_{(K+1),\xi}^{\text{extrap}}\,\alpha_A^{\text{non-dir},\xi}(u) \quad (58)$$

Frequency-dependent directional screening:

$$\alpha_A^{\text{screened}}(u) = \sum_{\xi=1}^{K+1} c_{(K+1),\xi}^{\text{extrap}}\,\alpha_A^{\text{screened},\xi}(u) \quad (59)$$

Static induced directional screening:

$$\overrightarrow{\boldsymbol{\alpha}}_A^{\text{static}} = \sum_{\xi=1}^{K+1} c_{(K+1),\xi}^{\text{extrap}}\,\overrightarrow{\boldsymbol{\alpha}}_A^{\text{static},\xi} \quad (60)$$

As explained above, $\alpha_A^{\text{force-field},\xi}$, $\alpha_A^{\text{screened},\xi}(u)$, $\overrightarrow{\boldsymbol{\alpha}}_A^{\text{static},\xi}$ are the values computed using $2^{\xi-1}$ screening points (*i.e.*, screening increment of $2^{1-\xi}$). The coefficients $c_{(K+1),\xi}^{\text{extrap}}$ for $K = 1$ to 7 RES are given in the ESI.† Note that

$$\alpha^{\text{low\_freq}} = \alpha_A^{\text{screened}}(u = N\text{imfreqs}) \quad (61)$$

$$\alpha^{\text{force-field}} = \alpha_A^{\text{non-dir}}(u = N\text{imfreqs}) \quad (62)$$

Table 4 lists computed results for graphene. After testing, we settled on five RES for the force-field polarizabilities (*i.e.*, non-directional screening), five RES for the fluctuating polarizabilities used to compute C_6 coefficients, and seven RES for the static polarizabilities. The static polarizability requires more RES than the force-field and fluctuating polarizabilities, because of the longer range dipole–dipole interactions contributing to the static polarizability. The inverse-free and GEPP algorithms converged to the same results in the limit $\Delta \to 0$. The inverse-free algorithm is preferable, because it exhibits better computational cost scaling than GEPP. As an additional test, the (5,7) RES calculation was repeated using a 20 000 atom supercell (which is a 100 × 100 replication of the 2 atom primitive unit cell) which shows the MCLF results for a large supercell are identical to those computed for a primitive unit cell. The 20 000 atom graphene supercell was constructed using DDEC6 AIM properties from the primitive cell; no DFT calculation on the large supercell was needed.

2.6 Code design and parallelization

The MCLF code was parallelized using OpenMP. OpenMP divides the work among different computing cores on a single cache-coherent node.[38,39] The Fortran code was parallelized by adding OpenMP compiler directives that parallelized the most computationally intense loops. OpenMP codes can be compiled in serial mode by simply instructing the compiler to ignore the OpenMP directives. We used several techniques for efficient OpenMP parallelization discussed in one of our previous articles.[12] Specifically, the array indices were ordered in a cache friendly manner, large arrays were declared as shared variables to avoid large memory increases when adding more parallel threads, and when needed reductions were used with the *parallel* directive rather than with the *do schedule* directive (see Section 3.3 for example).[12] Directives such as *single*, *atomic*, and *critical* that require one thread to wait on another should be kept to a minimum.[12,38,39] To maximize parallelization efficiency, each thread should be given enough work such that the overhead time to set up the threads is a small percentage of the parallel region time.[12,38,39] For example, when parallel work is done over the 'small' and 'large' atom pair lists (*e.g.*, Fig. 3 and 4), the parallel threads were created just prior to looping over the 'small' list and terminated just after looping over the 'large' list.

Paradoxically, the computer code to perform MCLF analysis is actually simpler than the mathematical equations that define MCLF analysis. The reason is that many array indices that appear in the mathematical equations are actually not required in the computer code, because these quantities can be computed in-place. For this reason, pseudocodes for the main parts of MCLF analysis are illustrated in this section. Fig. 1 illustrates the pseudocode for input file reading and unscreened calculation using m-scaling. Fig. 2, 3, and 4 illustrate pseudocodes for Richardson extrapolation with inverse-free algorithms used to compute

$\{\alpha_A^{\text{non-dir}}(u)\}$, $\{\alpha_A^{\text{screened}}(u)\}$, and $\{\overrightarrow{\alpha}_A^{\text{static}}\}$, respectively. The array indices in these figures correspond to what is actually needed in the computer program: (1) the atom number appears as an array index rather than as a subscript, (2) the screening increment does not appear as an array index because temporary results for each screening increment are computed in-place, (3) the imfreq integration point u appears as an array index only where needed, *etc.* Fig. 5 illustrates a pseudocode for computing the screened dispersion coefficients and QDO parameters. On each figure, the OpenMP parallelized loops are marked with '(this loop index is parallelized)'.

3. Linear-scaling TS-SCS algorithm

3.1 Overview and problem statement

The TS-SCS calculation consists of four main parts: (a) input file reading and unscreened calculation using the TS scaling law[4] to compute α^{TS} and C_6^{TS} for each atom, (b) setting up the 'small' and 'large' atom pair lists, (c) self-consistent screening to computed the TS-SCS polarizability at each imfreq point, and (d) Casimir–Polder integration to compute the TS-SCS C_6 coefficients. Just as for MCLF, we used Romberg integration with 16 imfreq points to compute C_6 for TS-SCS.

Step (b) follows a procedure analogous to that described in Section 2.2 with the following differences. For the 'small' list, similar information was stored for TS-SCS as for MCLF, except $f_{\text{MBS}}^{A,b}$ does not need to be stored for TS-SCS. For the 'large' list, only $\overrightarrow{g}^{AB}$ needs to be stored and $\overrightarrow{h}^{AB}$ does not need to be stored for TS-SCS. This allowed each pair in the 'large' list for TS-SCS to be stored in a single cache line (*i.e.*, 8 double-precision real numbers), while two cache lines were used to store data for each pair in the 'large' list of MCLF analysis. (Stored data for each pair in the 'large' list of MCLF analysis required slightly less than two cache lines, but this was rounded up to two whole cache lines to give different atom pairs their own cache lines.) Using whole cache lines provides a slight simplification for tasks that parallelize over atom pairs in the 'large' list.)

For MCLF analysis, the smooth cutoff function described in the companion article was used.[1] This same smooth cutoff function could also be used for TS-SCS. However, for consistency with prior literature,[40] we used the sharp cutoff function

$$f_{\text{cutoff}}(d_{Ab}) = (1 - H(d_{Ab} - d_{cutoff})) \tag{63}$$

for TS-SCS analysis, where $d_{Ab} = r^{AB,L}$ is the distance between atom A and the image b in bohr. d_{cutoff} is the dipole interaction cutoff length. H is the Heaviside step function. In this article as well as the companion one,[1] we set d_{cutoff} to 50 bohr.

Eqn (1) is analogous to solving for induced dipole moments in a polarizable force-field

$$\sum_{t=1}^{3}\sum_{B=1}^{N\text{atoms}}\left(\frac{\delta_{st}}{\alpha_A^{\text{force-field}}} - \tau_{st}{}^{AB}(u)\right)\overrightarrow{\mu}_B^{\text{induced}} = \vec{E}_A^0 \tag{64}$$

where $\vec{E}_A^0$ is the electric field acting on site A due to permanent charges, permanent multipoles, and externally applied electric field. $\overrightarrow{\mu}_{\text{B}}^{\text{induced}}$ is the dipole moment induced on site B. $\alpha_A^{\text{force-field}}$ is the polarizability of site A. Specifically, setting $\alpha_A^{\text{force-field}}$ to $\alpha_A^{\text{unscreened}}(u)$ and $\vec{E}_A^0$ to (1, 0, 0) yields $\overrightarrow{\mu}_{\text{B}}^{\text{induced}}$ that gives the first column of $\overrightarrow{\overrightarrow{\alpha}}_B^{\text{SCS}}(u)$. Solving again with $\vec{E}_A^0$ set to (0, 1, 0) yields $\overrightarrow{\mu}_{\text{B}}^{\text{induced}}$ that gives the second column of $\overrightarrow{\overrightarrow{\alpha}}_B^{\text{SCS}}(u)$. Finally, solving with $\vec{E}_A^0$ set to (0, 0, 1) yields $\overrightarrow{\mu}_{\text{B}}^{\text{induced}}$ that gives the third column of $\overrightarrow{\overrightarrow{\alpha}}_B^{\text{SCS}}(u)$. Hence, a mathematical procedure that is optimized for solving the polarizability eqn (64) can be an efficient way to solve the TS-SCS eqn (1).

As described by Applequist *et al.*, the coefficients matrix in the polarizability equation can be expressed as a two-dimensional array (*i.e.*, as a matrix with rows and columns) by combining spatial and atomic indices.[34] Since there are three spatial components (*i.e.*, x, y, and z) in the dipole moment for each atom, the coefficients matrix for the polarizability equation (see eqn (1) and (64)) has Nrows = 3 × Natoms and an equal number of columns.[34] This coefficients matrix is formed by splicing together Natoms × Natoms blocks, where each block is a 3 × 3 unit for the spatial indices.[34] From a programming perspective, it is simply a matter of choice as to whether to employ a two-dimensional array with combined atomic and spatial indices or to keep the atomic and spatial indices as distinct array dimensions. Equivalent mathematical problems arise in both cases. Regardless, a large-sized coefficients array arises when there is a large number of atoms in the unit cell.

Developing algorithms to solve large linear equation systems (*e.g.*, $\mathbf{Ax} = \mathbf{B}$) played a key role in the history of electronic computing.[41–44] (The typeface is used to distinguish $\mathbf{A}$ as a matrix from A as an atom.) Here, we are particularly interested in solving large systems of sparse linear equations, and many different algorithms have been described in the prior literature for doing this.[42,43,45,46] Krylov subspace techniques solve a linear equation system using iterations in which some error norm is minimized in a Krylov subspace whose order increases with iteration number.[47,48] A Krylov subspace of order n defined by matrix $\mathbf{M}$ and vector $\boldsymbol{W}$ is[48]

$$K_n(\mathbf{M},\boldsymbol{W}) = \text{span}[\boldsymbol{W}, \mathbf{M}\boldsymbol{W}, \mathbf{M}^2\boldsymbol{W}, \mathbf{M}^3\boldsymbol{W}, \ldots\mathbf{M}^{n-1}\boldsymbol{W}] \tag{65}$$

A key consideration is how to generate an orthonormal vector basis that spans the Krylov subspace sequence without having to store vectors from all prior iterations.[49,50] For a Hermitian matrix $\mathbf{M}$, short-term recurrences can be constructed that span the Krylov subspace while requiring only vectors from a few recent iterations to be explicitly stored (*e.g.*, from the current and two most recent prior iterations).[44,47,48] By definition, a Hermitian matrix equals its Hermitian conjugate (*i.e.*, complex conjugate transpose). Conjugate gradient algorithms are a widespread class of Krylov subspace techniques.[43,47,48,51]

Conditioning plays a key role in conjugate gradient type methods.[42,46,47,51,52] Conditioning multiplies the original coefficients matrix by one or more matrices to transform the original linear equation $\mathbf{Ax} = \mathbf{B}$ into a new linear equation $\mathbf{M}y = \boldsymbol{W}$ that is

easier to solve.[42,46,47,51,52] Either left multiplication or right multiplication, or a combination of both, can be used during conditioning.[42,47,51,52] Conditioning can accomplish one of more of the following:[42,47,51,52]

(i) If the original coefficients matrix **A** is not Hermitian, conditioning may make the new coefficients matrix **M** Hermitian., or

(ii) If the original coefficients matrix **A** is not positive (semi) definite, conditioning may make the new coefficients matrix **M** positive (semi)definite., or

(iii) If the original coefficients matrix **A** is not square, conditioning may make the new coefficients matrix **M** square. This is important for solving least-squares problems., or

(iv) To improve convergence speed, conditioning may make the eigenspectrum of **M** more clustered than that of **A**. After conditioning, conjugate gradient methods minimize some error norm within the Krylov subspace $K_n(\mathbf{M},\boldsymbol{W})$.

Recently, truncated conjugate gradient algorithms were used to solve for induced dipole moments, polarization energies, and the corresponding forces in polarizable force fields.[53,54] Aviat *et al.* used the Orthomin[47] conjugate gradient algorithm (also called Hestenes' and Stiefel's method[55]) truncated at fixed order to solve the multibody polarization equations efficiently without introducing spurious energy drifts during molecular dynamics simulations using polarizable force fields.[53,54] The key limitation of the Orthomin conjugate gradient algorithm is the coefficients matrix **M** in the linear equation $\mathbf{M}y = \boldsymbol{W}$ must be Hermitian positive definite.[47] Although the polarization eqn (64) contains a Hermitian coefficients matrix, we do not know whether or not it is always positive definite. The coefficients matrix is positive definite if and only if all of its eigenvalues are positive.[56] For many common circumstances the coefficients matrix in this polarization equation is likely to be positive definite, but we do not have any information at hand about exceptions that could potentially give an indefinite coefficients matrix in the polarization equation. If the coefficients matrix **M** is indefinite, the Orthomin conjugate gradient algorithm will fail if $\langle z^{(i)}|\mathbf{M}z^{(i)}\rangle = 0$ at any iteration, when $z^{(i)} \neq 0$.

Herein, $\mathbf{x}^{(i)}$ is the estimated **x** at iteration i, $r^{(i)}$ is the residual, and $z^{(i)}$ is the conditioned residual:[47]

$$r^{(i)} = \mathbf{B} - \mathbf{A}\mathbf{x}^{(i)} \tag{66}$$

$$z^{(i)} = \mathbf{C}r^{(i)} = \boldsymbol{W} - \mathbf{M}y^{(i)} \tag{67}$$

In this article, the dot product of two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ is defined as

$$\langle \boldsymbol{v}|\boldsymbol{w}\rangle = \boldsymbol{v}^{\mathrm{H}}\boldsymbol{w} = (\boldsymbol{v}^*)^{\mathrm{T}}\boldsymbol{w} = \sum_i \boldsymbol{v}_i^*\boldsymbol{w}_i \tag{68}$$

Three old conjugate gradient algorithms are Orthomin, Orthodir, and Orthores.[47,50] As mentioned above, Orthomin can fail for indefinite coefficient matrices. Orthores is algebraically equivalent to Orthomin, and Orthores converges if and only if Orthomin converges.[47]

The Orthodir conjugate gradient algorithm can work for both positive definite and indefinite Hermitian coefficients matrix **M**.[47] However, it suffers from the accumulation of round-off errors. In the Orthodir algorithm, the y-search direction at each successive iteration is computed as

$$p^{(i)} = \mathbf{M}p^{(i-1)} - \varsigma^{(i)}p^{(i-1)} - \vartheta^{(i)}p^{(i-2)} \tag{69}$$

where $\varsigma^{(i)}$ and $\vartheta^{(i)}$ are chosen to fulfill some chosen conjugacy condition.[47] In exact arithmetic, eqn (69) would enforce orthogonality between $\mathbf{M}p^{(i)}$, $\mathbf{M}p^{(i-1)}$, and $\mathbf{M}p^{(i-2)}$. Because the choice of direction $p^{(i)}$ does not explicitly depend on the residual's value (see eqn (69)), a buildup of round-off errors over many iterations can cause the chosen z-search direction $\mathbf{M}p^{(i)}$ to become uncorrelated to the residual's value. When this occurs, the Orthodir algorithm does not operate as intended and may fail to converge.

Conjugate gradient of the normal equations rearranges the linear equation system so that the coefficients matrix is $\mathbf{M} = \mathbf{A}^{\mathrm{H}}\mathbf{A}$, thereby ensuring a positive semidefinite Hermitian coefficients matrix **M**.[42,43,47] Because each eigenvalue of $\mathbf{M} = \mathbf{A}^{\mathrm{H}}\mathbf{A}$ is the squared magnitude of the corresponding eigenvalue of **A**, all eigenvalues of $\mathbf{M} = \mathbf{A}^{\mathrm{H}}\mathbf{A}$ are non-negative real-valued. Two common variants are CGNE (also called Craig's method[57]) and CGNR.[42,43,47,55] Unfortunately, both of these algorithms often converge slowly.[42,43,47] For example, we programmed CGNE and CGNR for TS-SCS analysis and tested them on the $C_{50}H_{24}$ polyacene, but convergence was not reached within 100 iterations for at least one direction at a single imfreq point. For these tests, we used the same convergence threshold as for the FCR algorithm; namely, $<10^{-5}$ for the maximum absolute value of each conditioned residual component.

3.2 The FCR algorithm

This provided motivation to develop a failproof conjugate residual (FCR) algorithm that converges robustly and resists round-off errors. FCR solves any linear equation system with Hermitian coefficients matrix

$$\mathbf{A}\mathbf{x} = \mathbf{B} \tag{70}$$

for an exact solution (within the convergence tolerance) if one exists or for a conditioned least-squares solution if no exact solution exists. The matrices **x** and **B** may contain a single column or more than one column. Matrix **A** is non-singular if and only if its determinant is nonzero (*i.e.*, all of its eigenvalues are non-zero). In this case, matrix **A** is invertible and the equation $\mathbf{A}\mathbf{x} = \mathbf{B}$ has the unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{B}$. If matrix **A** is singular, then $\mathbf{A}\mathbf{x} = \mathbf{B}$ has either no solution or an infinite number of solutions. If the linear equation system is consistent (*i.e.*, has at least one solution), this FCR algorithm returns one of its solutions. When matrix **B** has more than one column, the FCR algorithm is applied separately to each column. When $\mathbf{B} = \mathbf{I}$ (identity matrix), the method solves for $\mathbf{x} = \mathbf{A}^{-1}$, the inverse of matrix **A**.

The goal of conditioning is to rotate and scale matrix **A** to improve convergence speed:

$$\mathbf{M} = \mathbf{CAC}^{\mathrm{H}} \tag{71}$$

$$\mathbf{M}y = \mathbf{CAC}^{\mathrm{H}}y = \mathbf{C}(\mathbf{B} - \mathbf{Ax}_0) = \boldsymbol{W} \tag{72}$$

where

$$\mathbf{x} = \mathbf{C}^{\mathrm{H}}y + \mathbf{x}_0 \tag{73}$$

Here, $\mathbf{x}_0$ is any initial guess for $\mathbf{x}$. This shift always makes $y_0 = 0$ as the initial guess for y. The conditioning matrix $\mathbf{C}$ must be non-singular. Since $\mathbf{A}$ is Hermitian, the matrix $\mathbf{M}$ will automatically be Hermitian for any conditioning matrix $\mathbf{C}$:

$$\mathbf{M}^{\mathrm{H}} = (\mathbf{CAC}^{\mathrm{H}})^{\mathrm{H}} = (\mathbf{C}^{\mathrm{H}})^{\mathrm{H}}\mathbf{A}^{\mathrm{H}}\mathbf{C}^{\mathrm{H}} = \mathbf{CAC}^{\mathrm{H}} = \mathbf{M} \tag{74}$$

In this article, the dot product of two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ is defined as

$$\langle \boldsymbol{v}|\boldsymbol{w}\rangle = \boldsymbol{v}^{\mathrm{H}}\boldsymbol{w} = (\boldsymbol{v}^*)^{\mathrm{T}}\boldsymbol{w} = \sum_i \boldsymbol{v}_i^*\boldsymbol{w}_i \tag{75}$$

A Hermitian matrix $\mathbf{M} = \mathbf{M}^{\mathrm{H}}$ can be freely moved between sides of the dot product. For example,

$$\langle \mathbf{M}\mathbf{M}q|p\rangle = \langle \mathbf{M}q|\mathbf{M}p\rangle = \langle q|\mathbf{M}\mathbf{M}p\rangle = q^{\mathrm{H}}\mathbf{M}^2p \tag{76}$$

As commonly known, the Hermitian conjugate of a matrix product follows

$$(\Upsilon\Lambda)^{\mathrm{H}} = \Lambda^{\mathrm{H}}\Upsilon^{\mathrm{H}} \tag{77}$$

If the linear equation system (*i.e.*, eqn (70)) is inconsistent (*i.e.*, has no exact solution), this FCR algorithm returns a statement that the linear equation system has no exact solution along with a value $y = y^{\mathrm{FCR}}$ that minimizes the least-squares problem

$$\text{Minimize } F = \langle z|z\rangle = |\boldsymbol{W} - \mathbf{M}y|^2 \tag{78}$$

This represents a best possible choice for y irrespective of whether an exact solution to eqn (70) exists. There are some applications where this least-squares fit has utility even if an exact solution to eqn (70) does not exist. (Note: the case where $\mathbf{M}y = \boldsymbol{W}$ is consistent arises as the special case of eqn (78) where the least-squares error is simply zero.)

As commonly defined, the kernel of $\mathbf{M}$ is the set of all y values that solve $\mathbf{M}y = 0$,

$$\text{kernel}(\mathbf{M}) \Rightarrow \mathbf{M}y = 0 \tag{79}$$

When $\mathbf{M}$ is non-singular, $y = 0$ is the only vector in the kernel. When $\mathbf{M}$ is singular, the kernel includes $y = 0$ along with an infinite number of non-zero vectors. Manifestly, any vector from the kernel of $\mathbf{M}$ can be added to y^{FCR} without changing the conditioned residual value. Therefore, the set of all equally good (aka 'best') solutions to eqn (78) is

$$y^{\mathrm{all}} = y^{\mathrm{FCR}} + \text{span}[\text{kernel}(\mathbf{M})] \tag{80}$$

If the kernel($\mathbf{M}$) is known, this allows calculating the entire set of $\{y^{\mathrm{all}}\}$ that give the same minimum value of F. If $\mathbf{M}y = \boldsymbol{W}$ is consistent, this is the set of y values yielding $F = 0$ (*i.e.*, satisfying eqn (72)). The number of vectors in $\{y^{\mathrm{all}}\}$ is one if $\mathbf{M}$ is non-singular; otherwise, $\{y^{\mathrm{all}}\}$ contains an infinite number of vectors.

The FCR method minimizes the conditioned residual norm

$$\Gamma^{(i)} = \langle z^{(i)}|z^{(i)}\rangle \tag{81}$$

Each iteration involves two search directions, $p^{(i)}$ and $q^{(i)}$:

$$y^{(i)} = y^{(i-1)} + \gamma^{(i)}p^{(i)} + \tau^{(i)}q^{(i)} \tag{82}$$

$$z^{(i)} = z^{(i-1)} - \gamma^{(i)}\mathbf{M}p^{(i)} - \tau^{(i)}\mathbf{M}q^{(i)} \tag{83}$$

subject to the constraints

$$p^{(i)}, q^{(i)} \in K_{2i}(\mathbf{M}, \boldsymbol{W}) \tag{84}$$

$$K_{2i}(\mathbf{M}, \boldsymbol{W}) = \text{span}[p^{(1)}, q^{(1)}, \ldots p^{(i)}, q^{(i)}] \tag{85}$$

In exact arithmetic, all z-search directions are chosen to be orthogonal (aka 'conjugate'):

$$\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(j\neq i)}\rangle = \langle \mathbf{M}p^{(i)}|\mathbf{M}q^{(j)}\rangle = \langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(j\neq i)}\rangle = 0 \tag{86}$$

The conditioned residual component along z-search directions $\mathbf{M}p^{(i)}$ and $\mathbf{M}q^{(i)}$ are removed in iteration i:

$$\langle \mathbf{M}q^{(i)}|z^{(i)}\rangle = \langle \mathbf{M}p^{(i)}|z^{(i)}\rangle = 0 \tag{87}$$

This yields

$$\gamma^{(i)} = \frac{\langle \mathbf{M}p^{(i)}|z^{(i-1)}\rangle}{\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle} \tag{88}$$

$$\tau^{(i)} = \frac{\langle \mathbf{M}q^{(i)}|z^{(i-1)}\rangle}{\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i)}\rangle} \tag{89}$$

Substituting eqn (88), (89), and (83) into (81) yields (see ESI† for derivation):

$$\Gamma^{(i)} = \Gamma^{(i-1)} - |\gamma^{(i)}|^2\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle - |\tau^{(i)}|^2\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i)}\rangle \tag{90}$$

After computing the new values of $z^{(i)}$ and $\langle \mathbf{M}z^{(i)}|\mathbf{M}z^{(i)}\rangle$, the following convergence tests are performed: (1) if the absolute value of every conditioned residual component is less than a chosen CR_convergence_tol, the algorithm exits and returns a message that the linear equation system is consistent and returns the current value of y (which is a nearly exact solution). (2) Else if $\sqrt{\langle \mathbf{M}z^{(i)}|\mathbf{M}z^{(i)}\rangle}$ is less than a chosen $\mathbf{M}z$_length_tolerance, the algorithm exits and returns a statement that the linear equation system is inconsistent and returns the current value of y (which is the solution to the least-squares problem in eqn (78)). (3) Else if the number FCR iterations reaches max_CR_steps, the algorithm exits with a message the maximum number of FCR iterations was reached and returns the current value of y (which is the partial solution to the FCR problem). (4) Else the algorithm continues to the next iteration.

In some circumstances (A) an exit due to condition (3) is considered an error, but in other circumstances (B) it is the preferred exit condition. In this case (A), the goal is to reach a specified precision rather than a specified number of FCR iterations. In this case, max_FCR_iterations is set to a large value (*e.g.*, 100–1000) and exit *via* condition (3) represents a convergence failure. In case of convergence failure, the algorithm could be restarted using a different (and hopefully better) conditioning matrix. In case (B), the goal is to reach a specified number of FCR iterations rather than a specified precision. In classical atomistic simulations *via* polarizable force fields, using a fixed number of conjugate gradient iterations leads to continuous forces with enhanced energy conservation; therefore, exiting after a small constant number of iterations is preferred.[53,54] This behavior can be achieved by setting max_FCR_iterations to a small whole number (*e.g.*, ~5) and setting FCR_convergence_tol and **M***z*_length_tolerance to very small values.

The direction $p^{(i)}$ is chosen to ensure

$$\langle \mathbf{M}p^{(i)}|z^{(i-1)}\rangle = \langle \mathbf{M}z^{(i-1)}|\mathbf{M}z^{(i-1)}\rangle > 0 \tag{91}$$

when the linear equation system is consistent. This guarantees

$$0 \le I^{(i)} < I^{(i-1)} \tag{92}$$

even if round-off errors occurred during prior computations. Hence the method makes forward progress towards the solution in each and every iteration. As derived in the ESI,† these requirements are fulfilled by choosing

$$p^{(i)} = \mathbf{M}z^{(i-1)} - \beta^{(i)}p^{(i-1)} - \xi^{(i)}q^{(i-1)} \tag{93}$$

$$\beta^{(i)} = \frac{\langle \mathbf{M}^2p^{(i-1)}|\mathbf{M}z^{(i-1)}\rangle}{\langle \mathbf{M}p^{(i-1)}|\mathbf{M}p^{(i-1)}\rangle} \tag{94}$$

$$\xi^{(i)} = \frac{\langle \mathbf{M}^2q^{(i-1)}|\mathbf{M}z^{(i-1)}\rangle}{\langle \mathbf{M}q^{(i-1)}|\mathbf{M}q^{(i-1)}\rangle} \tag{95}$$

The length of $p^{(i)}$ is non-zero (see ESI† for derivation):

$$\langle p^{(i)}|p^{(i)}\rangle = \langle \mathbf{M}z^{(i-1)}|\mathbf{M}z^{(i-1)}\rangle + \langle \beta^{(i)}p^{(i-1)} + \xi^{(i)}q^{(i-1)}|\beta^{(i)}p^{(i-1)} + \xi^{(i)}q^{(i-1)}\rangle > 0 \tag{96}$$

As explained in the ESI,† the Krylov subspaces will be spanned with resistance to round-off errors by assigning $q^{(i)}$ *via* the following ordered sequence:

$$\kappa_1{}^{(i)} = -\frac{\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2q^{(i-1)}\rangle}{\sqrt{|\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2p^{(i-1)}\rangle|^2 + |\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2q^{(i-1)}\rangle|^2}} \tag{97}$$

$$\kappa_2{}^{(i)} = \frac{\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2p^{(i-1)}\rangle}{\sqrt{|\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2p^{(i-1)}\rangle|^2 + |\langle \mathbf{M}^2p^{(i)}|\mathbf{M}^2q^{(i-1)}\rangle|^2}} \tag{98}$$

$$q^{(i)} \leftarrow \kappa_1{}^{(i)}\mathbf{M}^2p^{(i-1)} + \kappa_2{}^{(i)}\mathbf{M}^2q^{(i-1)} \tag{99}$$

$$\chi_2{}^{(i)} = \frac{\langle \mathbf{M}^2p^{(i-2)}|q^{(i)}\rangle}{\langle \mathbf{M}p^{(i-2)}|\mathbf{M}p^{(i-2)}\rangle} \tag{100}$$

$$\sigma_2{}^{(i)} = \frac{\langle \mathbf{M}^2q^{(i-2)}|q^{(i)}\rangle}{\langle \mathbf{M}q^{(i-2)}|\mathbf{M}q^{(i-2)}\rangle} \tag{101}$$

$$q^{(i)} \leftarrow q^{(i)} - \chi_2{}^{(i)}p^{(i-2)} - \sigma_2{}^{(i)}q^{(i-2)} \tag{102}$$

$$\chi_1{}^{(i)} = \frac{\langle \mathbf{M}^2p^{(i-1)}|q^{(i)}\rangle}{\langle \mathbf{M}p^{(i-1)}|\mathbf{M}p^{(i-1)}\rangle} \tag{103}$$

$$\sigma_1{}^{(i)} = \frac{\langle \mathbf{M}^2q^{(i-1)}|q^{(i)}\rangle}{\langle \mathbf{M}q^{(i-1)}|\mathbf{M}q^{(i-1)}\rangle} \tag{104}$$

$$q^{(i)} \leftarrow q^{(i)} - \chi_1{}^{(i)}p^{(i-1)} - \sigma_1{}^{(i)}q^{(i-1)} \tag{105}$$

$$\chi_0{}^{(i)} = \frac{\langle \mathbf{M}^2p^{(i)}|q^{(i)}\rangle}{\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle} \tag{106}$$

$$q^{(i)} \leftarrow q^{(i)} - \chi_0{}^{(i)}p^{(i)} \tag{107}$$

Eqn (106) and (107) ensure that

$$\langle \mathbf{M}q^{(i)}|\mathbf{M}p^{(i)}\rangle = 0 \tag{108}$$

even if round-off errors occurred during prior computations. With this result, eqn (103)–(105) and (93)–(95) ensure that

$$\langle \mathbf{M}q^{(i)}|\mathbf{M}p^{(i-1)}\rangle = \langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i-1)}\rangle = 0 \tag{109}$$

$$\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i-1)}\rangle = \langle \mathbf{M}p^{(i)}|\mathbf{M}q^{(i-1)}\rangle = 0 \tag{110}$$

even if round-off errors occurred during prior computations. In exact arithmetic, the length of $q^{(i)}$ is non-zero in all iterations before the last one, and may either be zero or non-zero on the last iteration (see ESI† for derivation).

The algorithm is initialized with

$$p^{(i\le 0)} = q^{(i\le 0)} = 0 \tag{111}$$

$$p^{(1)} = \mathbf{M}z^{(0)} = \mathbf{M}W \tag{112}$$

$$q^{(1)} = z^{(0)} - \frac{\langle \mathbf{M}W|\mathbf{M}^2W\rangle}{\langle \mathbf{M}p^{(1)}|\mathbf{M}p^{(1)}\rangle}p^{(1)} \tag{113}$$

All denominators in the method have the form $\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle$ or $\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i)}\rangle$. Since **M** is Hermitian, its eigenvalues are real-valued. Since the eigenvalues of $\mathbf{M}^2$ are the squared eigenvalues of **M**, then all eigenvalues of $\mathbf{M}^2$ are non-negative. Thus, $\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle$ and $\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i)}\rangle$ are non-negative. As derived in Section S2 of the ESI,† $\langle \mathbf{M}p^{(i)}|\mathbf{M}p^{(i)}\rangle = 0$ can only occur if $\langle \mathbf{M}z^{(i-1)}|\mathbf{M}z^{(i-1)}\rangle = 0$ which would have already led to the algorithm exiting due to convergence in iteration $(i-1)$. $\langle \mathbf{M}q^{(i)}|\mathbf{M}q^{(i)}\rangle = 0$ may arise from (a) round-off errors or (b) on the last iteration if all components of $z^{(i)}$ not in kernel(**M**) are already made zero by the $\mathbf{M}p^{(i)}$ search direction. The algorithm includes division by zero protection at all steps. Specifically, the value of the parameter $\gamma^{(i)}$ (eqn (88)), $\tau^{(i)}$ (eqn (89)), $\beta^{(i)}$ (eqn (94)), $\xi^{(i)}$ (eqn (95)), $\kappa_1{}^{(i)}$ (eqn (97)), $\kappa_2{}^{(i)}$ (eqn (98)), $\chi_2{}^{(i)}$ (eqn (100)), $\sigma_2{}^{(i)}$ (eqn (101)), $\chi_1{}^{(i)}$ (eqn (103)), $\sigma_1{}^{(i)}$ (eqn (104)), or $\chi_0{}^{(i)}$ (eqn (106)) is set to zero if the

denominator appearing in the respective equation is zero. (As derived in the ESI,† this is the rigorously correct parameter value in the limit of zero denominator.)

In exact arithmetic, the algorithm converges to the exact solution in a finite number of iterations. The conditioned residual is eliminated along two directions in each iteration (*i.e.*, along $\mathbf{M}p^{(i)}$ and $\mathbf{M}q^{(i)}$). In exact arithmetic, these directions are orthogonal to all previous directions $\mathbf{M}p^{(j<i)}$ and $\mathbf{M}q^{(j<i)}$. The number of independent directions in the residual $z^{(i)}$ is less than or equal to *N*rows, where *N*rows is the number of rows in matrix $\mathbf{M}$. Therefore, in exact arithmetic the algorithm converges to the exact result in at most ceiling(*N*rows/2) iterations. In finite arithmetic, the FCR algorithm resists round-off errors as described above.

Further convergence analysis can be obtained by expanding $\boldsymbol{W}$ in terms of the eigenvectors $\{\vec{\boldsymbol{V}}_j\}$ of $\mathbf{M}$:

$$\boldsymbol{W} = \sum_j c_j \vec{\boldsymbol{V}}_j \tag{114}$$

The building blocks of the Krylov subspace are thus

$$\mathbf{M}^k \boldsymbol{W} = \sum_j c_j (\lambda_j)^k \vec{\boldsymbol{V}}_j \tag{115}$$

where $\{\lambda_j\}$ are the corresponding eigenvalues of $\mathbf{M}$. These building blocks can generate at most Ξ linearly independent vectors, where $\boldsymbol{W}$ is a linear combination of eigenvectors of $\mathbf{M}$ having exactly Ξ distinct eigenvalues λ_j. Since each FCR iteration searches 2 independent directions, in exact arithmetic the algorithm will converge in at most ceiling(Ξ/2) iterations. For example, if matrix $\mathbf{M}$ contains *N*rows = 1 million, but matrix $\mathbf{M}$ has only 12 distinct eigenvalues, then FCR will converge to the solution in at most 6 iterations in exact arithmetic.

FCR expands $y^{(i)}$ as

$$y^{(i)} = a_0 \boldsymbol{W} + a_1 \mathbf{M}\boldsymbol{W} + \ldots a_{2i-1}\mathbf{M}^{2i-1}\boldsymbol{W} = \sum_j c_j \mathrm{Poly}^{(i)}(\lambda_j)\vec{\boldsymbol{V}}_j \tag{116}$$

where $\{a_k\}$ are some optimized coefficients and

$$\mathrm{Poly}^{(i)}(\lambda) = \sum_{k=0}^{2i-1} a_k (\lambda)^k \tag{117}$$

is the associated polynomial. Expanding,

$$\mathbf{M}y^{(i)} = \sum_j c_j \lambda_j \mathrm{Poly}^{(i)}(\lambda_j)\vec{\boldsymbol{V}}_j = \sum_j c_j \vec{\boldsymbol{V}}_j \left(\sum_{k=0}^{2i-1} a_k (\lambda)^{k+1} \right) \tag{118}$$

The conditioned residual value at each FCR iteration is given by

$$\begin{aligned} z^{(i)} &= \boldsymbol{W} - \mathbf{M}y^{(i)} = \sum_j c_j V_j - \sum_j c_j \lambda_j \mathrm{Poly}^{(i)}(\lambda_j)\vec{\boldsymbol{V}}_j \\ &= \sum_j c_j V_j \left(1 - \lambda_j \mathrm{Poly}^{(i)}(\lambda_j)\right) \end{aligned} \tag{119}$$

Therefore, the exact solution is reached when

$$\mathrm{Poly}^{(i)}(\lambda_j) \Rightarrow 1/\lambda_j \tag{120}$$

The FCR convergence properties are thus dictated by the difficulty of representing $\{1/\lambda_j\}$ *via* a polynomial $\mathrm{Poly}^{(i)}(\lambda)$. When the eigenvalues are close to one, this is trivial because

$$\frac{1}{1 + \underbrace{(\lambda_j - 1)}_{\varepsilon_j}} = 1 - \varepsilon_j + {\varepsilon_j}^2 - {\varepsilon_j}^3 + {\varepsilon_j}^4 \ldots \tag{121}$$

When the eigenvalues are extremely spread out in values, then it takes a higher-order polynomial (and hence a larger number of FCR iterations) to make $\mathrm{Poly}^{(i)}(\lambda) \approx 1/\lambda$ for all of the present eigenvalues. Thus, the primary goal of conditioning (eqn (71)) is to make the eigenvalues less spread out in values.

All FCR equations presented here apply to both complex-valued and real-valued matrices. If $\mathbf{M}$ or $\boldsymbol{W}$ are complex-valued, then $\{p^{(i)}\}$, $\{q^{(i)}\}$, $\{z^{(i)}\}$, and $\gamma^{(i)}$, $\tau^{(i)}$, $\beta^{(i)}$, $\xi^{(i)}$, ${\kappa_{1,2}}^{(i)}$, ${\chi_{0,1,2}}^{(i)}$, ${\sigma_{1,2}}^{(i)}$ must be declared as complex variables. On the other hand, if $\mathbf{M}$ and $\boldsymbol{W}$ are real-valued, then all of those quantities should be declared as real variables. For solving the TS-SCS and polarizability equations, the corresponding matrices were real-valued.

For singular $\mathbf{M}$, $\mathbf{M}y$ does not contain any contributions from any zero eigenvalues, because multiplying by $\mathbf{M}$ multiplies the corresponding basis eigenvector by its eigenvalue (which is zero) as shown in eqn (118). Consequently, if $\boldsymbol{W}$ contains any nonzero contribution (*i.e.*, $c_j \neq 0$) from an eigenvector $\vec{\boldsymbol{V}}_j$ having associated zero eigenvalue (*i.e.*, $\lambda_j = 0$), then the linear equation system is inconsistent. Since only the basis eigenvectors appearing in $\boldsymbol{W}$ need to be represented by $\mathbf{M}y$ (see eqn (116)–(119)), the linear equation system is consistent and solved exactly by the FCR algorithm if and only if all nonzero contributions (*i.e.*, $c_j \neq 0$) of $\boldsymbol{W}$ have non-zero eigenvalues (*i.e.*, $\lambda_j \neq 0$). Therefore, even $\mathbf{M}y = \boldsymbol{W}$ systems for singular $\mathbf{M}$ can be solved exactly by FCR as long as the zero eigenvalues (*i.e.*, $\lambda_j = 0$) have zero contribution (*i.e.*, $c_j = 0$) to $\boldsymbol{W}$. In this case, the returned solution y^{FCR} has no contribution from any eigenvector whose associated eigenvalue is zero (*i.e.*, y^{FCR} is orthogonal to kernel($\mathbf{M}$)).

As shown in eqn (118), $\mathbf{M}y$ cannot contain any eigenvectors from the kernel($\mathbf{M}$), because each term contributing to $\mathbf{M}y$ contains the factor $(\lambda_j)^{k+1\geq 1}$ which is zero for any V_j in kernel($\mathbf{M}$). Consequently, any V_j in kernel($\mathbf{M}$) having non-zero c_j cannot be removed from the conditioned residual (see eqn (118) and (119)). Therefore, when $\mathbf{M}y = \boldsymbol{W}$ is inconsistent, minimizing $|\boldsymbol{W} - \mathbf{M}y|^2$ corresponds to removing all conditioned residual components that are not in kernel($\mathbf{M}$), while conditioned residual components in kernel($\mathbf{M}$) remain at their initial levels (*i.e.*, same contribution as in $\boldsymbol{W}$).

The ESI† explains further details. Section S2.1† defines the problem definition and matrix conditioning. Section S2.2† contains eigenvalue decomposition analysis using Krylov subspaces. Section S2.3† defines conjugate search directions that span the Krylov subspaces. Section S2.4† details vector lengths in exact arithmetic. Section S2.5† explains robust convergence and round-off error resistance. Section S2.6† presents a step-by-step computational procedure of this FCR algorithm.

3.3 Using FCR to solve the TS-SCS equations

The following procedure was used to solve the TS-SCS equations using the FCR algorithm. First, the following quantities were defined

$$\boldsymbol{W} = \sqrt{\alpha_A^{\text{unscreened}}(u)}\vec{\boldsymbol{E}}_A^0 \tag{122}$$

$$\mathbf{M} = \left(\delta_{st} - \sqrt{\alpha_A^{\text{unscreened}}(u)\alpha_B^{\text{unscreened}}(u)}\,\tau_{st}{}^{AB}(u)\right) \tag{123}$$

for one value of u with $\vec{\boldsymbol{E}}_A^0$ set to (1, 0, 0). The FCR algorithm was then used to solve the linear equation $\mathbf{M}y = \boldsymbol{W}$ for y which was solved for $\vec{\boldsymbol{\mu}}_{\text{B}}^{\text{induced}}$ by multiplying by $\sqrt{\alpha_B^{\text{unscreened}}(u)}$:

$$y = \frac{\vec{\boldsymbol{\mu}}_B^{\text{induced}}}{\sqrt{\alpha_B^{\text{unscreened}}(u)}} \tag{124}$$

$\vec{\boldsymbol{\mu}}_{\text{B}}^{\text{induced}}$ gives the first column of $\vec{\vec{\boldsymbol{\alpha}}}_B^{\text{SCS}}(u)$. In this scheme, $\vec{\boldsymbol{\mu}}_{\text{B}}^{\text{induced}}$ corresponds to '**x**' in the linear equation system '$\mathbf{Ax} = \mathbf{B}$', and **B** corresponds to the externally applied electric field, $\{\vec{\boldsymbol{E}}_A^0\}$. The conditioning matrix **C** is diagonal with $\{\sqrt{\alpha_A^{\text{unscreened}}(u)}\}$ along the diagonal. Solving again with $\vec{\boldsymbol{E}}_A^0$ set to (0, 1, 0) yields $\vec{\boldsymbol{\mu}}_{\text{B}}^{\text{induced}}$ that gives the second column of $\vec{\vec{\boldsymbol{\alpha}}}_B^{\text{SCS}}(u)$. Finally, solving with $\vec{\boldsymbol{E}}_A^0$ set to (0, 0, 1) yields $\vec{\boldsymbol{\mu}}_{\text{B}}^{\text{induced}}$ that gives the third column of $\vec{\vec{\boldsymbol{\alpha}}}_B^{\text{SCS}}(u)$. This entire process is repeated at each imfreq point u.

We now discuss a few computational examples. For the examples we studied, the FCR algorithm indicated the linear equations were consistent and converged to a nearly exact solution. Table 5 shows the FCR algorithm converged to the same solution in the same number of iterations for a large supercell as for the small primitive unit cell of the same material. Table 5 also shows FCR converged to the same solution (within the convergence tolerance) as direct matrix inversion using GEPP. Since the number of diverse atom types in graphene and ice is not large, we explored three more diverse chemical structures shown in Fig. 6. These included a large biomolecule (B-DNA) whose geometry was taken from a prior study,[14] the $C_{50}H_{24}$ polyacene geometry taken from the companion article,[1] and a metal–organic framework whose geometry was taken from the Computation Ready Experimental (CoRE) metal–organic framework (MOF) database.[58] We used the MOF having Cambridge Structural Database[59] code KUCDIW. Electron densities for these three materials were generated in VASP using the PBE functional, a 400 (B-DNA and MOF) and 750 ($C_{50}H_{24}$) eV planewave cutoff, and the PAW method. The k-point mesh and grid spacing followed previous recommendations.[12] FCR convergence for these three materials is summarized in Table 6 and Fig. 7. As shown in Fig. 7, the conditioned residual norm decreased rapidly and monotonically with increasing iteration number for all three materials. Table 6 shows that a summed total of up to a couple thousand large matrix-vector multiplies may be required to complete TS-SCS(FCR) analysis across all imfreq points. Convergence was highly efficient, with FCR convergence along one direction for a single imfreq point occurring in $\leq$31 iterations. Each FCR iteration required four large matrix-vector multiplies.

Except where otherwise specified, all computations in this article and the companion article[1] were performed using 64-bit real numbers. For comparison, computations performed using 128-bit reals are also displayed in Table 6 and Fig. 7. Using 128-bit reals increases computational cost and decreases round-off errors compared to 64-bit reals. Calculations converged in fewer FCR iterations using 128-bit reals, but the overall computational time was higher. The converged results were equivalent within the convergence tolerance.

We used OpenMP to parallelize the most computationally intense parts of this TS-SCS algorithm. The loops parallelized included calculation of unscreened and screened total C_6 for the unit cell (analogous to the loops parallelized in Fig. 1 and 5) and all of the large matrix-vector multiplies in the FCR algorithm. Fig. 8 shows an excerpt of the OpenMP enabled code for the large matrix-vector multiplies to compute $\mathbf{MM}p(:,1) = \mathbf{M} \times \mathbf{M}p(:)$ and $\mathbf{MM}q(:,1) = \mathbf{M} \times \mathbf{M}q(:)$.

The TS-SCS(FCR) code achieves linear-scaling computational cost for sufficiently large Natoms when the number of FCR iterations does not increase appreciably with increasing system size. To date, we have not observed any materials for which TS-SCS(FCR) converges slowly. However, the fixed number of required iterations for MCLF is a clear advantage compared to the variable number of iterations to converge TS-SCS. First, it makes MCLF convergence highly repeatable, which will become important for applications that involve differentiation with respect to atomic positions (*e.g.*, computing forces). Second, it makes MCLF computational times highly predictable, because the number of required iterations is known up front.

4. Performance results

4.1 Required computational time and memory

In addition to the linear-scaling MCLF and TS-SCS algorithms described in Sections 2 and 3 above, MCLF and TS-SCS were also programmed using direct matrix inversions *via* Gaussian elimination with partial pivoting (GEPP). GEPP is a widespread algorithm described in many numerical methods textbooks.[60] This allowed us to compare both the computational time and precision (see Tables 4 and 5) of the inverse-free algorithms to the direct inverse algorithms.

Both the required computational time and memory of the inverse-free algorithms are proportional to the number of atoms in the unit cell times the number of separately cataloged pairwise interactions per atom. *Case 1*: for an isolated molecule much smaller than the dipole interaction cutoff length, increasing the number of atoms in the molecule also increases the number of pairwise interactions per atom. In this case, the required computational time and memory scale proportional to the number of atoms squared. *Case 2*: quadratic scaling of computational time and memory is also observed for periodic materials having small unit cells. As the number of atoms in the unit cell increases, the number of separately cataloged pairwise interactions per atom also increases. *Case 3*: when the unit cell is large enough to completely enclose a sphere of dipole interaction cutoff

length radius, the number of separately cataloged pairwise interactions per atom saturates. Making the unit cell even larger does not increase the number of separately cataloged pairwise interactions per atom. In this case, both the required computational time and memory scale linearly with increasing system size.

Fig. 9 plots required computational time and RAM to perform MCLF analysis on ice crystals containing different numbers of atoms in the periodic unit cell. These calculations described the same hexagonal ice crystal structure, but with different sized unit cells. MCLF results from these different sized unit cells are numerically equivalent. Electron densities for the unit cells containing 12 to 8748 atoms were taken from ref. 18. Herein, we also constructed periodic unit cells containing 20 736 to 263 424 atoms from the computed DDEC6 AIM properties of the smaller unit cells; no DFT calculations on these large supercells were needed. As shown in Fig. 9, the required computational time and memory for MCLF analysis scaled linearly with increasing number of atoms in the unit cell when the unit cell was large enough to enclose a sphere of dipole interaction cutoff length radius. The TS-SCS method using direct matrix inversion (GEPP algorithm) is plotted for comparison, because the prior literature used a similar approach.[2] As shown in Fig. 9, MCLF is less computationally expensive than TS-SCS using GEPP. While TS-SCS(GEPP) calculations larger than 4116 atoms per unit cell did not complete in one week on a single processor, an MCLF calculation with 263 424 atoms in the unit cell completed in 4.1 days on a single processor.

Fig. 10 compares the FCR and GEPP algorithms for performing TS-SCS analysis on these ice supercells. Direct matrix inversion using GEPP had nearly cubic scaling computational time with increasing number of atoms in the unit cell, which made it infeasible for unit cells containing >4116 atoms. Similar to MCLF analysis, the required computational time and memory for the TS-SCS(FCR) algorithm scaled linearly with increasing number of atoms in the unit cell when the unit cell was large enough to enclose a sphere of dipole interaction cutoff length radius. The TS-SCS(FCR) calculation with 263 424 atoms in the unit cell completed in 0.97 days on a single processor. TS-SCS(FCR) requires only about 50–60% of the memory as MCLF.

The computational cost of the GEPP algorithm for direct matrix inversion could be minimized by using a highly optimized linear algebra package such as LAPACK. This would result in faster computational times for GEPP than we reported here. However, this optimization would only lower the prefactor and not the exponent in the required computational time scaling relation. Even for highly optimized code, the cubic-scaling computational time of GEPP makes it impractical for materials containing a large number of atoms in the unit cell.

Tables 7 and 8 list the calculation time breakdowns for MCLF and TS-SCS methods using the inverse-free algorithms. The computational times were consistent between runs, as demonstrated by the small standard deviations (<5%). For MCLF, the most time consuming section is the fluctuating screening, followed by static polarizability screening, and atom image pair matrix initialization. For TS-SCS, the most time consuming section is TS-SCS screening, followed by atom image pair matrix initialization. For both methods, the other sections take only negligible time. Fluctuations in file reading times were presumably due to variations in how quickly the files could be accessed, which depended on current file system load across users. The drop in computational time for unscreened α & C_6 computation from 111 132 to 263 424 atoms is due to switching to the wp lookup table method for computing C_6 with 263 424 atoms.

4.2 Parallelization efficiency

The parallelization efficiency is defined as (time for serial calculation)/((time for parallel calculation) $\times$ (number of parallel computing cores)). (Serial and parallel computational times were the average of three runs.) Tables 9 and 10 list the calculated efficiency of MCLF and TS-SCS methods for selected ice crystals. The parallelization efficiencies were excellent. Some jobs had efficiencies greater than 100%, because the parallel program uses OpenMP while the serial program does not. When enabled, OpenMP can speed up the calculation even if only one processor is used.

Table 11 compares the RAM requirements for serial execution to 8 parallel cores. Results are listed for both MCLF and TS-SCS(FCR) algorithms. Our results show that running the program in serial and parallel modes requires about the same amount of memory regardless of the number of cores used. In other words, adding parallel cores does not significantly increase the program's memory requirements.

To quantify the performance of these algorithms for larger systems, an ice supercell containing >2 million atoms in the unit cell was prepared from the DDEC6 AIM properties. Specifically, a $2 \times 2 \times 2$ supercell of the 263 424 atom unit cell gave a periodic supercell containing 2 107 392 atoms. Due to the large size, serial (*i.e.*, one processor) TS-SCS(FCR) and MCLF calculations could not complete in less than one week; therefore, only parallel calculations were run. Secondly, the source code had to be compiled using 64-bit integers to accommodate the large range of array index values. This 64-bit integer source code and a large memory node were also used for the 1 053 696 atom system reported in Table 3 above. (All other calculations reported in this paper used 32-bit integers, because this was the compiler's default.) Thirdly, the program had to run on a large memory node, because a normal node does not contain enough RAM to complete the calculation. The processor speed for the large memory node on the Comet cluster was 2.2 GHz compared to 2.5 GHz for its normal nodes. Therefore, one must exercise caution when comparing timing results for this large supercell to results reported above for the other ice supercells that ran on the normal nodes. The MCLF calculation for this material took 21.7 hours (average of 3 runs with standard deviation of 5%) on 48 parallel computing cores with 950 GB RAM. The TS-SCS(FCR) calculation took 5.6 hours (average of 3 runs with standard deviation of 2%) on 48 parallel computing cores with 500 GB RAM.

5. Conclusions

We developed computationally efficient algorithms to compute atom-in-material polarizabilities and dispersion coefficients using MCLF and TS-SCS analysis. Our MCLF algorithm uses

Richardson extrapolation of the screening increments. Our TS-SCS algorithm uses a special conjugate residual algorithm that resists round-off errors. Both our algorithms have computational time and memory requirements scaling linearly with the number of atoms in the unit cell when the unit cell is much larger than the dipole interaction cutoff distance. For both small and large systems, our algorithms require less computational time and memory than direct matrix inversion, with negligible change in computational precision. Our algorithms achieved this by avoiding both large matrix inversions and large dense matrix multiplies. This is an important achievement, because direct matrix inversion and dense square matrix multiplication have computational costs scaling between $N\text{rows}^2$ and $N\text{rows}^3$.[11] Other important algorithms to achieve linear scaling included: (a) classification of atoms into spatial regions followed by constructing two lists of interacting atom pairs and (b) a lookup table method to compute C_6^{total}.

Our algorithms were easily parallelized to take advantage of multiple computing cores. To minimize false sharing, our algorithms access data in cache line friendly order. Excellent parallelization efficiencies were obtained. Moreover, adding parallel computing cores did not significantly increase memory requirements.

This made it possible to apply the MCLF and TS-SCS methods to materials containing orders of magnitude more atoms per unit cell than previously feasible. Our algorithms can be readily applied to materials containing millions of atoms in the unit cell. The largest example studied herein was an ice crystal containing >2 million atoms in the unit cell. To perform TS-SCS on this material, the FCR algorithm solved a linear equation system containing >6 million rows, 7.57 billion interacting atom pairs in the large list and 87 million in the small list, 45.4 billion stored non-negligible matrix components used in each large matrix-vector multiplication, and ~19 million unknowns per frequency point (>300 million total unknowns). This problem was solved in 5.6 hours by 48 parallel computing cores with 500 GB RAM. The MCLF calculation for this material took 21.7 hours on 48 parallel computing cores with 950 GB RAM.

The MCLF and TS-SCS software programs described here will be distributed through the same code repository as the Chargemol program. Required inputs for the TS-SCS program are: (i) an *xyz* file containing the list of atoms (as element symbols and x, y, z coordinates in Å), unit cell information (*i.e.*, lattice vectors if system is periodic), and AIM $\langle r^3 \rangle$ moments and (ii) a calculation_parameters.txt file listing calculation parameter values. Required inputs for the MCLF program are: (i) separate *xyz* files for AIM volumes, $\langle r^3 \rangle$ moments, $\langle r^4 \rangle$ moments, weighted $\langle r^4 \rangle$ moments, and net atomic charges, and (ii) a calculation_parameters.txt file listing calculation parameter values. Settings in the calculation_parameters.txt file choose whether to use an inverse-free method or direct matrix inversion, the Romberg integration order, the dipole interaction cutoff length, whether to ignore PBC, the number of Richardson extrapolation steps (for MCLF), the convergence threshold (for TS-SCS(FCR)), *etc.* The default values are reliable and almost never need to be changed.

Finally, the new failsafe conjugate residual algorithm should find widespread applications to a plethora of scientific computing problems, because it resists round-off errors and solves any linear equation system with Hermitian coefficients matrix. This conjugate residual algorithm has many desirable mathematical properties, because it minimizes the norm of the conditioned residual within a Krylov subspace of increasing order with each successive iteration.

Authors' contributions

T. A. M. supervised the study, obtained funding, developed all computational and mathematical methods, generated and wrote all mathematical derivations and proofs (including entire FCR algorithm, wp lookup table method, atom image pair array initialization, Richardson extrapolation, and Romberg integration), wrote the entire ESI,† designed all computational tests, and wrote the MCLF and TS-SCS software codes. Both authors performed calculations, data analysis, and prepared tables and figures for the main text. The manuscript was written mainly by T. A. M. with minor contribution from T. C.

Conflicts of interest

There are no conflicts of interest to declare.

Acknowledgements

This project was funded by National Science Foundation (NSF) CAREER Award DMR-1555376. Supercomputing resources were provided by the Extreme Science and Engineering Discovery Environment (XSEDE).[61] XSEDE is funded by NSF grant ACI-1548562. XSEDE project grant TG-CTS100027 provided allocations on the Comet cluster at the San Diego Supercomputing Center (SDSC). All timing and memory tests were run on the Comet cluster using Intel Xeon Haswell processors. (The normal nodes had Xeon E5-2680v3 processors.) The authors sincerely thank the technical support staff of XSEDE and SDSC.

References

1 T. A. Manz, T. Chen, D. J. Cole, N. Gabaldon Limas and B. Fiszbein, New scaling relations to compute atom-in-material polarizabilities and dispersion coefficients: part 1. Theory and accuracy, *RSC Adv.*, 2019, **9**, 19297–19324.
2 A. Tkatchenko, R. A. DiStasio, R. Car and M. Scheffler, *Phys. Rev. Lett.*, 2012, **108**, 236402.
3 A. Ambrosetti, A. M. Reilly, R. A. DiStasio and A. Tkatchenko, *J. Chem. Phys.*, 2014, **140**, 18A508.
4 A. Tkatchenko and M. Scheffler, *Phys. Rev. Lett.*, 2009, **102**, 073005.
5 T. Bucko, S. Lebegue, J. Hafner and J. G. Angyan, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2013, **87**, 064110.
6 T. Bucko, S. Lebegue, J. G. Angyan and J. Hafner, *J. Chem. Phys.*, 2014, **141**, 034114.

7 T. Bucko, S. Lebegue, J. Hafner and J. G. Angyan, *J. Chem. Theory Comput.*, 2013, **9**, 4293–4299.
8 T. Gould, S. Lebegue, J. G. Angyan and T. Bucko, *J. Chem. Theory Comput.*, 2016, **12**, 5920–5930.
9 T. Gould and T. Bucko, *J. Chem. Theory Comput.*, 2016, **12**, 3603–3613.
10 R. E. Watson, *Phys. Rev.*, 1958, **111**, 1108–1110.
11 V. Strassen, *Numerische Mathematik*, 1969, **13**, 354–356.
12 N. Gabaldon Limas and T. A. Manz, *RSC Adv.*, 2018, **8**, 2678–2707.
13 T. A. Manz and N. Gabaldon Limas, *RSC Adv.*, 2016, **6**, 47771–47801.
14 N. Gabaldon Limas and T. A. Manz, *RSC Adv.*, 2016, **6**, 45727–45747.
15 T. A. Manz and D. S. Sholl, *J. Chem. Theory Comput.*, 2012, **8**, 2844–2867.
16 T. A. Manz and D. S. Sholl, *J. Chem. Theory Comput.*, 2011, **7**, 4146–4164.
17 T. A. Manz and D. S. Sholl, *J. Chem. Theory Comput.*, 2010, **6**, 2455–2468.
18 T. A. Manz, *RSC Adv.*, 2017, **7**, 45552–45581.
19 G. Kresse and J. Furthmuller, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1996, **54**, 11169–11186.
20 G. Kresse and J. Furthmuller, *Comput. Mater. Sci.*, 1996, **6**, 15–50.
21 G. Kresse and J. Hafner, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1993, **47**, 558–561.
22 J. P. Perdew, K. Burke and M. Ernzerhof, *Phys. Rev. Lett.*, 1996, **77**, 3865–3868.
23 G. Kresse and D. Joubert, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1999, **59**, 1758–1775.
24 P. E. Blochl, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1994, **50**, 17953–17979.
25 H. B. G. Casimir and D. Polder, *Phys. Rev.*, 1948, **73**, 360–372.
26 N. A. de Lima, *J. Chem. Phys.*, 2010, **132**, 014110.
27 K. T. Tang, *Phys. Rev.*, 1969, **177**, 108–114.
28 E. B. Anders, *J. Assoc. Comput. Mach.*, 1966, **13**, 505–510.
29 J. Dutka, *Hist. Math.*, 1984, **11**, 3–21.
30 D. C. Joyce, *SIAM Rev.*, 1971, **13**, 435–488.
31 A. P. Jones, J. Crain, V. P. Sokhan, T. W. Whitfield and G. J. Martyna, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2013, **87**, 144103.
32 L. N. Trefethen and R. S. Schreiber, *SIAM J. Matrix Anal. Appl.*, 1990, **11**, 335–360.
33 L. N. Trefethen, *ACM SIGNUM Newsletter*, 1985, **20**, 2–5.
34 J. Applequist, J. R. Carl and K. K. Fung, *J. Am. Chem. Soc.*, 1972, **94**, 2952–2960.
35 G. Schulz, *Z. Angew. Math. Mech.*, 1933, **13**, 57–59.
36 E. W. Weisstein, *Matrix Norm, MathWorld – A Wolfram Web Resource*, http://mathworld.wolfram.com/MatrixNorm.html, accessed August 2018.
37 A. Sivri, *Practical Extrapolation Methods: Theory and Applications*, Cambridge University Press, Cambridge, UK, 1 edn, 2003, pp. 19–41.
38 B. Chapman, G. Jost and A. R. van der Pas, *Using OpenMP*, The MIT Press, Cambridge, Massachusetts, 2008.
39 M. Hermanns, *Parallel Programming in Fortran 95 using OpenMP*, Madrid, Spain, 2002, pp. 1–71.
40 T. Bucko, S. Lebegue, T. Gould and J. G. Angyan, *J. Phys.: Condens. Matter*, 2016, **28**, 045201.
41 J. L. Gustafson, The Quest for Linear Equation Solvers and the Invention of Electronic Digital Computing, *IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA'06)*, IEEE Computer Society, 2006, pp. 1–7, DOI: 10.1109/JVA.2006.50.
42 Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edn, 2003.
43 R. W. Freund, G. H. Golub and N. M. Nachtigal, *Acta Numerica*, 1992, **1**, 1–44.
44 C. Lanczos, *J. Res. Natl. Bur. Stand.*, 1952, **49**, 33–53.
45 C. C. Paige and M. A. Saunders, *ACM Trans. Math. Software*, 1982, **8**, 195–209.
46 H. C. Elman, in *Large-Scale Matrix Problems and the Numerical Solution of Partial Differential Equations*, ed. J. Gilbert and D. Kershaw, Oxford University Press, Oxford, UK, 1994, pp. 69–118.
47 S. F. Ashby, T. A. Manteuffel and P. E. Saylor, *SIAM J. Numer. Anal.*, 1990, **27**, 1542–1568.
48 V. Simoncini and D. B. Szyld, *Numer. Lin. Algebra Appl.*, 2007, **14**, 1–59.
49 D. M. Young and K. C. Jea, *Lin. Algebra Appl.*, 1980, **34**, 159–194.
50 W. D. Joubert and D. M. Young, *Lin. Algebra Appl.*, 1987, **88–9**, 449–485.
51 H. A. Vandervorst and K. Dekker, *J. Comput. Appl. Math.*, 1988, **24**, 73–87.
52 C. S. Liu, H. K. Hong and S. N. Atluri, *Comput. Model. Eng. Sci.*, 2010, **60**, 279–308.
53 F. Aviat, L. Lagardere and J. P. Piquemal, *J. Chem. Phys.*, 2017, **147**, 161724.
54 F. Aviat, A. Levitt, B. Stamm, Y. Maday, P. Y. Ren, J. W. Ponder, L. Lagardere and J. P. Piquemal, *J. Chem. Theory Comput.*, 2017, **13**, 180–190.
55 M. R. Hestenes and E. Stiefel, *J. Res. Natl. Bur. Stand.*, 1952, **49**, 409–436.
56 E. W. Weisstein, *Positive Definite Matrix, MathWorld – A Wolfram Web Resource*, http://mathworld.wolfram.com/PositiveDefiniteMatrix.html, accessed February, 2019.
57 E. J. Craig, *J. Math. Phys.*, 1955, **34**, 64–73.
58 Y. G. Chung, J. Camp, M. Haranczyk, B. J. Sikora, W. Bury, V. Krungleviciute, T. Yildirim, O. K. Farha, D. S. Sholl and R. Q. Snurr, *Chem. Mater.*, 2014, **26**, 6185–6192.
59 F. H. Allen, *Acta Crystallogr., Sect. B: Struct. Sci.*, 2002, **58**, 380–388.
60 K. J. Beers, *Numerical Methods for Chemical Engineering*, Cambridge University Press, Cambridge, UK, 2007, pp. 10–23.
61 J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott and N. Wilkins-Diehr, *Comput. Sci. Eng.*, 2014, **16**, 62–74.

62 C.-K. Skylaris, P. D. Haynes, A. A. Mostofi and M. C. Payne, *J. Chem. Phys.*, 2005, **122**, 084119.
63 D. R. Bowler and T. Miyazaki, *Rep. Prog. Phys.*, 2012, **75**, 036503.
64 D. R. Bowler, T. Miyazaki and M. J. Gillan, *J. Phys.: Condens. Matter*, 2002, **14**, 2781–2798.
65 S. Goedecker, *Rev. Mod. Phys.*, 1999, **71**, 1085–1123.
66 P. Ordejon, D. A. Drabold, R. M. Martin and M. P. Grumbach, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1995, **51**, 1456–1476.
67 D. J. Cole and N. D. M. Hine, *J. Phys.: Condens. Matter*, 2016, **28**, 393001.
68 K. A. Wilkinson, N. D. M. Hine and C.-K. Skylaris, *J. Chem. Theory Comput.*, 2014, **10**, 4782–4794.
69 L. Hung and E. A. Carter, *Chem. Phys. Lett.*, 2009, **475**, 163–170.