Title:

Scalable anisotropic vibrations of megascale macromolecules

Author List

Jordy Homing Lam[1,5,6], Aiichiro Nakano*[1,3,4], Vsevolod Katritch*[1,2,5,6]

Affiliations

[1] Department of Quantitative and Computational Biology, University of Southern California, Los Angeles, CA, USA.

[2] Department of Chemistry, University of Southern California, Los Angeles, CA, USA.

[3] Department of Physics and Astronomy, University of Southern California, Los Angeles, CA, USA.

[4] Department of Computer Science, University of Southern California, Los Angeles, CA, USA.

[5] Bridge Institute and Michelson Center for Convergent Biosciences, University of Southern California, Los Angeles, CA, USA.

[6] Center for New Technologies in Drug Discovery and Development, University of Southern California, Los Angeles, CA, USA.

*To whom correspondence shall be addressed: anakano@usc.edu, katritch@usc.edu

Abstract

The Normal Mode Analysis (NMA) is a standard approach to elucidate the anisotropic vibrations of macromolecules at their folded states, where low-frequency collective motions can reveal rearrangements

of domains and changes in the exposed surface of macromolecules. Recent advances in structural biology have enabled the resolution of megascale macromolecules with millions of atoms. However, the calculation of their vibrational modes remains elusive due to the prohibitive cost associated with constructing and diagonalizing the underlying eigenproblem and the current approaches to NMA are not readily adaptable for efficient parallel computing on graphic processing unit (GPU). Here, we present eigenproblem construction and diagonalization approach that implements level-structure bandwidth-reducing algorithms to transform the sparse computation in NMA to a globally-sparse-yet-locally-dense computation, allowing batched tensor products to be most efficiently executed on GPU. We mapped, optimized, and compared several low-complexity Krylov-subspace eigensolvers, supplemented by techniques such as Chebyshev filtering, sum decomposition, external explicit deflation and shift-and-inverse, to allow fast GPU-resident calculations. The method allows accurate calculation of the first 1000 vibrational modes of some largest structures in PDB (> 2.4 million atoms) at least 250 times faster than existing methods.

Introduction

The Normal Mode Analysis (NMA) is a standard approach to derive motions from static snapshots of a macromolecular structure. As a computational probe to shape-changing motions, the analysis has found wide applicability in the refinement and fitting of macromolecular structures[1,2] and in the simulations of functional motions when coupled with enhanced sampling techniques[3,4]. Many of these predicted modes of motion align consistently with available experimental data in kinases[5], ion channels[6] and transporters[7]. Such successful applications have led to an increasing appreciation of the interdependence among biological structure, dynamics, and function[8]. In NMA, the input macromolecular structure is assumed to be in a conformational minimum of its potential energy. Spatial arrangements of atoms (or atom groups like residues) and forces among them are then assimilated into the Hessian matrix and the equation of motion is analyzed under the harmonic approximation. The outcome of NMA are eigenvectors, representing sets of optimal displacements for each atom in the system, ranked by their ascending eigenvalues that reflect

increasing strain along those directions. Early attempts of NMA on macromolecules[9–12] were derived from all-atom potentials and full dense Hessian matrices. This approach to NMA is still the golden standard in analyzing the shear and hinge motions of proteins with sizes typically less than 5000 atoms.[13] However, applying NMA in this form to larger macromolecules is difficult as their energy minimizations are prone to overstepping, rendering unphysical modes. [14,15] In a pioneering work by Tirion[15], this requirement was relaxed by replacing the detailed all-atom potentials with an elastic network of pairwise Hookean potential, connecting atoms within a neighborhood boundary. The NMA analysis then proceeds by assuming that the provided structure was minimized under experimental conditions. As such, crystal coordinates, which lack hydrogens and/or sidechains, can be analyzed directly in absence of a forcefield. This practical form of NMA is now commonly known as the elastic network model (ENM). Despite its simplicity, as demonstrated convincingly by Hinsen[16,17] and Sanejouand[18], the low-frequency ENM modes were shown to agree with the fluctuations observed in X-ray crystallography as well as with NMAs performed under standard all-atom potentials, as long as only low-frequency modes are concerned. The ENM also found major applications in Cryogenic Electron Microscopy (Cryo-EM) and Cryogenic Electron Tomography (Cryo-ET) for the flexible refinement of the composite density maps[19–21] and the flexible fitting of atomic models[22–24] when EM volume or atomic templates are available. These applications of NMA are especially compelling for large macromolecular complexes, when other methods are computationally expensive. However, computing the NMA, even in this practical ENM form, remains challenging for large macromolecular systems of more than 1,000,000 particles. Namely, the storage, the construction and the diagonalization of the Hessian matrix all create imminent difficulties in terms of time and memory complexities as the size of the matrix increases quadratically with the number of atoms N. While ideally the memory complexity of NMA only increases linearly, the linear factors due to packing density of atoms as well as the cubic time complexity to diagonalize the Hessian matrix present a major bottleneck in calculations (See Methods). As such, many innovative approaches, including Anisotropic Network Model (ANM) [15,25], Rotation-Translation Block (RTB) method [26] and Block Normal Mode (BNM) [27], were undertaken to eliminate the degrees of freedom (DOF) to be differentiated in these larger systems, hence

reducing N for 8-10 times. Agreement of these methods with experimental data[18,25] are satisfactory, though the displacement information of the discounted atoms were lost and the appropriate level of granularity, especially in presence of elongated (e.g., lipids) or planar (e.g., aromatics) chemical moieties, are hard to be determined. To apply NMA without excessive coarse-graining, the development of faster numerical recipes is necessary[28,29]. More recently, the choices on diagonalization algorithms were examined on a hyperthreaded machine in the work of Koehl[30]. This ultimately allowed the first 100 ENM modes of a ZIKV virus (PDBID: 5IZ7), with around 800 thousand atoms and around 300 million nonzero entries in its Hessian, to be calculated within an hour when the Jacobi-Davidson Method (JDM)[31,32] were filtered with an 80-degree low-pass Chebyshev polynomial of the Hessian. [30] However, the current state-of-the-art is still far from handling megascale systems with more than a million atoms and billions of non-zero entries in the Hessian.

In this work, we designed several synergistic algorithms to fully engage dense graphic processing unit (GPU) kernels in both construction and diagonalization of the Hessian in atomic NMAs. Specifically, we developed a level-structure algorithm to minimize the bandwidth of the Hessian, prior to its construction, and thereby converting its sparse computation into a globally-sparse-yet-locally-dense computation, which enables the batched execution of tensor products on GPUs. We also mapped, optimized and compared several low-complexity Krylov-subspace eigensolvers, supplemented by techniques such as Chebyshev filtering[33–35], sum decomposition, external explicit deflation[36] and shift-and-inverse, to allow fast calculations on a GPU device. The level-structure algorithm produces an isomorph of the original elastic network, for which its unpermuted eigenpairs can be recovered in linear complexity using the bijection mapping. The implementation of this INCHING ("Isomorphic Nma Calculations Harnessing 1 Necessary Gpu") algorithm presented here was benchmarked on macromolecules from the Protein Data Bank[37] with sizes up to 2.4 million atoms. Compared to other existing methods, 250-370 times speedup in throughput was achieved while maintaining residual error under 10^{-12}. The utility of the INCHING method was also demonstrated through examples, including the largest experimentally resolved atomistic structure of a mature HIV-1 capsid[38] (PDBID:3J3Q) with 2.4 million atoms and 1.6 billion non-zero entries in its Hessian.

Using our INCHING program, we were able to resolve its first 64 atomic normal modes within 44 minutes of wall-clock time on a single NVIDIA® A100 Tensor Core GPU and its first 1000 atomic normal modes within 63 hours. Fast, accurate, and highly scalable GPU-optimized implementation of NMA approach will find practical applications in conformational analysis of large and dynamic macromolecular complexes, and facilitate their refinement from cryo-EM/cryo-ET data.

Results

Overview of the INCHING Algorithm

The NMA is a study of a high-dimensional potential energy surface under harmonic approximation. In this work, the INCHING algorithm is applied to the elastic network model (ENM), a practical form of NMA introduced by Tirion[15]. As described in Methods, NMA is based on solving a standard eigenproblem, $HQ = \Omega Q$, concerning a Hessian matrix $H \in \mathbb{R}^{3N\times 3N}$, where $\Omega \in \mathbb{R}^{3N\times 3N}$ and $Q \in \mathbb{R}^{3N\times 3N}$ are the eigenvalue matrix and the eigenvector matrix, respectively. For large collective conformational changes, only the eigenpairs with the smallest non-zero eigenvalues (the lowest-frequency modes) are of interest. However, challenges in the construction and diagonalization of the Hessian grow with the system size, and are notoriously resistant to parallelization. Consequently, current approaches to NMA are not readily adaptable for GPU computing, which imposes even stricter memory limits and algorithmic requirements to fully leverage on-chip parallelism. As illustrated in Fig 1a, the INCHING algorithm was implemented in three synergistic stages – permutation, construction, and diagonalization – to resolve the challenges of calculating NMA on GPU. In the following sections, we will describe each stage in detail.

Permutations to Achieve Bandwidth Reduction

GPUs are hardware engineered to efficiently access contiguous memory locations, thus inherently favoring dense computations. However, in practice, the experience of memory access is dictated by the sparsity

pattern of the data presented. Since the inception of NMA, it has been recognized that the Hessian is globally-sparse with prevailing zeros, primarily due to the long-range cut-offs applied to molecular mechanics potentials.[11] These globally-sparse Hessians are also locally-sparse, requiring very large bandwidths with non-uniform adjacencies (Fig. 1b c.f. Fig 1c.) due to sequentially distal segments of polymer(s) interacting in tertiary and/or quaternary structure(s) [39,40] (Extended Data Figure 1), and the situation is further complicated by chemicals (e.g., cofactors, ions, water, lipids) with no inherent order being integrated into the polymer structure (Extended Data Figure 2). All these factors contribute to inefficient on-chip parallelism when dense row-sweeps were performed in batches.

The primary objective of the INCHING algorithm is to strategically permute columns and rows of the Hessian, prior to all its computations, such that a globally-sparse-yet-locally-dense computation can be achieved in subsequent stages. By permuting the atom ordering, we can always generate a graph isomorph of the original elastic network, while allowing retrieval of its unpermuted eigenpairs in linear complexity if a bijection mapping is provided. However, finding a permutation that exactly minimizes the bandwidth of a matrix is NP-complete[41]. To approximate this reduction, we have designed a level-structure algorithm called 3DRCM, which extends on the well-established Reverse Cuthill-McKee (RCM) [42,43] algorithm, to handle 3-D coordinate data. Traditional RCM and variants[44–46] operate under the assumption that the input —a matrix— is realized and stored once-and-for-all. This framework is suitable for testing new offline analyses on the preserved matrix. However, a major problem overlooked is that it may not even be practical to realize the matrix efficiently, when its parallel computation spans a large bandwidth. In 3DRCM, a 3-D-tree data structure[47] is taken as input instead of a matrix to eliminate this circular reference. This modification enables efficient dynamic neighborhood lookups in the permutation process and avoids premature realization of the Hessian matrix. In Extended Data Figure 3a, we showed that for macromolecules with over 100 thousand atoms, the mean bandwidth for the Hessian in default PDB sequence ordering can vary between 10% and 90% of N. However, by permuting the Hessian with 3DRCM before its construction, the mean bandwidth of the Hessian is consistently reduced and remains below 10% of N, hence confirming the local density of the matrix. In Method and Extended Data Figure 3b, we also

showed that the time complexity of 3DRCM is practically dominated by a linear term proportional to the packing density of the macromolecule, thereby ensuring the cost-effectiveness of the algorithm.

GPU-resident Construction of the Hessian with Batched Tensor Products

The proposed 3DRCM permutation enables spatial neighbors from each batch of atoms to be retrieved as locally-dense indexing slices. In our Method, we further showed that, by extending on the tensorial representation suggested by Koehl[30], the Hessian can be constructed in batches through vectorized operations, such as tensor products and broadcasts, to leverage efficient on-chip parallelism in GPU. This replaces the need for explicit index loading and external storage of the distance matrix to enable fast computations. Besides, in our implementation, only the lower triangle of the Hessian was incrementally stored, and later accessed, in Compressed Sparse Row (CSR) format, this halves the memory requirement. In Extended Data Figure 4, we showed that even for a system comprising 2.4 million atoms (PDBID: 3J3Q), the computation of the 3DRCM permutation ordering and the subsequent construction of the Hessian on GPU took only 5.2 minutes and 6.9 minutes in wall-clock time, respectively. Our method shows significant speedup over ProDy2.4 [48], an open-source implementation intended for coarse-grained protein representation in ANM tasks. For the largest case that ProDy2.4 handled (PDBID: 6NCL), containing 305 thousand atoms, ProDy2.4 took 7.8 hours to construct the Hessian, whereas our INCHING approach delivered the matrix in just 93 seconds, showing a 302-fold acceleration.

GPU-resident Diagonalization of the Hessian Matrix

In NMA calculations, a significant portion of computational resources is often dedicated to diagonalizing the Hessian, where the throughput is dependent on the performance of the eigensolver. One popular method in solving large symmetric eigenproblem is the Implicitly Restarted Lanczos Method (IRLM)[49], which is available in the ARPACK package[50], a backend incorporated into ProDy2.4. In ARPACK, the matrix-vector multiplications are accelerated by threaded BLAS subroutines. However, despite the utilization of a

high-end AMD processor with 64 threads, ProDy2.4 routines (ProDy-ARPACK-FullSparse) fail to converge within 48 hours when systems exceed 305 thousand atoms (PDBID: 6HIV, 311 thousand atoms), indicating that we have reached the limit of hardware improvement. Note that ProDy2.4 with the default LAPACK[51] backend (ProDy-LAPACK-FullDense) working on a full dense Hessian is faster than all other programs when there are less than 500 atoms, but we were not able to proceed once exceeding 29 thousand atoms due to rapid rise in memory consumption. We also implemented a version of INCHING (INCHING-ARPACK-FullSparse) that uses our fast Hessian construction routine on GPU followed by a one-time device-to-host transfer to carry out diagonalization with ARPACK, but we were not able to proceed beyond 1.2 million atoms in 48 hours, reflecting the bottleneck in calculation is fundamentally the diagonalization process.

GPU-accelerated approaches for solving large sparse symmetric eigenproblems continues to be a vibrant area of research.[52–57] To explore this next-generation technology, we implemented, optimized and compared several diagonalization methods on a single NVIDIA® GPU device, including the Implicitly Restarted Lanczos Method (IRLM)[49,50], the Thick Restart Lanczos Method (TRLM) [58] , the Jacobi Davidson Method (JDM)[31,32] and some of their Chebyshev-filtered versions e.g. the Chebyshev-filtered Thick Restart Lanczos Method (CTRLM)[34] and Chebyshev-Davidson Method (CDM)[35]. In general, these methods all share the objective of computing eigenpairs within an interval (e.g., those with the smallest eigenvalues), but they differ in their approaches to update the Krylov subspace that approximates the eigenpairs. (See Methods for detail). The IRLM and TRLM are variants of the Hermitian Lanczos Method (HLM), but they differ in how they utilize the solutions of a much smaller tridiagonal eigenproblem to reinitialize the Krylov subspace at restarts. On the other hand, the JDM directly corrects the Krylov subspace by solving for an approximate fit to eliminate residuals and subsequently also works over solutions of a smaller projected eigenproblem. The CTRLM and CDM uses filters (low- or band-pass) constructed from Chebyshev polynomials to magnify wanted interval in the spectrum. The choice among these methods is not obvious, though their speed all depends on the cost of matrix-vector multiplication. In our INCHING protocols (INCHING-TRLM-HalfSparse, INCHING-IRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-

CTRLM-HalfSparse, INCHING-CDM-HalfSparse), these multiplications are accelerated by the SpMV CSR kernel in cuSPARSE[59]. The GPU computation was instructed through CuPy[60], a python API to NVIDIA®'s CUDA[61], cuBLAS[62], cuSPARSE[63] and custom kernel programming. To accommodate the lower memory capacity of GPU, the Hessian is accessed as a sum decomposition of its lower triangle, halving the memory requirement. Alternatively, the calculations can also be done in the full sparse matrix with a further 40% speedup, when GPU memory is not exhausted. (See Extended Data Figure 5) In Extended Data Figure 6, we further showed that an explicit external deflation[36] of the first 6 rigid modes with zero eigenvalues (rotations and translations) can help to improve runtime of the remaining 58 non-zero eigenpairs in sub-megascale regime while sharing very similar memory footprint. In addition, by incorporating low- and band-pass Chebyshev filters[34,57], we also lifted the memory limit regarding the number of modes to be resolved. (See INCHING-CTRLM-HalfSparse in Figure 2 and Extended Data Figure 8)

Benchmarks

Benchmarks on correctness, memory and speed for calculating the first 64 modes are illustrated in Fig. 2. All reported runtimes are wall-clock time, and the tests were conducted on a computer with a 64-threads AMD EPYC™ 7513 processor and a single NVIDIA® A100 Tensor Core GPU, unless specifically noted. The accuracy is measured by the 2-norm of the residual error. Across 116 benchmark cases encompassing macromolecules ranging from around a thousand atoms to around 2.4 million atoms, including PDBID:3J3Q, the largest experimentally resolved atomic structure in PDB at <10Å range, our INCHING protocols were able to afford accuracy at 10^{-12} level and achieved a peak memory consumption consistently lower than the ProDy2.4 routines. Importantly, with memory requirement halved by accessing only the lower triangle of the Hessian with a sum decomposition, the throughputs of our INCHING protocols, including construction and diagonalization of the Hessian, are still 146-251 times faster than ProDy-ARPACK-FullSparse and are 265-1290 times faster than ProDy-LAPACK-FullDense, depending on our choice of diagonalization algorithm. Remarkably, for a 305-thousand-atoms system (PDBID:

6NCL), all of our INCHING protocols took at most 5 minutes to converge with the fastest convergence being achieved by INCHING-TRLM-HalfSparse at 2.9 minutes. The same task took ProDy-ARPACK-FullSparse 12.4 hours to converge. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse and their Chebyshev-filtered versions (INCHING-CTRLM-HalfSparse, INCHING-CDM-HalfSparse) also converged for the HIV-1 capsid structure (PDBID:3J3Q, 2.4 million atoms), with the fastest convergence achieved by INCHING-CTRLM-HalfSparse within 44 minutes. In Extended Data Figure 8a-c, we show that, at the same accuracy level as JDM (10^{-12}), moderate speed-up in sub-megascale regime (mean at 1.21) can be achieved by CDM with an 80-degree polynomial, though the speed-up diminished to a slow-down in the megascale regime (mean at 0.91). This is in contrast to applying an optimized low-pass filter to TRLM, where we showed that the Chebyshev-filtered TRLM (CTRLM) can steadily deliver a speed-up over TRLM in the megascale regime (mean at 7.44). In Extended Data Figure 8d-e, we further showed that linear or better scaling in runtime has been achieved in terms of the radii R_C = 6,8,14Å to be considered, except for several cases at a lower radius R_C = 6Å, likely due to poorer conditioning of the matrix. In Extended Data Figure 9, by incorporating a band-pass Chebyshev filter developed in recent works[34,57] into our TRLM implementation (INCHING-CTRLM-HalfSparse), we also achieved linear time scaling in the number modes to be solved, while keeping memory usage constant. This ultimately allows 1000 modes of all the benchmarks to be solved, in batches of 64 eigenvectors, without compromising memory or run time or atomic details. The method was also tested on a RTX4090 NVIDIA® GPU in batches of 28 eigenvectors in Extended Data Figure 10b,c.

Anisotropic vibrations of some megascale systems

To illustrate the usage of our software, we have applied INCHING to some of the largest atomic objects available, both natural and artificial. In Fig 3, we illustrate the first non-zero mode of the mature HIV-1 capsid structure (PDBID:3J3Q), the largest experimentally resolved atomic object to date at <10Å resolution range, containing 2.4 million atoms and 1.6 billion non-zero entries in its Hessian. The cone-shaped capsid is an assembly of 186 hexamers and 12 pentamers, and it was suggested that these pentamers

located at its hemispherical ends induce stable closure of the capsid by allowing sharp bite angles at the surface.[64] With our INCHING-JDM-HalfSparse protocol, we were able to resolve its first 64 normal modes at residual error 10^{-12} within 1.3 hours on an NVIDIA® A100 Tensor Core GPU, where the first non-zero mode corresponds to the oscillation of its hexameric surface roughly anchored at the pentamers supporting the theory of quasiequivalence.[65] In Fig 4, the normal modes of a dilated human nuclear pore complex (NPC)[66] resolved at 50 Å were shown (EMDB: EMD14321, PDBID: 7R5J). The structure contains 4.8 million atoms and 3.5 billion non-zero entries in its Hessian. We were able to obtain the first 30 non-zero modes of this NPC structure within 12.5 hours at accuracy of 10^{-12}. In this calculation, an explicit external deflation[36] was applied to remove the first six rigid modes. We observe that while the first 14 non-zero modes mostly concern motions in the cytoplasmic and nuclear ring, the constriction of the inner ring can be observed at the fifteenth non-zero mode, reflecting the key functionally relevant conformational change. This calculation is at the edge of 80 GB of memory limit for the hardware used, though more powerful GPU systems on the market could handle even larger macromolecule superstructures. In Fig 5, we applied the same methodology on the largest artificial DNA origami nano-structure, a DNA airplane made of 33kbps at its relaxed state[67] containing around 1.8 million atoms, including hydrogens, with a Hessian containing 1.8 billion non-zero entries. The structure has an apparent bilateral symmetry, where the joints leading to the wings are not chemically symmetric.[67] Interestingly, while its first non-zero mode presents a symmetric flop in its wings, the second non-zero mode demonstrates a complicated non-symmetric twisting motion involving its wings and stabilizers. In Extended Data Figure 10a, we also solved the first 64 modes of a 5-million pseudo-atoms coarse-grained representation of a Faustovirus capsid[68] (PDBID: 5J7V 26 million atoms resolved at 15.5 Å), illustrating the potential to incorporate coarse-graining strategies in handling systems that cannot fit into memory.

Discussion

In recent years, there has been a significant shift in the focus of NMA methodologies, with an emphasis on accommodating larger atomic systems. The aim of an NMA is to gain insight into how some macroscopic motions involving communicating domains, can arise from microscopic interactions at the atomic level. In this respect, the mode shapes of NMA, which orchestrate collective motions concerning distal parts of the macromolecule, can often provide hints to understanding the biological structure at hand. Existing methods to analyze NMA can readily handle medium-sized structures with a few thousand atoms, but beyond this application of NMA is limited by computing resources and is not easily scalable by parallelization. This fundamentally restricts the molecular plasticity analysis in areas such as Cryo-ET and Cryo-EM, where macromolecular structures studied are often composed of millions of atoms and rather flexible. In these applications, fast and accurate mega-scale NMA would greatly facilitate modeling conformational dynamics in the refinement of the composite density maps[19–21] and the flexible fitting of atomic models[22–24] when EM volume or atomic-resolution template structures are available. Two major challenges in NMA scaling to mega-atom macromolecular complexes are the construction of the Hessian matrix and solution of the large-scale eigenproblem entailed. In this work, we have developed and implemented several advanced numerical recipes optimized for GPU computing, including a bandwidth-reduction algorithm and several alternative eigensolvers, to handle these challenges. This allows us to resolve normal modes, under the practical form of elastic network model, without coarse-graining the provided atomic structure for systems with several million atoms. In many cases, coarse-graining of biomolecular residues is expert-driven, and the appropriate level of granularity can be hard to determine when the chemical moiety is elongated (e.g., lipids) or is planar in shape (e.g., aromatics), especially in heterogeneous complexes comprised of protein, nucleic acid, lipid membranes, and sugars. The coarse-grained particle motions must be broadcasted into an all-atom construct if an atomic-scale understanding of the system is desired. In this respect, the implementation of an all-atom NMA, faithful to the formulation of Tirion[15], is a direct response to this shortcoming as all the atoms are now included in the system without neglecting each of their degrees

of freedom (DOF). Several recent approaches to reduce DOF of atomic systems are interesting and indispensable to further scale up though. One of the first and the most popular approaches is the Anisotropic Network Model (ANM), which consider a subset of n_b atoms from the all-atom system, usually only the phosphorus P in nucleic acids and/or the backbone carbon Cα of the proteins[15,25], thus effectively reducing N for 8-10 times. Agreement of ANM with experimental data[18,25] is satisfactory, though the displacement information of the discounted atoms is lost. This issue was alleviated by the rotation-translation block (RTB) method[26], where a projection operator was developed to coarse-grain the atomic system into a system of n_b rigid blocks, each with its own translational and rotational DOF, hence effectively reducing the $3N \times 3N$ Hessian matrix to a $6n_b \times 6n_b$ RTB matrix. Further refinement along this line is the Block Normal Mode (BNM)[27] method, which surrogates the initial realization of the peak-memory-consuming $3N \times 3N$ Hessian matrix; this was done by exploiting the block structure of the projection operator and by constructing only part of the sparse full atomic Hessian on-the-fly. Very recently, it was also shown that an extrapolation of RTB modes can effectively predict nonlinear motions at large amplitudes. [69] Depending on the granularity of the system, more than tenfold reduction in the size of the Hessian matrix can be achieved. However, the cost of diagonalization can still be very prohibitive.

In this respect, parallel computing on GPU, as we implemented in this work, is an effective way to amortize the cost at a fundamental level, without compromising accuracy or atomistic detail. We also expect that advances in GPU hardware[70] in memory and processing rate, integrated with techniques in solving large-scale eigenproblems, in particular recent trends in spectrum slicing [34,52–57,71], would eventually allow even faster NMA calculations on systems exceeding 20 million atoms, such as structure of a Faustovirus[68] available in low resolution (PDBID: 5J7V, 15.5 Å). Nonetheless, given the applicability of NMA in biological systems, we believe our framework will be useful in the exploration of megascale structural dynamics. The growing complexity of the megacomplexes resolved by Cryo-EM and/or cryo-ET now calls for methods that can rapidly capture conformational and functional plasticity, potentially as an intrinsic part

of the refinement pipeline. Such methods will play a crucial role in advancing our understanding of these macromolecular machines.

Methods

Normal Mode Analysis of an Elastic Network Model

The Normal Mode Analysis (NMA) is a classic approach to derive motions from static structures at local minima of a potential energy surface. The potential energy V is dependent on the conformation $X \in \mathbb{R}^{3N}$ for a structure with N atoms at time t. Without loss of generality, we begin with a particular conformation $X^{(0)} \in \mathbb{R}^{3N}$. For small displacements, we may then tolerate a second-order Taylor expansion,

$$V(X|X^{(0)}) = V(X^{(0)}|X^{(0)}) + \nabla V(X^{(0)}|X^{(0)})(X - X^{(0)}) + \frac{1}{2}(X - X^{(0)})^T \nabla^2 V(X^{(0)}|X^{(0)})(X - X^{(0)}) + \cdots$$

(Equation 1)

By choosing the energy level $V(X^{(0)}|X^{(0)}) = 0$ and assuming local minimum $\nabla V(X^{(0)}|X^{(0)})^T = 0$, we are left with the quadratic form,

$$V_{NMA} = \frac{1}{2}\Delta X^T H \Delta X$$

(Equation 2)

, where $H =: \nabla^2 V(X^{(0)}|X^{(0)})^T \in \mathbb{R}^{3N\times 3N}$ is the Hessian matrix and $\Delta X =: (X - X^{(0)}) \in \mathbb{R}^{3N}$ is the deviation from the minimum. We will defer the layout of H to the next section when the form of potential energy V is defined and proceed to discuss the outcome of an NMA. Substituting V_{NMA} into the equations of motion gives a partial differential system

$$M\frac{d^2\Delta X}{dt^2} = -H\Delta X$$

(Equation 3)

And, the general solution $\Delta X = Qe^{-i\omega t}$ gives a generalized eigenproblem

$$HQ = MQ\Omega$$

(Equation 4)

Where the eigenvector matrix $Q \in \mathbb{R}^{3N\times 3N}$ can be viewed as an eigentensor $Q \in \mathbb{R}^{3N\times N\times 3}$. The physical meaning of this eigentensor Q is that each of its atom slice (the second index) gives a displacement vector, hence $3N$ linearized mode shapes, i.e., the collective motions, of the static structure $X^{(0)}$ can be obtained. In NMA, the first 6 modes will always have zero eigenvalues corresponding to rigid translational and rotational displacements. The eigentensor Q is also the best displacement under orthonormality constraint $\min_{Q} V_{NMA} \;\; s.t. \;\; Q^T Q = I$. Eigenfrequency $\sqrt{\omega_i}$ is the square root of the eigenvalue ω_i.

A variety of V exists. In the standard NMA, potential energies from a detailed all-atom forcefield can be used. However, this approach suffers from the tedious, and sometimes virtually unachievable, requirement of energy minimization.[10] This requirement was surrogated in the work of Tirion[15], where the potential energy of an atomic system was considered as an elastic network model (ENM) connecting atoms i and j

$$V(X) =: \sum_{ij} \frac{1}{2} k_{ij} \left(r_{ij} - r_{ij}^{(0)}\right)^2$$

(Equation 5)

Where r_{ij} is the variable Euclidean distance between atom pairs; $r_{ij}^{(0)}$ is the equilibrium Euclidean distance from $X^{(0)}$. Importantly, a Heaviside function k_{ij}, parameterized on $r_{ij}^{(0)}$, was used to eliminate the long-range interactions beyond the threshold R_C. This can also be considered as applying an adjacency matrix $k_{ij} \in K$ to the network.

$$k_{ij}\left(r_{ij}^{(0)}\right) = \begin{cases} 1, & r_{ij}^{(0)} \le R_C \\ 0, & r_{ij}^{(0)} > R_C \end{cases}$$

(Equation 6)

As in the formulation of Tirion, a unified mass $M = I$ were taken, hence reducing the generalized eigenproblem to a standard eigenproblem.

$$HQ = Q\Omega$$

(Equation 7)

Permuting the Hessian to Produce Graph Isomorphs

In our later exposition, we will permute the Hessian matrix such that the resultant matrix is locally dense. Observe that a similarity transform of H with permutation matrix $P \in \mathbb{R}^{3N\times 3N}$ preserves the eigenvalues Ω as

$$(PHP^T)(PQ) = (PQ)\Omega$$

(Equation 8)

and the eigenvector P^TPQ of the original problem can be recovered by simply applying inverse of the unitary permutation matrix, which is its transpose. Importantly, if we restrict the permutation to atom ordering, i.e., permuting every 3 consecutive indices as an immutable group in the $3N$ indices, then the resultant Hessian is a representation of a graph isomorph of the original elastic network, where the bijection is provided by correspondence between the original atom ordering and the permuted atom ordering. This property means that the similarity transform can be computed by simply initiating with a permuted coordinate and similarly Q can be recovered by permuting the atom slice of PQ using the same bijection with linear cost.

Constructing the Hessian Using Batched Tensor Products and Broadcasts

We begin with the following un-vectorized notation, where the superscript (0) is dropped for readability.

$$X^{(0)} =: \left[X_1^{(0)}, \dots, X_n^{(0)}\right]; Vec(X^{(0)}) =: [x_1, y_1, z_1, \dots, x_n, y_n, z_n\ ]$$

(Equation 9)

Applying differentiations, the off-diagonal force constant block is specified by

$$H_{ij} = -\frac{k_{ij}}{\left(r_{ij}^{(0)}\right)^2}\left(X_i^{(0)} - X_j^{(0)}\right)\left(X_i^{(0)} - X_j^{(0)}\right)^T$$

$$= -\frac{k_{ij}}{\left(r_{ij}^{(0)}\right)^2}\begin{bmatrix} \left(x_i - x_j\right)^2 & (x_i - x_j)(y_i - y_j) & (x_i - x_j)(z_i - z_j) \\ (y_i - y_j)(x_i - x_j) & \left(y_i - y_j\right)^2 & (y_i - y_j)(z_i - z_j) \\ (z_i - z_j)(x_i - x_j) & (z_i - z_j)(y_i - y_j) & \left(z_i - z_j\right)^2 \end{bmatrix}$$

$$H_{ii} = -\sum_j H_{ij}$$

(Equation 10)

For any matrix A with n rows, we can define for a row in it,

$$\beta_i =: \left| i - \min_j (j \mid a_{ij} \neq 0) \right|$$

(Equation 11)

Then the bandwidth of the matrix is $\max_i(\beta_i)$ and the profile of the matrix is $\sum_i \beta_i$, the mean bandwidth is $\frac{\sum_i \beta_i}{n}$. Clearly, due to the cutoff R_C, the layout of the Hessian tensor $H \in \mathbb{R}^{N\times 3\times N\times 3}$, viewing from the atom slices, will have the same bandwidth as the adjacency matrix K of the macromolecule. The tensor is globally sparse in most cases. This observation also applies to standard NMA as long range cut-offs were used for van der Waals, electrostatic, and hydrogen-bonding interactions.[12] In general, it is not advisable to construct a dense Hessian matrix that contains a large number of zero entries due to the quadratic storage consumption and the resultant increase in fill-ins. To construct a sparse Hessian matrix, we find the tensorial layout by Koehl[30] a good starting notation. In which, he defined the following $N \times 3$ matrix, which can also be vectorized as $3N$ elements.

$$U_{ij}\left(X^{(0)}\right) =: \left(0, \ldots, 0, \frac{X_i^{(0)} - X_j^{(0)}}{r_{ij}^{(0)}}, 0, \ldots, 0, \frac{X_j^{(0)} - X_i^{(0)}}{r_{ij}^{(0)}}, 0, \ldots, 0\right)$$

(Equation 12)

The distance $r_{ij}^{(0)}$ can be precomputed and filtered by $k_{ij}\left(r_{ij}^{(0)}\right)$ for pairs within R_C. Then, the off-diagonal block of the entire Hessian tensor is given as a sum of Kronecker products.

$$H_{ij} = \sum_i \sum_j k_{ij} U_{ij} \otimes U_{ij}^T$$

(Equation 13)

More importantly, when Hessian-vector multiplication Hw is accessed in eigensolvers, the following can be done separately for each (i, j) pairs.

$$Hw = \sum_{ij} k_{ij}\left(U_{ij} \otimes U_{ij}^T\right)w = \sum_{ij} k_{ij}(U_{ij}w)U_{ij}$$

(Equation 14)

Given explicit indexing for locations of the non-zeros k_{ij}, the above Hessian-vector product can be effectively parallelized on a hyperthreaded computer.

In our implementation, the tensorial representation above was refined to fully exploit on-chip parallelism on GPU and to avoid explicit index loading which tends to thread divergence on GPU. To facilitate discussion, we define a difference matrix $G_{ij} \in \mathbb{R}^{N\times 3}$

$$G_j\left(X^{(0)}\right) =: \left(0, \dots, 0, X_{b_3}^{(0)} - X_j^{(0)}, \dots, X_i^{(0)} - X_j^{(0)}, \dots, X_{b_4}^{(0)} - X_j^{(0)}, 0, \dots, 0\right)$$

(Equation 15)

Supposed, we know $|b_3 - b_4|$ is the bandwidth of an off-diagonal batch of H, then, for two contiguous atom indices $(b_1, b_2]$ and $(b_3, b_4]$, the batch $G \in \mathbb{R}^{3|b_1-b_2|\times 3|b_3-b_4|}$ can be given by the four index corners (b_1, b_2, b_3, b_4).

$$G_{(b_1,b_2,b_3,b_4)} = \sum_{i\in(b_1,b_2]} \sum_{j\in(b_3,b_4]} G_i \otimes G_j^T$$

(Equation 16)

In practice, $(b_3, b_4]$ could include intervals that lack neighboring atoms for the range $(b_1, b_2]$, but the indexing slice $(b_3, b_4]$ can be further refined into multiple contiguous indexing slices to eliminate unnecessary calculations. Nevertheless, filtering for non-zero elements in the batch can be done by observing Equation 10 in that the squared interatomic distance is given by the trace of each block $\left(r_{ij}^{(0)}\right)^2 = tr\left(G_{ij}\right)$, then we can consider the following block-wise operations, readily parallelizable on GPU by in-place tensorial broadcasts.

$$Y_{ij} = tr\left(G_{ij}\right)$$

$$K_{ij} = \begin{cases} 1, & Y_{ij} \leq R_C^2 \\ 0, & Y_{ij} > R_C^2 \end{cases}$$

$$Y_{ij} = \begin{cases} Y_{ij}^{-1}, & \forall i \neq j \\ 0, & \forall i = j \end{cases}$$

$$H_{(b_1,b_2,b_3,b_4)} = \left(\left(K_{(b_1,b_2,b_3,b_4)} \odot Y_{(b_1,b_2,b_3,b_4)}\right) \otimes 1_{3\times 3}\right) \odot G$$

(Equation 17)

The $\odot$ is the Hadamard product operator; $1_{3\times 3}$ is the all-ones matrix presenting the broadcast. This is followed by row-sum in the diagonal $H_{ii} = -\sum_{j\in(b_3,b_4]} H_{ij}$ accordingly. Only the lower triangle is incrementally stored in the Column-Sparse-Row (CSR) format for subsequent calculation, hence consuming $O(9p)$ memory for the data content; p is the number of non-zero pairwise interaction for $i \geq j$. The dense batch $G_{(b_1,b_2,b_3,b_4)}$ presented above will require $O(9|b_1 - b_2||b_3 - b_4|)$ erasable memory, but we will show later that the local bandwidth $|b_3 - b_4|$ can be reduced.

Bandwidth and Layout of the Hessian

Obviously, the cost of the parallel computation will depend on the local bandwidth $|b_3 - b_4|$ of the batch $(b_1, b_2]$. The Hessian obtained with the default ordering of atoms can result in very large bandwidths with non-uniform patterns meaning interactions among sequentially distal parts within the tertiary structure or individual chains within a quaternary structure. A hypothetical minimal example is a macromolecule with

3 peptide chains α, β and γ, where only α-γ and β-γ interactions were found, but not α-β; in this case, the order α-β-γ will have a much larger bandwidth than α-γ-β. Many reasons can attribute to this observation. An example found in the benchmark set is a chimeric Sesbania mosaic virus coat protein [72](PDBID: 4Y5Z) composed of 12 sets of pentamers arranged on the vertices of an icosahedron, where each protein chain in the pentamer interacts with 7 other chains within 8 Å. (See Extended Data Figure 1) A systematic order to build up this icosahedron, as done by the authors, is to place a pentamer on each of the four corners of the three orthogonal golden rectangles, done one rectangle after another starting at the shorter sides of the golden rectangles. (See Extended Data Figure 1a) In this case, pentamers on the shorter sides interact within 8 Å among each other, but pentamers on the long sides of the same rectangle do not. This creates exactly the situation where the aforementioned α-β-γ order arises. Indeed, the vertices on a icosahedron can never be clustered satisfactorily and there are specialized algorithm to calculate their vibrational dyanmics[73]. Besides, due to the inconsistent presence of water molecules, the exact symmetry is destroyed in the crystal structure, which is commonly encountered. Nevertheless, the degeneracies in modes due to the apparent symmetry of this macromolecule is captured in our program. (See Extended Data Figure 1d) There are also cases where no meaningful sequential ordering is possible when intact chemical structures (e.g. cofactors, ions, water, lipids) are intercalated between sequentially distal parts of the polymer. A practical example in the benchmark set is a PsbM-deletion mutant of photosystem II [74] (PDBID: 5H2F), where elongated lipids and cofactors are integral part of the macromolecule complex surrounded by several protein chains. In this case, it is not obvious as for how to rearrange the atoms to reduce the bandwidth, but we can show that the Hessian can always be permuted to reduce bandwidth by the algorithm described in the next section. (See Extended Data Figure 2)

Bandwidth Reduction of the Hessian Matrix

Finding a permutation to minimize the bandwidth is NP-complete[41], but a reduction of bandwidth and its overall profile can be approximated by algorithms such as the Reverse Cuthill McKee (RCM) algorithm. The RCM is a greedy approximation with a breadth-first level structure. The input of a

standard RCM algorithm is an adjacency graph, and its outputs is a permuted node ordering. In RCM, starting from a peripheral node with the lowest degree of connection, adjacent unvisited nodes are collected as a level structure and re-ordered by their degree of connection. Provided that an adjacency matrix K is precomputed, the time complexity of the RCM is bounded by $O(4|E| + 2cm|E| + N)$, where $m =: \max_i \sum_j K_{ij}$ is the maximum degree among all nodes and $2|E|$ is the number of edges in K. A detail proof is in reference [43]. The term $4|E|$ is referring to $2|E|$ operations to determine the degree of each node plus another $2|E|$ operations to sweep through the adjacency matrix to locate the neighbors. The term $2cm|E|$ refers to the insertion sorting of the degree in retrieved neighbors. The last term N is due to reversal of order. The input of RCM is an adjacency matrix K. For 3-D coordinates, this may be obtained by a cell-linked list data structure in $O(27Nn_c)$ where n_c is the average number of particles per cell in volume R_c^3 or simply an all-to-all calculation as done in ProDy2.4 for small-sized systems. Therefore, a standard RCM algorithm operating on 3-D coordinate data will require a total time complexity of $O(27Nn_c + 4|E| + 2cm|E| + N)$ and ideally a $O(mN)$ memory to operate due to the adjacency matrix. To surrogate the storage of the adjacency matrix, we supplemented the RCM with a k-D tree data structure[47], which takes the coordinate data $X^{(0)}$ directly as input. The algorithm is labelled as 3DRCM to avoid confusion with the standard RCM. A pseudocode of our 3DRCM implementation is provided in the Supplementary Information as Algorithm 1. A 3-D tree is a balanced space-partitioning binary tree, which can be constructed and stored by finding and storing hyperplanes that split the number of points in halves. As such, the construction of a 3-D tree takes $O(NlogN)$ time and $O(3N)$ space. On a 3-D tree, each range search for adjacent neighbors within threshold distance takes $O(3N^{2/3})$ [75]. At this stage, the total time complexity of 3DRCM appears to be $O(2 * 3N^{\frac{5}{3}} + 2cm|E| + N)$, where the first term $2 * 3N^{\frac{5}{3}}$ is due to the collection of degree in each node and the search for neighbor, but we will show that in both 3DRCM and RCM the term $O(2cm|E|)$ is dominating. A tighter bound on $O(2cm|E|)$ in the context of NMA can be obtained as follows. For a stable macromolecule without atomic clashes, the shortest interatomic distance is due to covalent bonds at around 1 Å. The Kepler's conjecture proven by Hales[76] states that the

maximum volume ratio occupied by equidistant sphere packing is $\frac{\pi}{3\sqrt{2}} < 0.7405$. Hence, the maximum number of atoms allowable without clashes within radius R_C is $\rho =: R_C^3 \frac{\pi}{3\sqrt{2}} \geq m$. This packing density ρ presents an upper bound to the number m of non-zero entries in a row on the adjacency matrix. This bounds the total number of edges as

$$\rho N \geq mN > 2|E|$$

(Equation 18)

For an 8 Å radius, there can only be less than 380 atoms in absence of clash. Therefore, the time complexity of 3DRCM and RCM in the context of NMA are bounded by $O(6N^{\frac{5}{3}} + c\rho^2 N + N)$ and $O(27N\rho\ + 2\rho N + c\rho^2 N + N)$ respectively. Taking $c = 1$, the 3DRCM is dominated by the pseudolinear term $O(\rho^2 N)$ unless N is beyond 3.7 million atoms. From experience, for a system containing 2.4 million atoms, computing the permutation ordering with 3DRCM and the calculation of the Hessian took only 5.2 minutes and 6.9 minutes in wall-clock time respectively. See Extended Data Figure 4. Hence, for most practical purpose, the trade-offs in time complexity in 3DRCM compared to RCM is negligible.

Low-complexity Krylov-Subspace Eigensolvers

In this work we have implemented several iterative methods[31,32,49,58] to solve large eigenvalue problems on the GPU. Specifically, in NMA, we are mostly interested in the smallest non-zero eigenpairs. The Hessian matrix of concern is a sparse positive semidefinite real symmetric matrix without weak diagonal dominance. To facilitate communication, we adopt more generic notations here. The matrix of concern is notated as $A \in \mathbb{R}^{n\times n}$; the exact and the approximate eigenpairs are$(\Lambda \in \mathbb{R}^{m\times m}, U \in \mathbb{R}^{n\times m})$ and $\left(\widetilde{\Lambda} \in \mathbb{R}^{m\times m}, \widetilde{U} \in \mathbb{R}^{n\times m}\right)$ respectively; n and m denotes the dimension of the basis set and the number of basis respectively. The standard eigenproblem is thus $AU = U\Lambda$ and we assumed orthonormality among the exact

eigenvectors. The approximate $\tilde{\lambda} \in \tilde{\Lambda}$ can be obtained from the Rayleigh quotient $\tilde{\lambda} = \tilde{u}^T A\, \tilde{u}$ and the residual $r =: A\tilde{u} - \tilde{\lambda}\tilde{u}$ can always be evaluated by its 2-norm, the residual error $\left|\left|r\right|\right|_2$.

In the following paragraphs, we will progressively introduce two branches of iterative methods implemented, namely the Hermitian Lanczos Methods followed by the Jacobi-Davidson Method. Only rationales and key equations were presented. A common central idea is to find $\tilde{u}$ by refining an initial guess $v_0 \in \mathbb{R}^n$ to build up an orthonormal basis $V \in \mathbb{R}^{n \times m}$ in the Krylov subspace

$$\mathcal{K}_k(A, v_0) =: span\{\, v_0, A^1 v_0, A^2 v_0, \dots, A^{k-1} v_0 \,\}$$

(Equation 19)

The residual is minimized when the Galerkin condition

$$\left(V e_j\right)^T \left(A\, \tilde{u} - \tilde{\lambda}\tilde{u}\right) = 0 \;\forall\, j \in (0, \dots, m-1)$$

(Equation 20)

is satisfied, where $\tilde{u} =: Vy$. The $y \in \mathbb{R}^{m \times 1}$ is an unknown component to combine V, but if $V \in \mathcal{K}_k$ has its orthonormality maintained satisfactorily, for example by incorporating the Modified Gram Schmidt algorithm (MGS), then y is the eigenvector of a symmetric tridiagonal eigenproblem[77] of a much smaller size $m \times m$

$$(V^T A V) y = \tilde{\lambda} y,$$

(Equation 21)

and $\tilde{u} \in \mathbb{R}^n$ can be recovered by definition. This orthogonal projection technique, an example of Rayleigh-Ritz process, is common to all three methods implemented.

Technical remarks. Given the recurring application of certain techniques in the implemented methods, an assortment of technical remarks is presented here before progressing further. The MGS can be applied more than once to prevent loss of orthogonality due to float point roundoffs and the procedure is called a full

reorthogonalization (FRO). A pseudocode of orthogonalizing a vector against a basis with MGS, and similarly with an Iterative Classical Gram-Schmidt (ICGS) algorithm is provided in the Supplementary Information as Algorithm 2 and Algorithm 3. The low-complexity $O(n^3)$ cost of the matrix-vector multiplications in producing $\mathcal{K}_k$ can be amortized by parallelisms in GPU. When only the lower triangle L of A is available, the multiplication with an arbitrary vector x can be done as a sum decomposition $Lx + L^T x - Dx$, where $D =: diag(A)$. In our implementation, a custom kernel that utilizes the SpMV algorithm in CuSPARSE is written to accommodate the availability of the lower triangle, halving the memory requirement to access the Hessian. In all cases, the convergence rate of the i-th eigenpair is dictated by the ratio of adjacent exact eigenvalues $\left|\frac{\lambda_i}{\lambda_{i+1}}\right| \leq 1$, the smaller the better. Note that A was shifted upward as $A + I$ to avoid poor condition number. Several strategies to improve the convergence rate for the smallest eigenpairs includes (1) the Shift-and-Inverse technique, where the Ritz pairs of $(A - \sigma I)^{-1}$ closest to σ was sought instead and the inverse is incorporated into the matrix-vector product by solving for $v^{(k+1)}$ in $(A - \sigma I)v^{(k+1)} = v^{(k)}$, (2) the implicitly shifted QR algorithm by Francis[78], where multiple shifts were applied to the subspace iteration problem typically concerning a small eigenproblem (See part b of Algorithm 5) and (3) filter diagonalization[33,34,57] that magnifies convergence rate for regions of a spectrum (See Algorithm 4,8 and 9 and later sections). Finally, the six exact eigenvectors U_6 corresponding to the rigid modes can be removed from $\mathcal{K}_k$ by an explicit external deflation[36]; the dense deflated matrix $A - \sigma_H U_6\ I_6 U_6^T$, with eigenvalues corresponding to U_6 raised to σ_H, is not stored but incorporated into the matrix-vector multiplications; the sparse U_6 is stored in CSR format. This is implemented in all three methods and can be optionally called.

Hermitian Lanczos Method (HLM). The $V \in \mathcal{K}_k$ described earlier can be obtained iteratively as v_{k+1} by projecting away components of previously obtained $v_i \forall\ i \in 0 \ldots k$ from the current matrix-vector product Av_k. In the passing, the smaller eigenproblem $T =: V^T AV$ is also produced. Importantly, for a symmetric matrix A and $V \in \mathcal{K}_k$, the T is symmetric tridiagonal. Therefore, the recursion to obtain v_{k+1} simplifies to three terms

$$v_{k+1} = Av_k - \beta_k v_{k-1} - \alpha_k v_k$$

(Equation 22)

Where $\alpha_k =: v_k^T A v_k$ and $\beta_k =: v_{k-1}^T A v_k$ are the main diagonal and the first subdiagonal of T. This is the essence of the HLM proposed by Lanczos[79]. Collecting the recursion and a rearrangement reveals the Lanczos factorization stopping at the k-th step.

$$AV_k = V_k T_k + \beta_k v_{k+1} e_k^T$$

(Equation 23)

A pseudocode of a p-step Lanczos Factorization is provided in the Supplementary Information as Algorithm 4. Note that the approximate eigenvectors obtained from the HLM will converge to those with extremal eigenvalues, i.e., both the largest and the smallest eigenpairs, but only the smallest eigenpairs are wanted in our problem setting. Further, while the basis V_k builds up in each iterative step, eventually we will experience overflow in storage, likely before convergence. To address these concerns, two practical variants of the HLM with different restarting techniques were proposed.

- **Implicitly Restarted Lanczos Method (IRLM).** The IRLM algorithm was proposed by Lehoucq and Sorensen.[49] In IRLM, the Lanczos factorization is supplemented with an implicitly shifted QR algorithm by Francis [78]. The purpose is to shift the smaller eigensystem $T^{(j-1)}$ at each restart round j by the current approximation of the $m - k$ unwanted eigenvalues such that the convergence rate of the k wanted smallest eigenvalues is improved. The only information we need to determine these shifts is the sorted approximated eigenvalues and the number of wanted eigenvalues we desired. As a result, after obtaining each QR factorization $Q^{(j)}R^{(j)} = T^{(j-1)} - \tilde{\lambda}_j I$, the Lanczos factorization (Equation 23) is modified as

$$A\hat{V}_k = \hat{V}_k \hat{T}_k + v_{k+1}\hat{b}$$

(Equation 24)

where $\hat{b} = Q^{(j)^T} e_k$ and $\hat{T} = R^{(j)} Q^{(j)} + \tilde{\lambda}_k I$ and the first k orthonormal basis is updated as $\hat{V}_k = V_k Q^{(j)}$. This is equivalent to performing $m - k$ simple polynomial filterings. [80] In our implementation, to promote parallelism, the $m - k$ QR factorizations was performed before a cumulative update of the matrices; this is followed by a full reorthogonalization of the updated basis V_k. Note that in IRLM the residual error $||r||_2$ is bounded by the β_{k+1}, a result due to Paige[81] such that $||r||_2$ needs not be computed. A pseudocode of our IRLM implementation is provided in the Supplementary Information as Algorithm 5.

- **Thick-Restart Lanczos Method (TRLM).** The TRLM algorithm was proposed by Wu and Simon[58]. In TRLM, the smaller eigenproblem T is again solved with its eigenpairs sorted, but rather than performing implicit shift, the T is projected onto its wanted eigenvectors Y_k as $\hat{T} = Y_k^T T Y_k$ and the basis updated to $\hat{V} = V Y_k$ accordingly. As a result, the Lanczos factorization (Equation 23) is modified as

$$A\hat{V}_k = \hat{V}_k \hat{T}_k + \beta_k v_{k+1} e_k^T Y$$

(Equation 25)

When we want to find the next v_{k+1} by orthogonalizing Av_k against the previous $\hat{V}_k$, these transforms allow us to compute it with $\beta_{k-1}(Y^T e_{k-1})$ known from the restart

$$\hat{v}_{k+1} = A\hat{v}_k - \hat{\alpha}\hat{v}_k - \hat{V}_{k-1}\beta_{k-1}(Y^T e_{k-1})$$

Full reorthogonalization was done in the basis $\hat{V}_k$. A pseudocode of our TRLM implementation is provided in the Supplementary Information as Algorithm 6.

Jacobi-Davidson Method (JDM). The JDM algorithm was proposed by Fokkema, Sleijpen and Van der Vorst[32]. Similar to the HLM, the JDM also considers approximations in the Krylov subspace, but rather than refining the approximations with a Lanczos factorization, it attempts to solve the following equation

$$A(\tilde{u} + z) = (\tilde{\lambda} + \eta)(\tilde{u} + z)$$

(Equation 26)

where an approximate solution pair (η, z) to correct the approximate eigenpair $(\tilde{\lambda}, \tilde{u})$ can be obtained by incorporating the Galerkin condition $(\tilde{u} + z)^T z = 0$ into projectors resulting in the following correction equation

$$(I - \tilde{u}\tilde{u}^T)\big(A - \tilde{\lambda}I\big)(I - \tilde{u}\tilde{u}^T)z = -r$$

(Equation 27)

The correction equation does not need to be solved exactly such that A in this equation can be replaced by a preconditioner of A or by simply finding a solution of z to certain extent of precision. For the NMA eigenproblems, we cannot find a satisfactory preconditioner that is not dense such that GPU memory is not overflowed. Therefore, in our implementation, the latter strategy was adopted by restricting the number of steps or precision in the solution in the generalized minimal residual iteration (GMRES), which is also a Shift-and-Inverse example. An implementation of the GMRES taking a sparse matrix input was modified from the CuPy package. A pseudocode of our JDM implementation is provided in the Supplementary Information as Algorithm 7.

Filter Diagonalization

The Lanczos and Davidson methods can be adjusted by filters to isolate certain eigenvalues. The motivation of this Filter Diagonalization (FD) approach is to transform the spectrum such that eigenvalues from wanted intervals become dominant[33,82]. (See Extended Data Figure 10b for an illustration.) For example, to obtain the lower spectrum, the Chebyshev-Davidson Method[35] (CDM) removed the Jacobi correction in JDM (which approximates a moving rational filter) and applied a moving low-pass, fixed-degree, Chebyshev polynomial of the first kind to the input matrix. (Algorithm 8) FD can also be applied to solve for the interior rather than the extremal eigenpairs.[34,57] This can be useful when we are extracting a large amount of eigenpairs from the lower end of the spectrum, as then the interior slices of the spectrum can be obtained without storing and orthogonalizing against the filtered subspace from the very lowest end. To achieve this, a band-pass filter, which is an M-degree Chebyshev expansion of a Dirac-delta-like function damped by a

Jackson kernel[33,83], was developed in the EVSL library[34,57] and incorporated into our INCHING-CTRLM protocol (Algorithm 6, shared with INCHING-TRLM protocol). The band-pass filter $p(t)$ were obtained following the three-term recurrence of T_j, the j-th degree Chebyshev polynomial of the first kind.

$$T_{j+2}(t) = 2t\,T_{j+1}(t) - T_j(t); T_o(t) = 1\,; T_1(t) = t;$$

$$p(t) = \sum_{j=0}^{j=M} g_j^{(M)}\,\mu_j T_j(t) \,/\, \sum_{j=0}^{j=M} g_j^{(M)}\,\mu_j T_j(\cos(\theta)) \quad \text{(Equation 28)}$$

The coefficients μ_j in the expansion and the damping kernel $g_j^{(M)}$ were precalculated with the following equations, where δ_{j0} is the Kronecker delta and $\tilde{\pi} =: \pi/(M+2)$ and θ is the adjusted center of the arccosine of the corresponding transformed wanted interval $[\alpha_s, \beta_s]$ within $[-1,1]$. (See reference[34,57] for further details.)

$$g_j^{(M)} =: \frac{\sin\big((j+1)\tilde{\pi}\big)}{(M+2)\sin(\tilde{\pi})} + \left(1 - \frac{j+1}{M+2}\right)\cos(j\tilde{\pi})$$

$$\mu_j = -\frac{1}{2}\delta_{j0} + \cos(j\theta) \quad \text{(Equation 29)}$$

The filter was applied during matrix-vector multiplication (Algorithm 9), where the spectrum of A were scaled and shifted to the range of cosine $[-1,1]$ using spectral bounds from the Lanczos process (i.e. Algorithm 4) following reference [35].

Implementation notes

In many parts of the algorithms presented above, matrix-vector multiplication, whether dense or sparse, is often the calculation bottleneck. In our implementations, this shortcoming is amortized by capitalizing on the high degree of parallelism in GPU and existing techniques built around NVIDIA® GPUs, including CUDA[61] (v. 11.6.2), cuBLAS[62], and cuSPARSE[63]. To facilitate installation, code readability and version

control, the CuPy package (v. 11.5) was used as an application programming interface to access these technologies. In all the algorithms implemented, only the lower triangle of the Hessian matrix was required as input in CSR format. Our INCHING-JDM and INCHING-IRLM implementations were written with reference to the thesis of Geus[84] and the ARPACK package[50] respectively. Our INCHING-TRLM implementation is modified from the CuPy library's default, where memory footprint was optimized by exploiting sum decomposition of the lower triangle; further speedup was achieved by reducing the number of transposes done to the basis set. Where a smaller projected eigenproblem has to be solved, the calculation was done on CPU by calling 'numpy.linalg.eigh' from the NumPy (v. 1.23.5) package[85]. For methods with Chebyshev filter, the implementations in the EVSL library[57] and the Chebyshev-Davidson method[35] were referenced. When a band-pass Chebyshev filter is invoked, the converged eigenvectors are sorted again before off-loading for storage. For benchmarks with ProDy, the ProDy (v.2.4) package [48] was installed with the NumPy (v. 1.23.5) package[85] and the SciPy (v.1.8.0) package[86]. Dense eigenproblems in ProDy are solved using a LAPACK backend called through NumPy. Sparse eigenproblems in ProDy are solved using a ARPACK backend called through SciPy 'scipy.sparse.linalg.eigsh'. Benchmarks on correctness, memory and speed were conducted on a single piece of A100 NVIDIA® GPU with tensor core activated and 80GB GPU memory capacity and the AMD EPYC™ 7513 processor with 64 threads. For systems with more than 2.1 billion non-zero entries, 64-bit integers were used for indexing in the CSR format, otherwise 32-bit integers were used by default. Methods were also tested on a RTX4090 NVIDIA® GPU with 24GB memory and 64-bit indexing used throughout.

Hyperparameters in calculations

In all cases, the following hyperparameters were used unless otherwise stated. We followed the work of Koehl[30] taking the neighborhood cutoff threshold R_C = 8Å for atomic systems. After performing 3DRCM, the indexing slice for each batch of atom was analyzed for presence of gaps, i.e., regions where no neighbor of the batch is present. The indexing slice was then split into multiple contiguous indexing slices by removing gaps that extend for more than 100 atoms. For INCHING-IRLM, the tolerance of error bound is

set to 10^{-10}. For both INCHING-TRLM and INCHING-JDM, the tolerance of residual error is 10^{-12}. The correction equation in INCHING-JDM was solved using the GMRES algorithm, where only 20 steps of minimization at max were allowed. The default maximum allowed number of restarts for INCHING-IRLM, INCHING-TRLM and INCHING-JDM is 15000 steps, but for R_C = 6Å benchmarks, at max 30000 steps were allowed to accommodate difficult convergence in some cases. The maximum allowed basis in the Krylov subspace is three times the desired number of output eigenpairs, i.e. which is 64×3 = 192 vectors in the benchmark. The 1000 modes presented in Extended Data Figure 9 (200 modes in Extended Data Figure 10) were calculated using INCHING-CTRLM in batches of 64 modes with 128 basis (28 modes with 56 basis) for all structures, though larger basis were affordable; external explicit deflation of the free modes were applied when low-pass filter is used. To scan through the lower end of the spectrum using INCHING-CTRLM, we begin by applying the low-pass filter to obtain the largest of the smallest m eigenvalues α_s and set $\alpha_s - 10^{-10}$ as the left bound of the next interior wanted interval $[\alpha_s, \beta_s]$; the process continues with the band-pass filter until a desired number of eigenpairs are obtained. Note that a binary search was performed to locate β_s such that $[\alpha_s, \beta_s]$ contains $< m$ eigenvalues for the band-pass case and $< 5m$ eigenvalues for the low-pass case. The polynomial degrees were also maximized to maintain the smallest transformed eigenvalues in $[\alpha_s, \beta_s]$ as 0.7 for both filters such that convergence rate remained constant. (See Extended Data Figure 10b.). All programs were stopped if convergence was not achieved within 48 hours, wall-clock time. All INCHING programs were stopped if the maximum number of restarts were exceeded. For all benchmarks with R_C = 14Å, 64-bit integers indexing were used for consistency.

Benchmark coordinates

While ideally all the structures on the Protein Data Bank can be considered, we randomly selected structures at an increasing interval. For systems with less than 100 thousand atoms, structures were downloaded in the PDB format; the interval of increase is around 1 thousand atoms. For systems with more than 100 thousand atoms, structures were downloaded as mmCIF format; the interval of increase is around 10

thousand atoms. Note that while the iterative methods all work very well, some structures can contain components not connected within 8 Å threshold, resulting in more than 6 rigid modes. This happens in a small portion of PDB structures when crystallographic water(s) or protein chain(s) with no neighbor within 8 Å were scrupulously put into the crystallographic map. These structures were removed from consideration to avoid confusion. Overall, 85 structures with less than 100 thousand atoms and 31 structures with more than 100 thousand atoms were tested. This benchmark set contains 116 structures in total with number of atoms ranging from around 1 thousand atoms to 2.4 million atoms.

Megascale atomic structures

We applied our method to some of the largest atomic objects available. The mature HIV-1 capsid structure (PDBID:3J3Q) containing around 2.4 million atoms was obtained from the Protein Data Bank[37] without any modification. The human Nuclear Pore Complex structure (EMDB: EMD14321, PDBID: 7R5J) was downloaded as a bioassembly from the Protein Data Bank[37] and chains LA,MA,NA,OA with clashing linkers were removed; the final structure contains 4.8 million atoms. The atomic structure of the 26 million atoms Faustovirus capsid (PDBID:5J7V) was downloaded from the Protein Data Bank; Cα were extracted from the structure resulting in a 5 million pseudo-atom coarse-grained representation; NMA with R_C = 14Å was performed with INCHING-CTRLM-HalfSparse with external explicit deflation of the 6 free modes; the first 64 non-zero modes were obtained. The atomic structure of the DNA airplane[67] was obtained from Nanobase.org[87] and the atomic coordinates were reconstructed using TacoxDNA[88]; the final structure includes hydrogens and contains 1.8 million atoms. The NMA of HIV-1 capsid structure were calculated using INCHING-JDM-HalfSparse and the same hyperparameters of numerical methods outlined above. For the DNA airplane and the NPC complex, the NMA was calculated using INCHING-JDM-HalfSparse with external explicit deflation of the 6 free modes, the number of modes to resolve, the maximum allowed basis in the Krylov subspace, the maximum number of allowed steps in GMRES and the maximum number of restarts allowed were revised to the first 30 non-zero normal modes, 120 vectors, 150 GMRES steps and 550000 restarts respectively, otherwise all other hyperparameters are the same. The megascale structures were displayed using PyMOL (v2.4.1).

A logistic kernel to visualize atomic motion

In all the iterative methods, the resultant eigenvector matrix is orthonormalized, which means as size of the system increases a decreasing magnitude of atomic displacement vector will be experienced. In our visualization module, to avoid this scaling problem, a logistic kernel is applied to fine-tune the atomic magnitude x_i of the eigentensor. First, to avoid eccentricity in the atomic magnitudes, they are clipped between (0.025, 0.975) quantile of the magnitudes. Second, a logistic kernel is applied on the clipped magnitude x_i for the i-th atom

$$y_i = G(x_i|\vartheta) = \frac{1}{(1+\vartheta e^{-x_i})} < 1$$

(Equation 30)

,where $\vartheta = 0.05$. To remove the effect of scaling on the normalized eigenvector as the number of atom increase, the output magnitudes are further centered at $\tilde{y}_i =: \frac{y_i - y^-}{y^+ - y^-} \leq 1$, where (y^-, y^+) are the minimum and maximum of $y_i \forall i$.

Data Availability

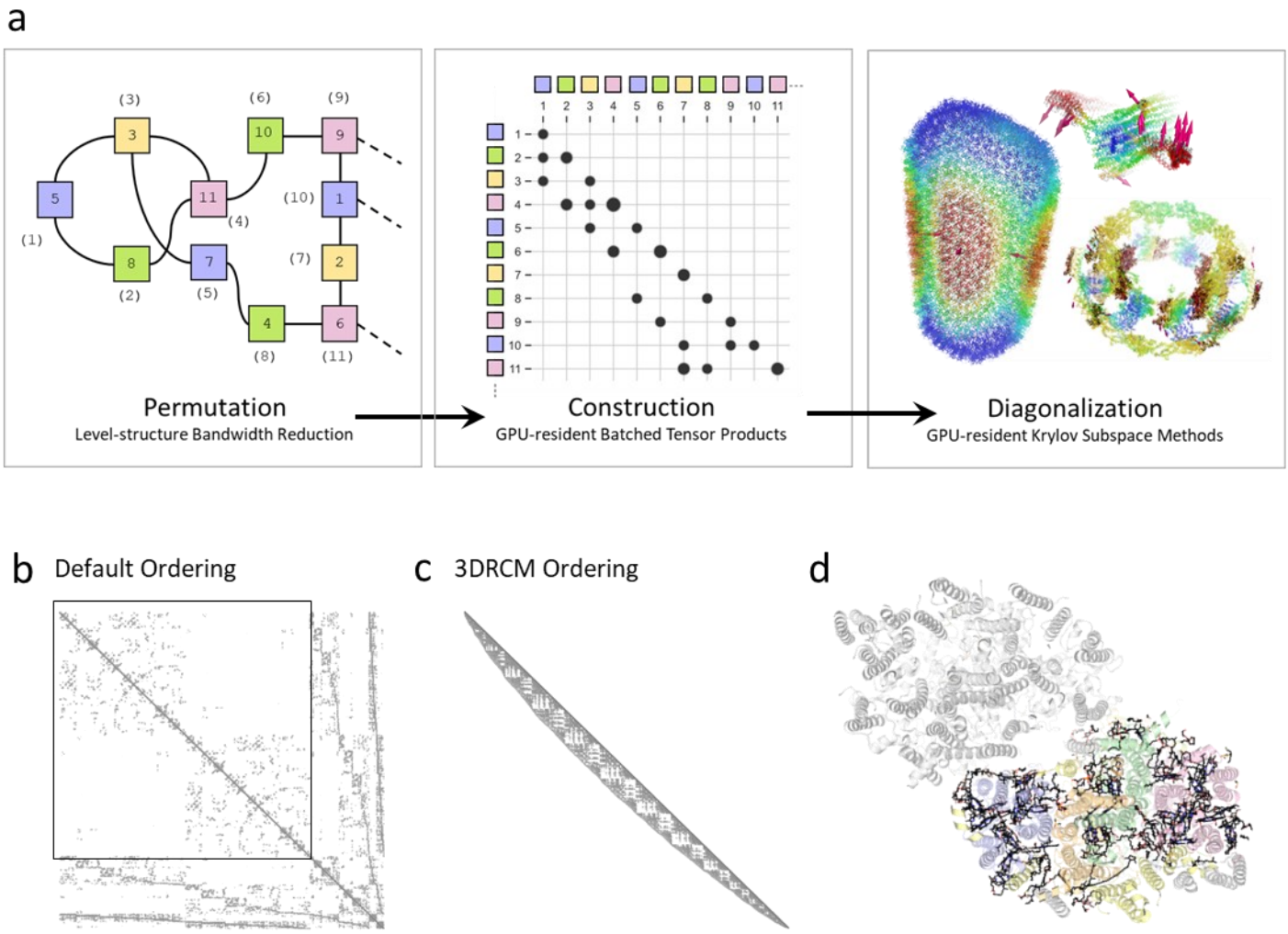
All atomic coordinates of macromolecules in the benchmark dataset were accessed from the Protein Data Bank. The benchmark dataset, including the atomic coordinates and supplementary data files on the calculated eigenvalues, speed, accuracy from elastic network model can be downloaded from 10.5281/zenodo.8061217.

Code Availability

The INCHING code is available at https://github.com/jhmlam/Inching. Jupyter notebooks and Python scripts for the experiments and analyses presented in the paper are available. Frozen versions of the software and associated code for analysis are also available in the 10.5281/zenodo.10443729.

Acknowledgement

This study was supported by internal funding from USC Dornsife college to V.K., and an NSF grant, OAC-2118061 to A.N. The authors acknowledge the Center for Advanced Research Computing (CARC) at the University of Southern California for providing computing resources that have contributed to the research results reported within this publication. URL: https://carc.usc.edu

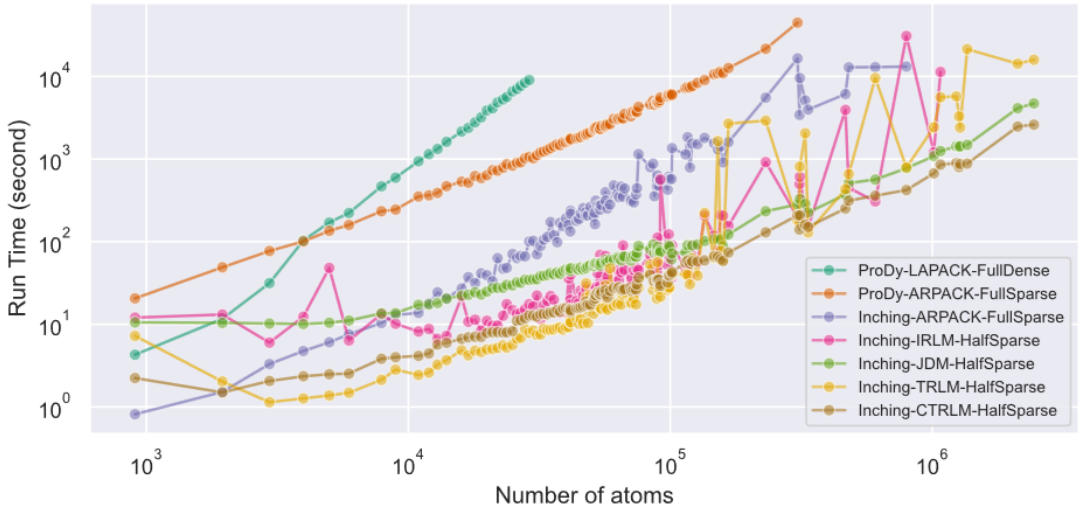


774

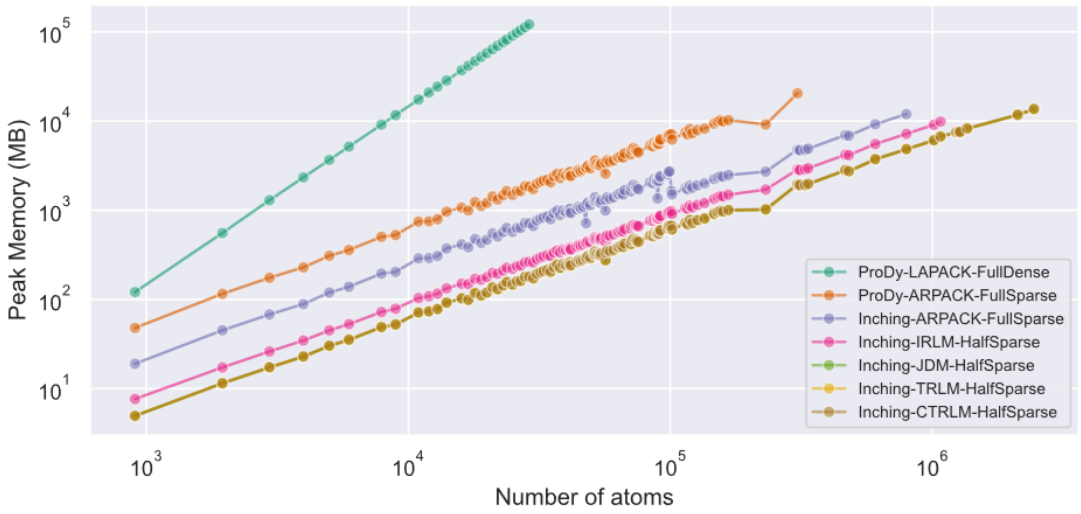
775 **Figure 1. Overview of the INCHING algorithm** (a) Our INCHING algorithm consists of three stages,
776 namely, permutation, construction, and diagonalization. In the permutation stage, 3DRCM takes atom
777 coordinates as the input and produce a bandwidth-reduced indexing as output. The initial atom ordering is
778 denoted within square-shaped nodes, while the 3DRCM-permuted atom ordering is represented using
779 parentheses. In the construction stage, the lower triangle of a Hessian with globally-sparse-yet-locally-
780 dense pattern is constructed incrementally on GPU with tensor products and broadcasts. In the
781 diagonalization stage, the eigenproblem is solved on GPU to produce depictable mode shapes. (b)
782 Sparsity pattern of the Hessian with default atom ordering for the PsbM-deletion mutant of photosystem

II. The top left square highlighted in correspond to the proteins in the macromolecular complex, which is also comprises cofactors, lipids and water molecules. (c) Sparsity pattern of the Hessian lower triangle with 3DRCM permuted atom ordering for the same complex (d) Overall spatial distribution of the chemicals intercalated in multiple protein chains in default sequential order (green, blue, pink, orange, yellow). Only one of the dimeric halves of the structure is colored, the rest of the macromolecule is shown as a grey surface; note that the complex is asymmetric due to slight difference in lipids and cofactors intercalated. The Hessian matrix in default ordering has a much longer bandwidth everywhere than that with the 3DRCM ordering.

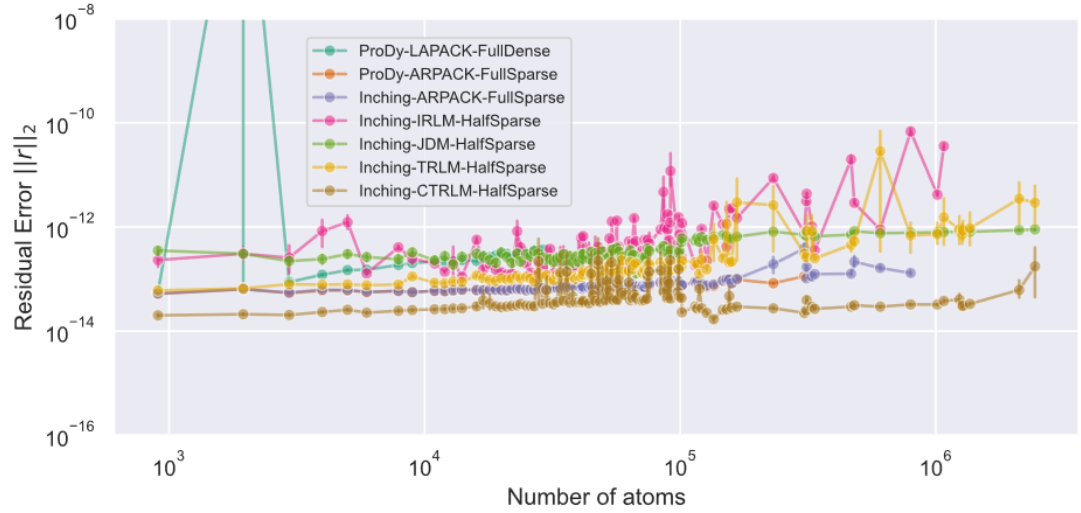
a


Run Time (second)
10^4
10^3
10^2
10^1
10^0
10^3
10^4
10^5
10^6
Number of atoms
ProDy-LAPACK-FullDense
ProDy-ARPACK-FullSparse
Inching-ARPACK-FullSparse
Inching-IRLM-HalfSparse
Inching-JDM-HalfSparse
Inching-TRLM-HalfSparse
Inching-CTRLM-HalfSparse

b

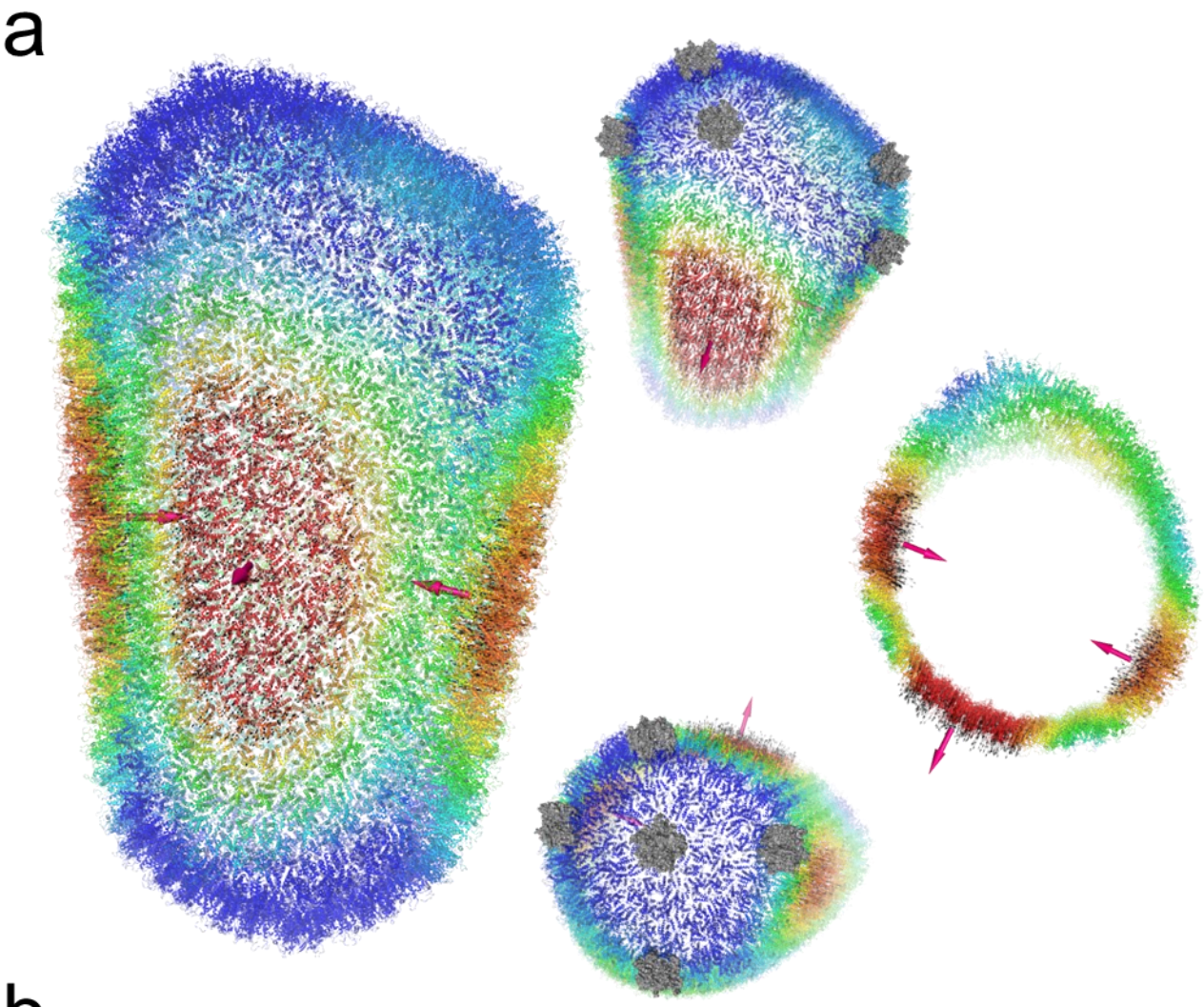

Peak Memory (MB)
10^5
10^4
10^3
10^2
10^1
10^3
10^4
10^5
10^6
Number of atoms
ProDy-LAPACK-FullDense
ProDy-ARPACK-FullSparse
Inching-ARPACK-FullSparse
Inching-IRLM-HalfSparse
Inching-JDM-HalfSparse
Inching-TRLM-HalfSparse
Inching-CTRLM-HalfSparse

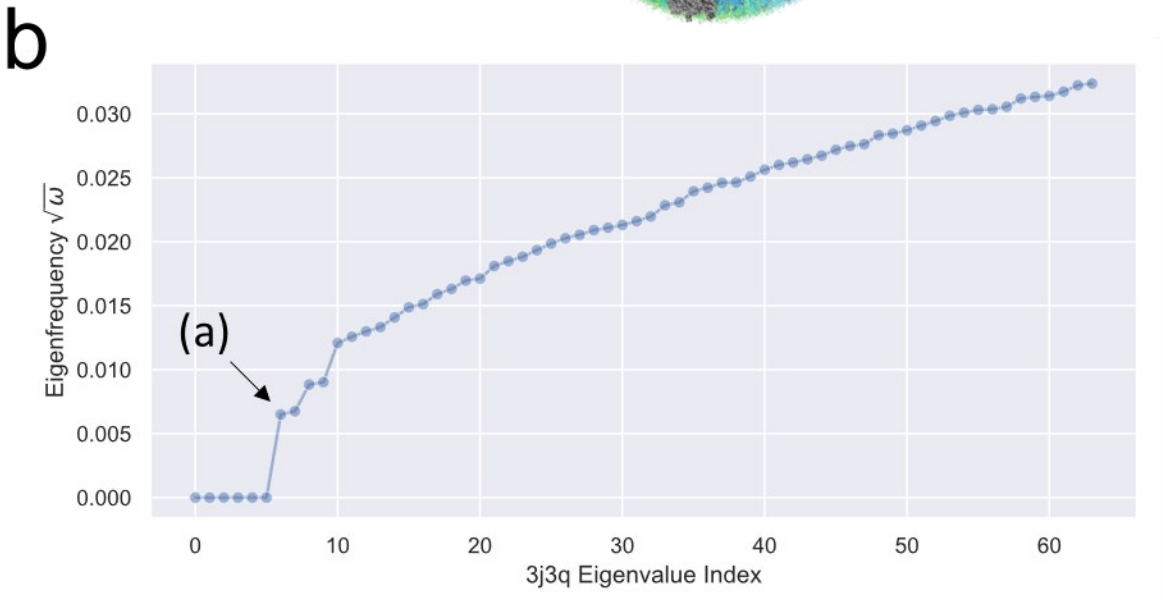
c


Residual Error $||r||_2$
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^3
10^4
10^5
10^6
Number of atoms
ProDy-LAPACK-FullDense
ProDy-ARPACK-FullSparse
Inching-ARPACK-FullSparse
Inching-IRLM-HalfSparse
Inching-JDM-HalfSparse
Inching-TRLM-HalfSparse
Inching-CTRLM-HalfSparse

791

Figure 2. Benchmark on throughput, memory consumption and correctness for each macromolecular structure in the benchmark dataset. For all methods, the first 64 eigenpairs were calculated. Methods tested includes 2 ProDy methods (i.e., "ProDy-LAPACK-FullDense" and "ProDy-ARPACK-FullSparse") and 5 INCHING methods (i.e., "INCHING-TRLM-HalfSparse", "INCHING-CTRLM-HalfSparse", "INCHING-JDM-HalfSparse", "INCHING-IRLM-HalfSparse", "INCHING-ARPACK-FullSparse"). Text in the name describe the eigensolver (e.g., ARPACK, LAPACK, JDM, TRLM, CTRLM, IRLM), the access of the Hessian matrix (e.g., "Full" means that the whole Hessian matrix is accessed and stored; "Half" means that only the lower triangle of the Hessian matrix is accessed and stored.), and the storage format of the Hessian matrix (e.g., "Dense" means a 2-D array is stored in double precision; "Sparse" means a CSR format is stored in double precision). "TRLM", "CTRLM", "JDM", "IRLM" are our implementation of the Thick Restart Lanczos Method (TRLM), Chebyshev-filtered Thick Restart Lanczos Method (CTRLM), Jacobi-Davidson Method (JDM) and Implicitly Restarted Lanczos Method (IRLM). (a) Benchmark on overall run time including Hessian realization and subsequent diagonalization. The run time is wall-clock time to complete all the calculation. (b) Benchmark on peak memory consumption. Note that for "ProDy-LAPACK-FullDense", we were not able to proceed once there are more than 29 thousand atoms due to memory overflow. Note that for "ProDy-ARPACK-FullSparse", we were not able to proceed once there are more than 305 thousand atoms as it takes more than 48 hours to converge. Also note that "INCHING-TRLM-HalfSparse", "INCHING-CTRLM-HalfSparse" and "INCHING-JDM-HalfSparse" share very similar peak memory requirement. (c) Benchmark on residual error. The error bar presented is the 95% confidence interval calculated from the residual error of all the eigenvalues of a macromolecular structure in the benchmark dataset. Note that for "ProDy-LAPACK-FullDense", the second macromolecular structure (PDBID: 1A8L) were not able to converge within 10^{-10} likely due to severe fill-ins. All programs were stopped if run time exceeds 48 hours. All INCHING programs were stopped if number of restarts exceeded 15000 rounds.


a


b
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Eigenfrequency $\sqrt{\omega}$
(a)
0
10
20
30
40
50
60
3j3q Eigenvalue Index

817

Figure 3. Vibrations of the mature HIV-1 capsid structure. (a) The first non-zero mode of the capsid. The black arrows are the displacement field of 1000 atoms randomly chosen from those in the top 90% quantile of vibration magnitude in the eigenvector. The arrow in magenta indicates an average direction for local clusters of the displacement field. Color scale in the cartoon from blue to red indicates increasing magnitude of vibration. Note that a logistic kernel is applied to the eigenvector to control the magnitude in visualization. (See Methods). The top-right inset is a clipped view of the capsid. The bottom-right inset indicates the locations of the pentamers (Grey opaque surface). (b) The first 64 eigenfrequency of the macromolecule. The black arrow indicates the first non-zero eigenmode displayed in (a).

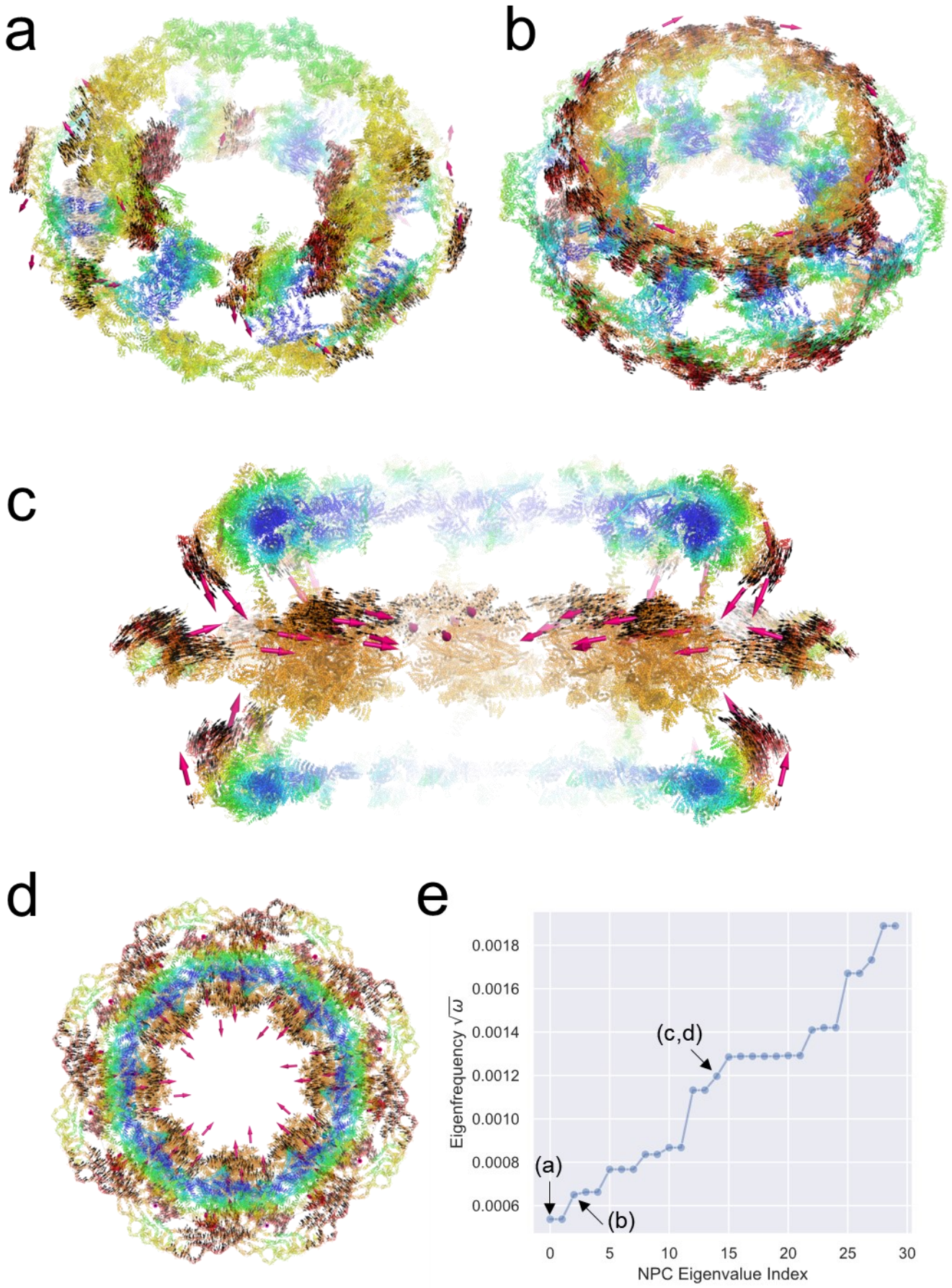
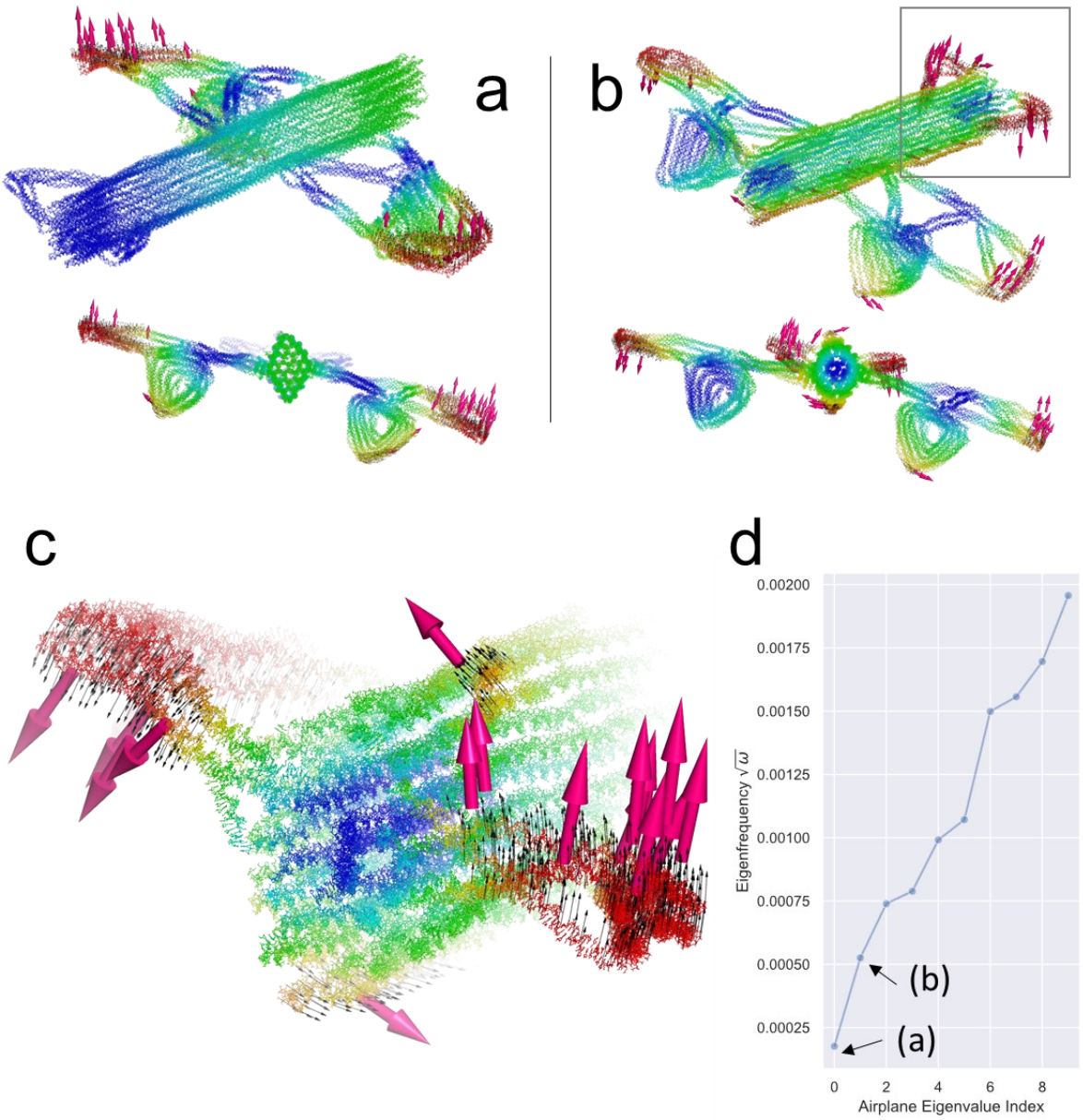

a
b
c
d
e
0.0018
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
Eigenfrequency $\sqrt{\omega}$
(c,d)
(a)
(b)
0
5
10
15
20
25
30
NPC Eigenvalue Index

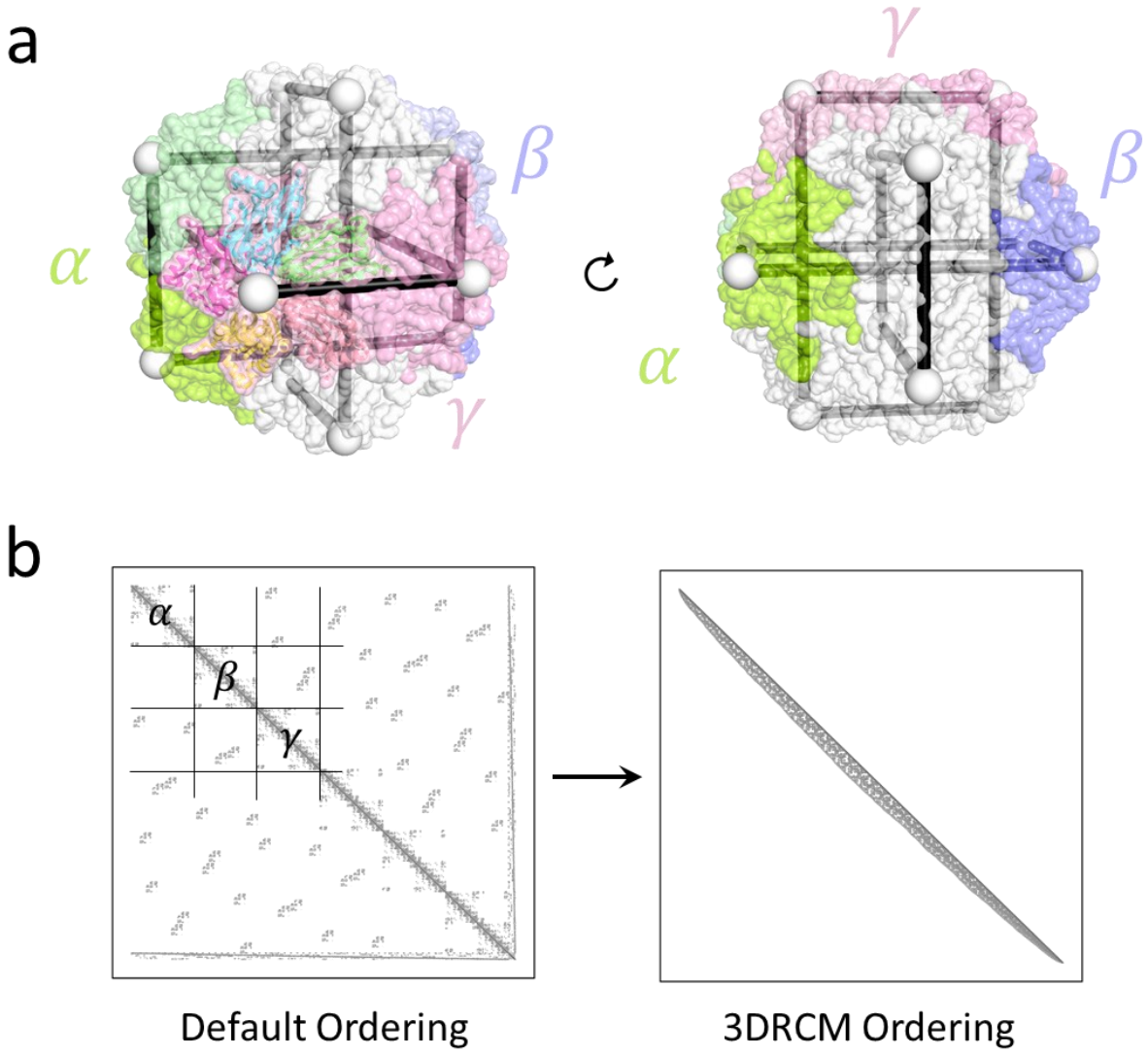
Figure 4. Vibrations of the dilated human Nuclear Pore Complex. Overview of motions in several representative non-zero modes. (a) The first non-zero normal mode (indexed as Mode 0) (b) the third non-zero normal mode (indexed as Mode 2) (c) Clipped view of the fifteenth non-zero normal mode (indexed as Mode 14) (d) Bird-eye view of the fifteenth non-zero normal mode (indexed as Mode 14) The arrows in magenta indicate an average directions for the local clusters of the displacement field, while the tiny black arrows show more detailed displacement fields of 10000 atoms randomly chosen from those in the top 80% quantile of vibration magnitude in the eigenvector. Color scale in the cartoon from blue to red indicates increasing magnitude of vibration. Note that a logistic kernel is applied to the eigenvector to control the magnitude in visualization. (See Methods). (e) The first 30 non-zero eigenfrequencies of the macromolecule. The black arrows indicate the first non-zero eigenmode displayed in (a), (b), (c) and (d).



841

842 **Figure 5. Vibrations of a DNA origami airplane.** Overview of motions in the first two non-zero modes
843 of the airplane. (a) The first non-zero eigenmode. (b) The second non-zero eigenmode; the square border
844 indicates the stabilizer, which is also shown in (c). Color scale in the cartoon from blue to red indicates
845 increasing magnitude of vibration. Note that a logistic kernel is applied to the eigenvector to control the
846 magnitude in visualization. (See Methods). The bottom inset show the airplane along its apparent bilateral
847 symmetry axis. (c) A zoom into the motion of the stabilizer in the second non-zero mode. The black
848 arrows are the displacement field of 1000 atoms randomly chosen from those in the top 90% quantile of

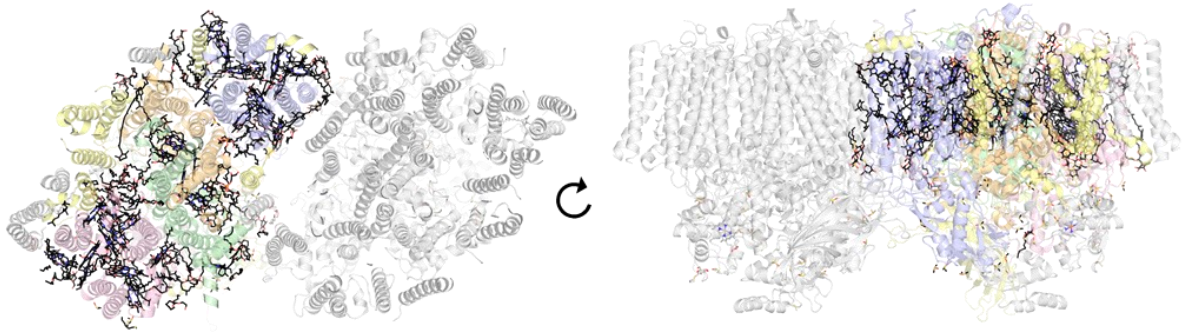
vibration magnitude in the eigenvector. The arrow in magenta indicates an average direction for local clusters of the displacement field, which also correspond to (a). (d) The first 10 non-zero eigenfrequencies of the macromolecule. Note that explicit external deflation were applied to remove the rigid modes.


a
α
β
γ
γ
β
α
b
α
β
γ
Default Ordering
3DRCM Ordering
c
Eigenfrequency √ω
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50
60
4y5z Eigenvalue Index

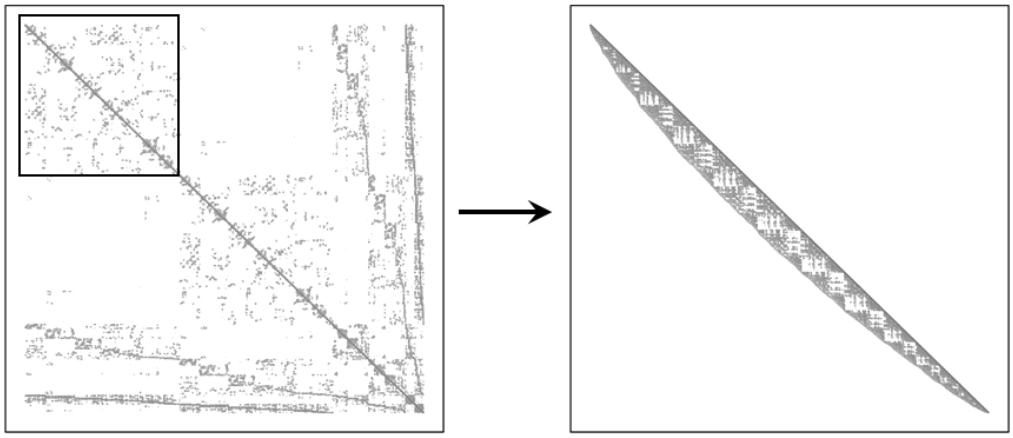
852

Extended Data Figure 1. Layout of the Hessian Matrix with and without permutation of atom ordering by 3DRCM for a case study on the chimeric Sesbania mosaic virus coat protein (PDBID: 4Y5Z). The protein structure is a homo-60-mer made of 12 sets of pentamers arranged as an icosahedron with apparent symmetry. (a) Overall spatial distribution of the pentamers in default sequential order. Segment α, β , and γ were colored in green, blue and pink respectively. Segment α consist of chain A,B,C,D,E,F,G,H,I and J. Segment β consist of chain K,L,M,N,O,P,Q,R,S and T. Segment γ consist of chain U, V, W, X, Y, Z, a, c, e and g. The three orthogonal golden rectangles were shown in black with vertices in white sphere. The rest of the macromolecule is shown as a grey surface. A pentamer is displayed as cartoons in segment γ. (b) The Hessian matrix in default ordering (left) and the lower triangle of the Hessian matrix in 3DRCM ordering (right). The Hessian matrix in default ordering has a much longer bandwidth everywhere than the Hessian matrix in 3DRCM ordering; the last few rows and columns in the default ordering refers to water and ions. In the default ordering, it is also clear that segment α do not interact with segment β, but segment γ interacts with both α and β. (c) The first 64 eigenfrequency of the macromolecule. Note that the macromolecule is asymmetric due to the inconsistent presence of waters. Inset shows the mode shapes from two apparently degenerate modes. The black arrows are the displacement field of 1000 atoms randomly chosen from those in the top 90% quantile of vibration magnitude in the eigenvector. The arrow in magenta indicates an average direction for local clusters of the displacement field. The vertices belonging to the same golden rectangle is displayed in the same color.

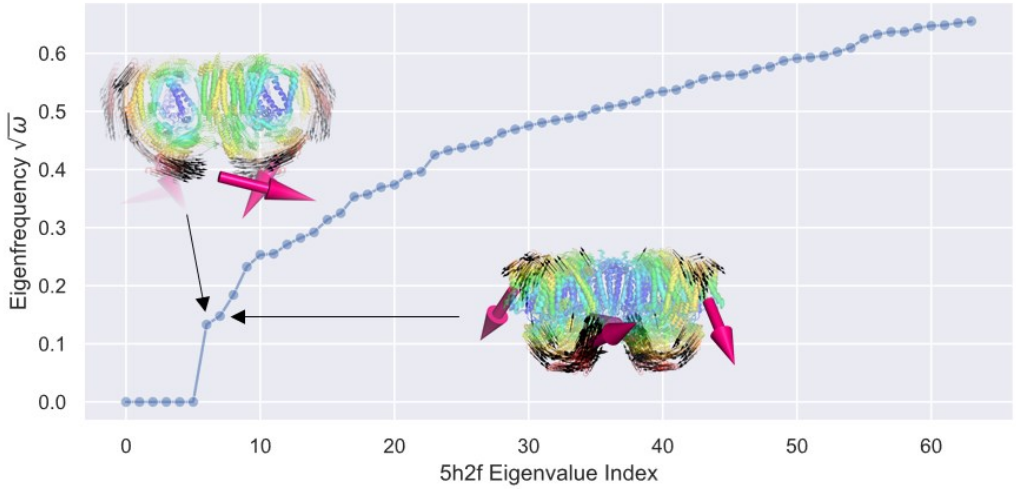
a



b

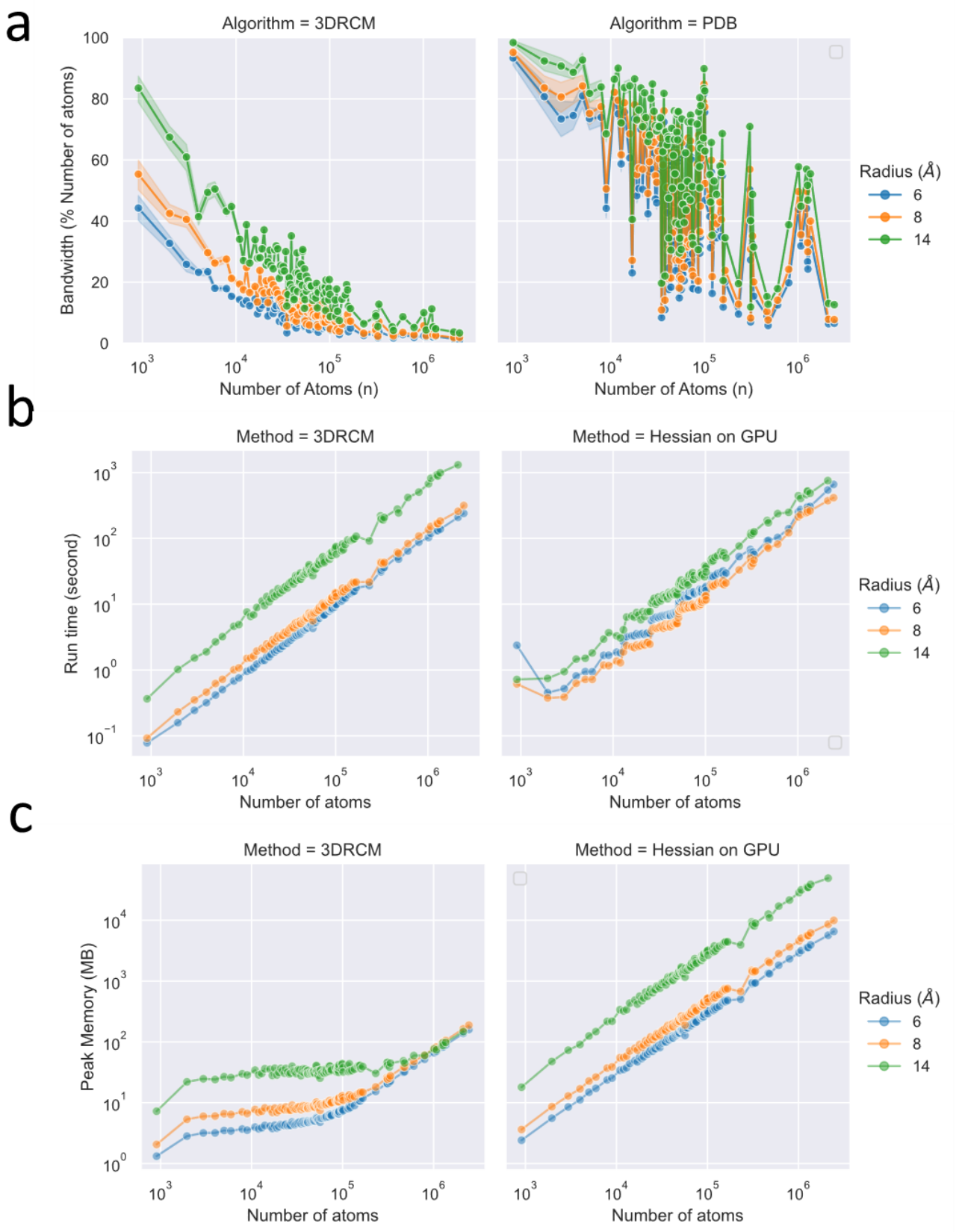

Default Ordering
3DRCM Ordering

c


Eigenfrequency $\sqrt{\omega}$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
5h2f Eigenvalue Index

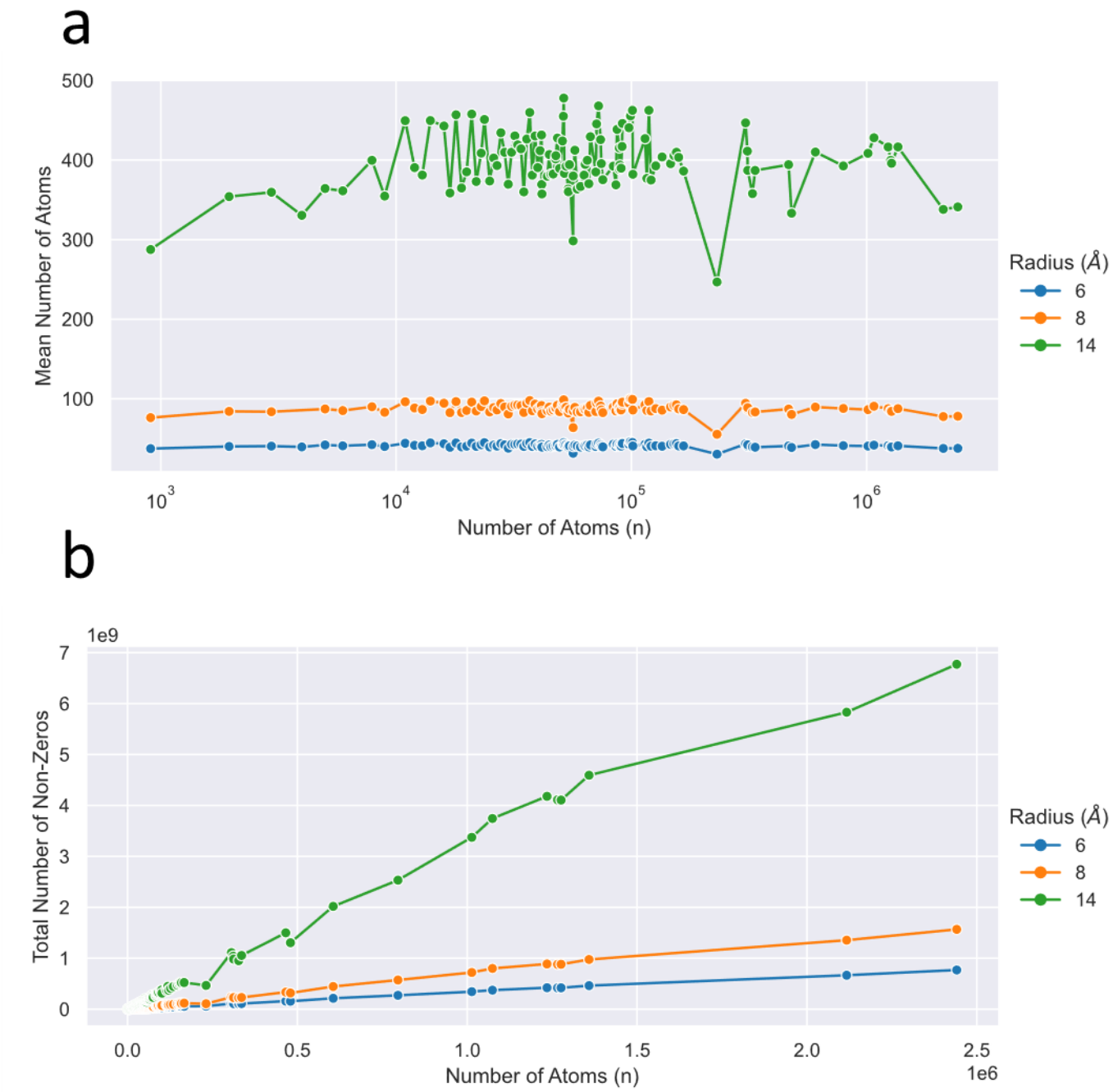
872

Extended Data Figure 2. Layout of the Hessian Matrix with and without permutation of atom ordering by 3DRCM for a case study on the PsbM-deletion mutant of photosystem II (PDBID: 5H2F). The macromolecular complex is made of cofactors, lipids, water and protein chains. (a) Overall spatial distribution of the chemicals intercalated in multiple protein chains in default sequential order (green, blue, pink, orange, yellow). Chemicals were shown in black. Only the right half of the structure is colored, the rest of the macromolecule is shown as a grey surface. (b) The Hessian matrix in default ordering (left) and the lower triangle of the Hessian matrix in 3DRCM of the Hessian matrix in 3DRCM ordering (right). The Hessian matrix in default ordering has a much longer bandwidth everywhere than the 3DRCM ordering. The top left square highlighted in the Hessian matrix of the default ordering correspond to that of the right half proteins. Note that the macromolecule is asymmetric due to slight difference in chemicals intercalated. (c) The first 64 eigenfrequencies of the macromolecule. Inset shows the stacked mode shapes from each of the first two modes. The black arrows are the displacement field of 1000 atoms randomly chosen from those in the top 90% quantile of vibration magnitude in the eigenvector. The arrow in magenta indicates an average direction for local clusters of the displacement field.

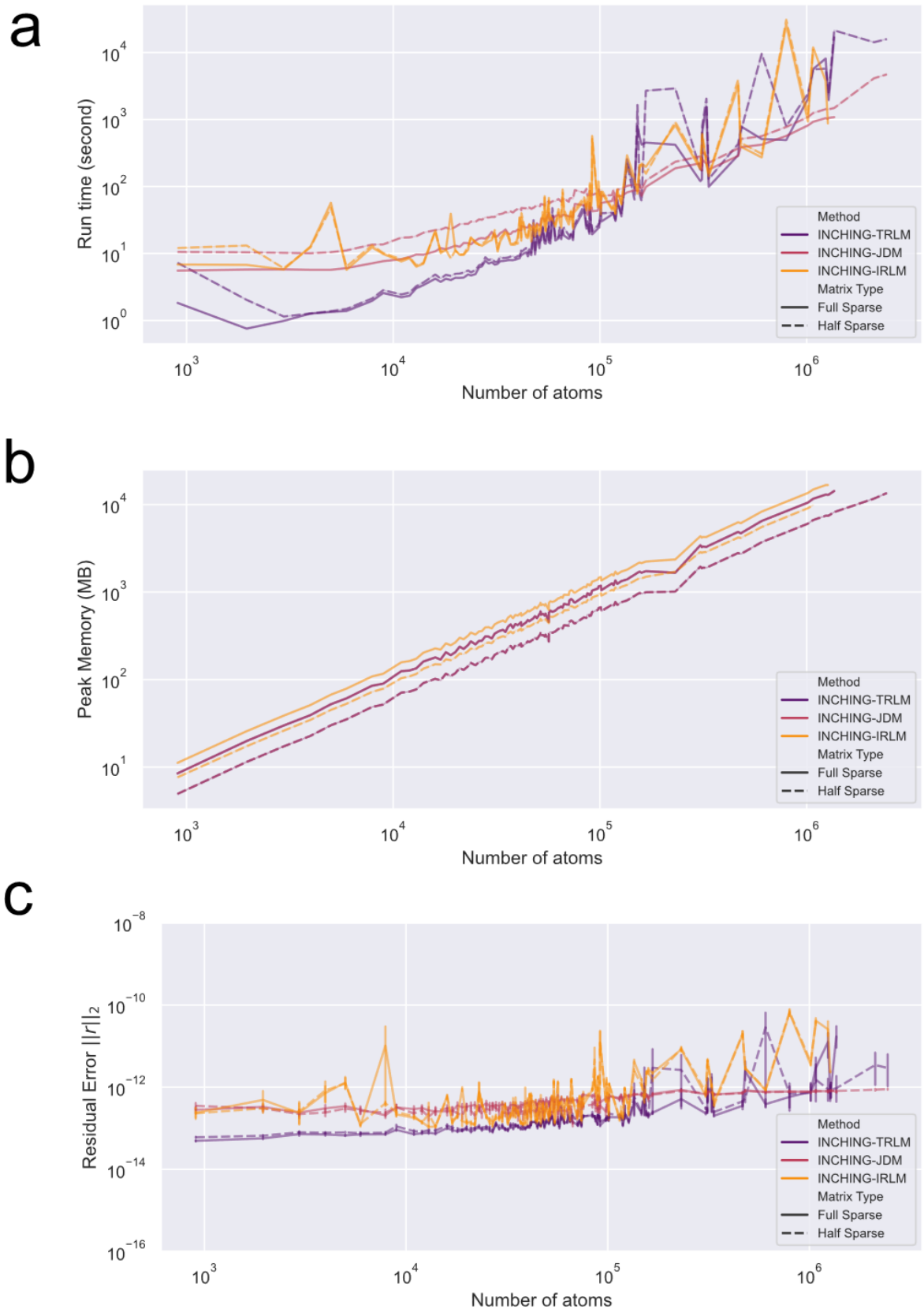

a
Algorithm = 3DRCM
Algorithm = PDB
Bandwidth (% Number of atoms)
Number of Atoms (n)
Radius (Å)
6
8
14
b
Method = 3DRCM
Method = Hessian on GPU
Run time (second)
Number of atoms
c
Peak Memory (MB)

888

Extended Data Figure 3. Benchmarks on permutation and construction of the Hessian on GPU. Color code refers to radius R_C = 6 Å (blue),8 Å (orange) or 14 Å (green). (a) Mean bandwidth of a 100-atoms batch of Hessian in default PDB ordering and 3DRCM ordering. To facilitate communication, the mean bandwidth reported here is the mean bandwidth in terms of atom slice (i.e., $N\times N$) of the ($N\times N\times 3\times 3$) Hessian tensor. The mean bandwidth is presented as the percentage of the total number of atoms N. The Hessian realized with 3DRCM permuted atom ordering is consistently less than 10% of N once there are more than 100 thousand atoms, whereas in the default PDB ordering it can fluctuate between 10% and 90% of N. The shade is the 95% confidence interval. (b) Run time of the algorithms. The reported time is wall-clock time. (c) Peak Memory of the algorithms. Note that for 3DRCM, the memory reported is in random access memory, whereas for the Hessian on GPU, the memory reported is the GPU memory. For R_C = 14 Å, 64-bit indexing were used throughout instead of 32-bit indexing to accommodate the larger amount of non-zero entries.

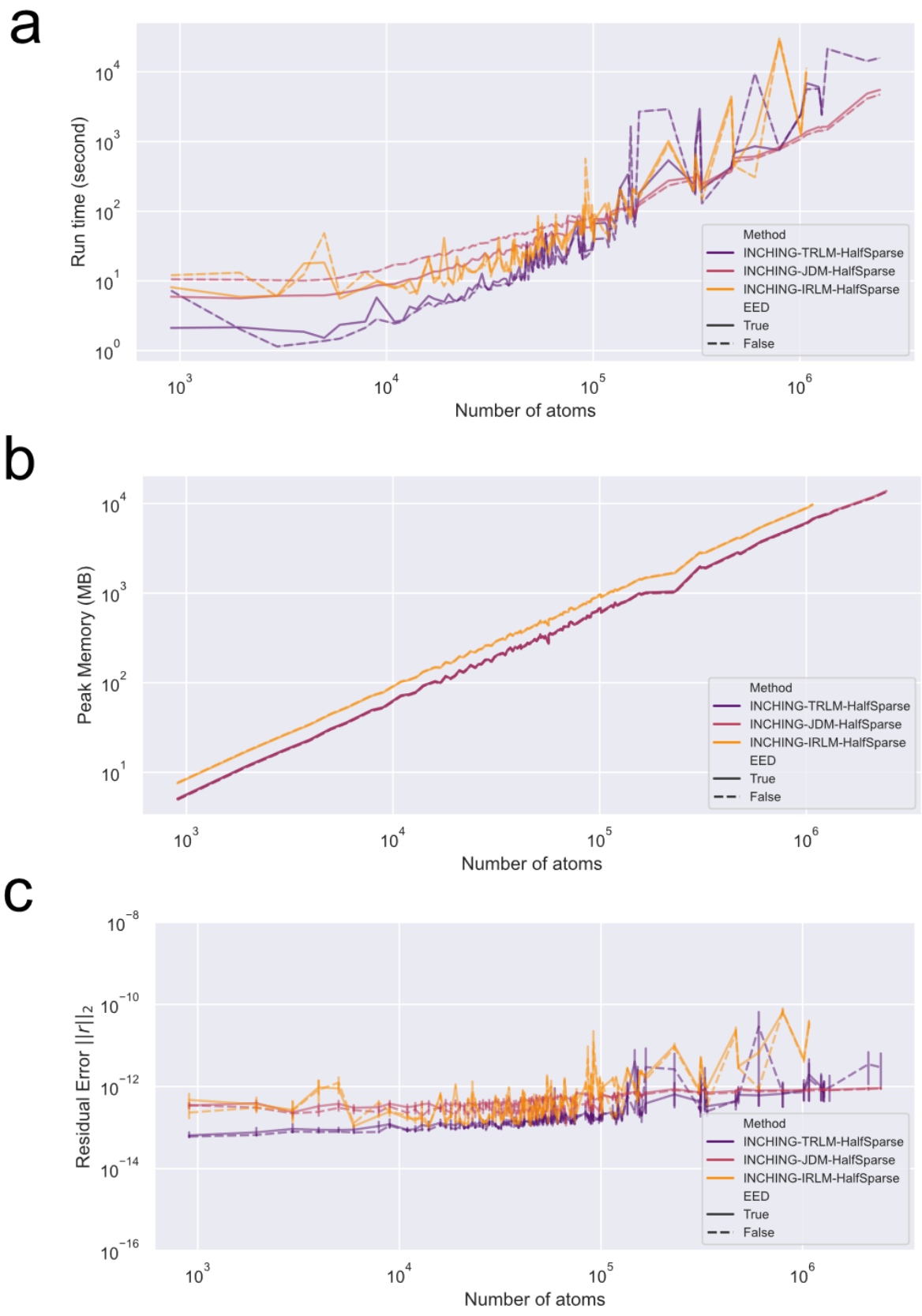


Extended Data Figure 4. Information on the benchmark dataset. The benchmark dataset has 116 macromolecules with size ranging from around 1000 atoms to around 2.4 million atoms. Color code refers to radius R_C = 6 Å (blue),8 Å (orange) or 14 Å (green). (a) Mean number of atoms within a sphere of radius R_C. (b) Total number of non-zero entries in the full Hessian matrix.


a
Run time (second)
10^4
10^3
10^2
10^1
10^0
10^3
10^4
10^5
10^6
Number of atoms
Method
INCHING-TRLM
INCHING-JDM
INCHING-IRLM
Matrix Type
Full Sparse
Half Sparse
b
Peak Memory (MB)
10^4
10^3
10^2
10^1
10^3
10^4
10^5
10^6
Number of atoms
Method
INCHING-TRLM
INCHING-JDM
INCHING-IRLM
Matrix Type
Full Sparse
Half Sparse
c
Residual Error $||r||_2$
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^3
10^4
10^5
10^6
Number of atoms
Method
INCHING-TRLM
INCHING-JDM
INCHING-IRLM
Matrix Type
Full Sparse
Half Sparse

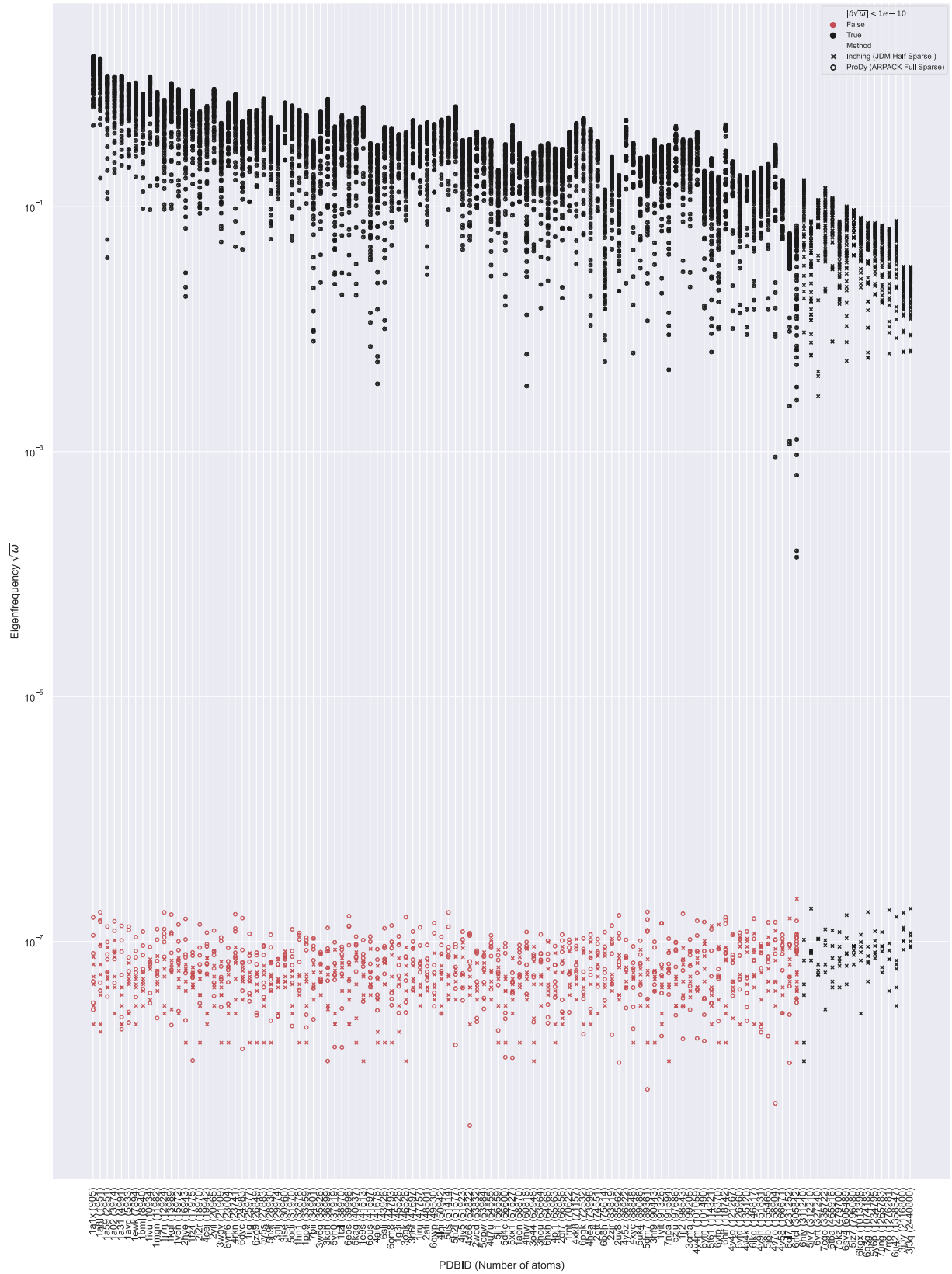
906

Extended Data Figure 5. Comparison of the effect of accessing and storing the full Hessian matrix rather than the lower triangle on throughput, memory consumption and correctness in the benchmark dataset. For all methods, the first 64 eigenpairs were calculated with radius R_C = 8 Å. Methods tested includes 3 INCHING methods (i.e. INCHING-TRLM, INCHING-JDM, INCHING-IRLM) with only the lower triangle of the Hessian stored and accessed in dash line and with the full Hessian matrix stored and accessed in solid line. The methods in dash line with only the lower triangle of the Hessian stored were also displayed in Figure 2 of main text. In all methods, the storage format of the Hessian matrix is always "Sparse" meaning a CSR format is stored in double precision. INCHING-TRLM, INCHING-JDM, INCHING-IRLM are our implementation of the Thick Restart Lanczos Method (TRLM), Jacobi-Davidson Method (JDM) and Implicitly Restarted Lanczos Method (IRLM). (a) Benchmark on overall run time including Hessian realization and subsequent diagonalization. The run time is wall-clock time to complete all the calculation. (b) Benchmark on peak memory consumption. Also note that INCHING-TRLM-HalfSparse and INCHING-JDM-HalfSparse share very similar peak memory requirement. (c) Benchmark on residual error. The error bar presented is the 95% confidence interval calculated from the residual error of all the eigenvalues of a macromolecular structure in the benchmark dataset. All programs were stopped if run time exceeds 48 hours. All INCHING programs were stopped if number of restarts exceeded 15000 rounds.

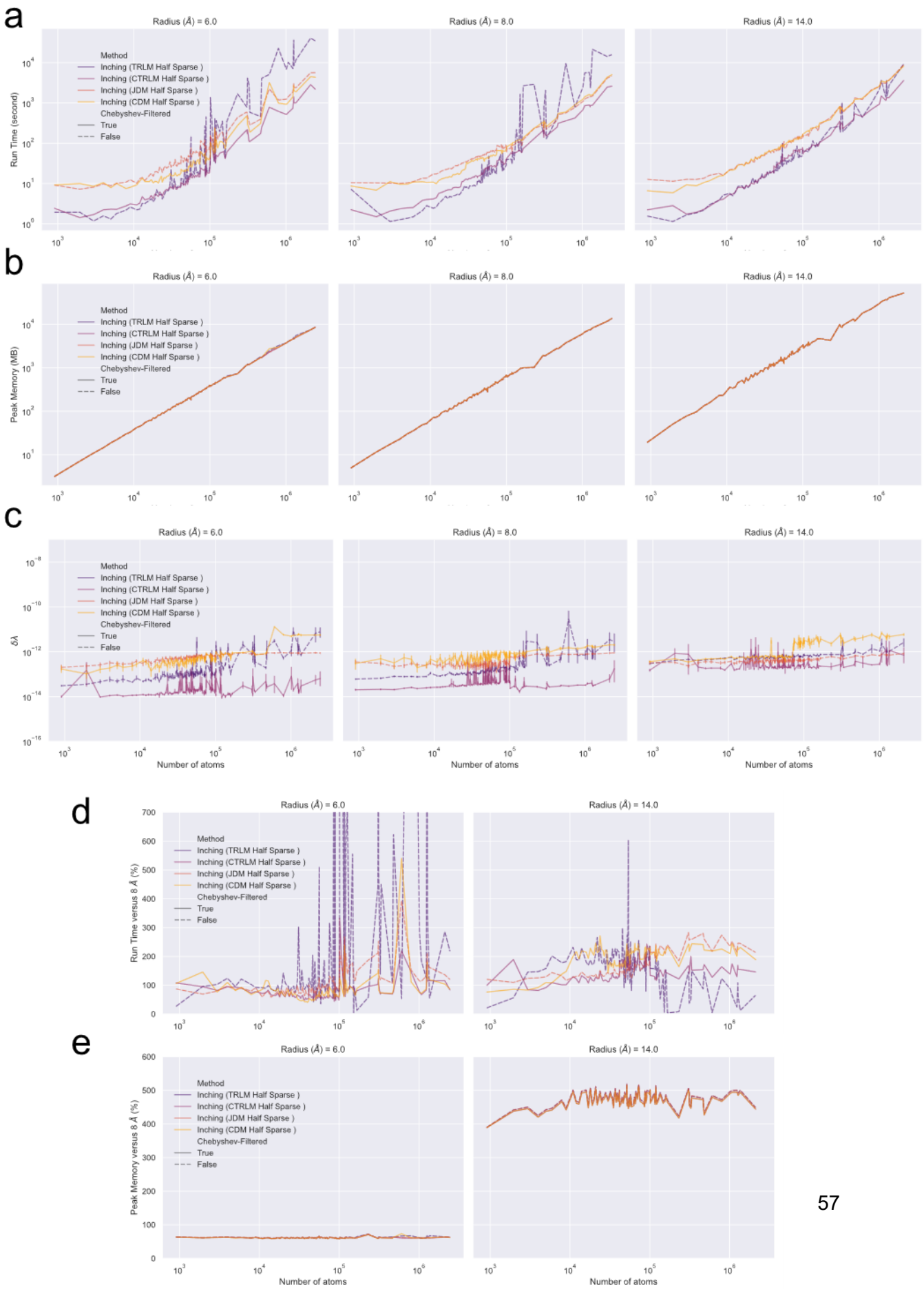

a
Run time (second)
Number of atoms
b
Peak Memory (MB)
Number of atoms
c
Residual Error ||r||2
Number of atoms
Method
INCHING-TRLM-HalfSparse
INCHING-JDM-HalfSparse
INCHING-IRLM-HalfSparse
EED
True
False

926

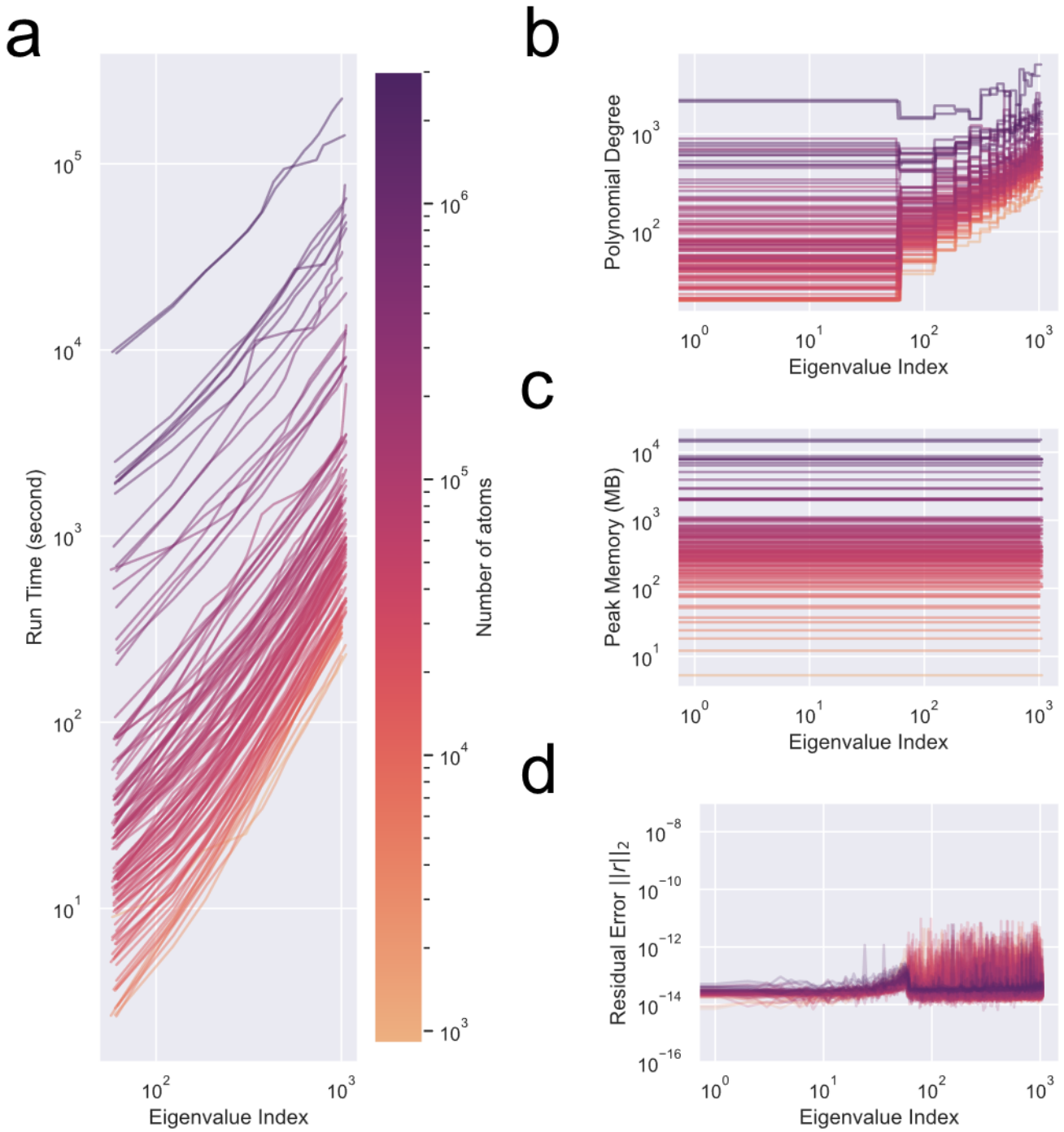
Extended Data Figure 6. Comparison of the effect of explicit external deflation (EED) on throughput, memory consumption and correctness in the benchmark dataset. For all methods, the first 64 eigenpairs were calculated with radius R_C = 8 Å. Methods tested includes 3 INCHING methods (i.e. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-IRLM-HalfSparse without EED in dash line and with EED in solid line. The methods in dash line (i.e. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-IRLM-HalfSparse without EED were also displayed in Figure 2 of main text. In all methods, "Half" means only the lower triangle of the Hessian matrix is accessed and stored and the storage format of the Hessian matrix is always "Sparse" meaning a CSR format is stored in double precision. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-IRLM-HalfSparse are our implementation of the Thick Restart Lanczos Method (TRLM), Jacobi-Davidson Method (JDM) and Implicitly Restarted Lanczos Method (IRLM). (a) Benchmark on overall run time including Hessian realization and subsequent diagonalization. The run time is wall-clock time to complete all the calculation. (b) Benchmark on peak memory consumption. Also note that INCHING-TRLM-HalfSparse and INCHING-JDM-HalfSparse share very similar peak memory requirement. (c) Benchmark on residual error. The error bar presented is the 95% confidence interval calculated from the residual error of all the eigenvalues of a macromolecular structure in the benchmark dataset. All programs were stopped if run time exceeds 48 hours. All INCHING programs were stopped if number of restarts exceeded 15000 rounds.


$|\delta\sqrt{\omega}| < 1e-10$
False
True
Method
Inching (JDM Half Sparse)
ProDy (ARPACK Full Sparse)
Eigenfrequency $\sqrt{\omega}$
10^{-1}
10^{-3}
10^{-5}
10^{-7}
PDBID (Number of atoms)

Extended Data Figure 7. Comparing Eigenfrequencies calculated. Eigenfrequencies computed with INCHING-JDM-HalfSparse in cross marker, compared to those from ProDy with ARPACK backend, i.e. ProDy-ARPACK-FullSparse in circle marker. The ARPACK package is a golden standard implementation of the Implicitly Restarted Lanczos Method (IRLM). Whenever the eigenvalue of the same index calculated by INCHING-JDM-HalfSparse differs from ProDy-ARPACK-FullSparse with absolute value larger than 10^{-10}, the difference is highlighted in red. For cases where ProDy-ARPACK FullSparse did not converge within 48 hours, there is no datapoint. In general, eigenfrequencies less than 10^{-6} are considered rigid modes with eigenvalue zero. In all cases considered, there are only 6 rigid modes. For calculation of eigenfrequencies, absolute value was applied to all eigenvalues to avoid spurious imaginary eigenfrequencies for rigid modes with small negative numerical eigenvalue, e.g., -10^{-14}. For all methods, the first 64 eigenpairs were calculated with radius R_C = 8 Å.

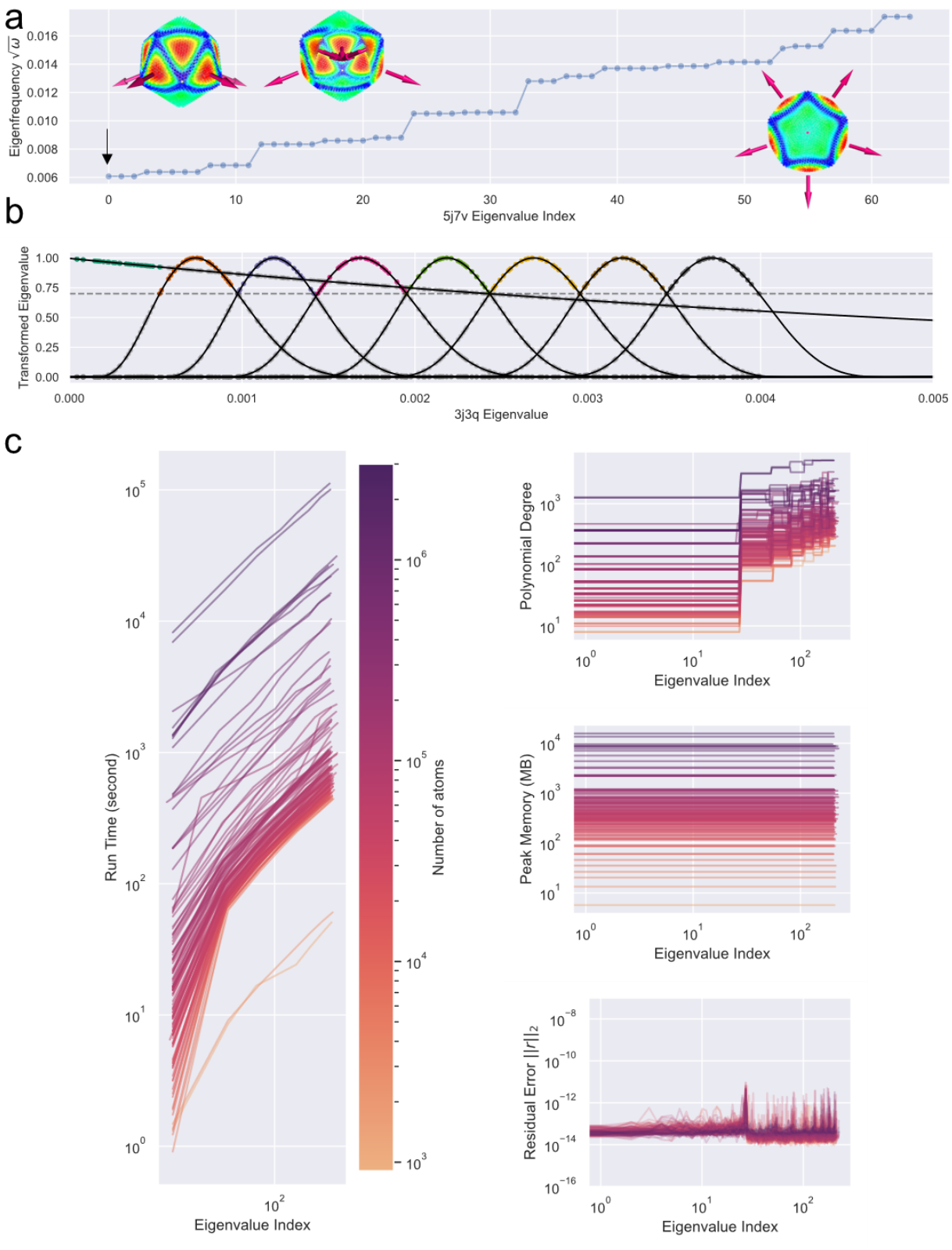

a
Radius (Å) = 6.0
Radius (Å) = 8.0
Radius (Å) = 14.0
Run Time (second)
Method
Inching (TRLM Half Sparse)
Inching (CTRLM Half Sparse)
Inching (JDM Half Sparse)
Inching (CDM Half Sparse)
Chebyshev-Filtered
True
False
b
Peak Memory (MB)
c
δλ
Number of atoms
d
Run Time versus 8 Å (%)
e
Peak Memory versus 8 Å (%)

Extended Data Figure 8. Scaling in cutoff radii and effect of Chebyshev filtering on throughput, memory consumption and correctness in the benchmark dataset. For all methods, the first 64 eigenpairs were calculated. Methods tested includes 4 INCHING methods (i.e. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-CTRLM-HalfSparse and INCHING-CDM-HalfSparse. Methods with Chebyshev low-pass filtering in solid line, otherwise dashline. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse and INCHING-CTRLM-HalfSparse were also displayed in Figure 2 of main text. In all methods, "Half" means only the lower triangle of the Hessian matrix is accessed and stored and the storage format of the Hessian matrix is always "Sparse" meaning a CSR format is stored in double precision. INCHING-TRLM-HalfSparse, INCHING-JDM-HalfSparse, INCHING-CTRLM-HalfSparse and INCHING-CDM-HalfSparse are our implementation of the Thick Restart Lanczos Method (TRLM), Jacobi-Davidson Method (JDM), Chebshev-filtered Thick Restart Lanczos Method (CTRLM) and Chebyshev-Davidson Method (CDM). Three cutoff radii were considered including R_C = 6 Å,8 Å and 14 Å. (a) Benchmark on overall run time including Hessian realization and subsequent diagonalization. The run time is wall-clock time to complete all the calculation. (b) Benchmark on peak memory consumption. (c) Benchmark on residual error. The error bar presented is the 95% confidence interval calculated from the residual error of all the eigenvalues of a macromolecular structure in the benchmark dataset. Comparison of 6 Å and 14 Å calculations in the percentage of run time (d) and peak memory (e) versus standard 8 Å calculations. Note that comparison of scaling in radii in memory is dependent on the number and indexing of non-zero entries in Hessian, where for all 14 Å calculations, 64-bit indexing were used instead of 32-bit indexing to accommodate the growth in non-zero entries in Hessian beyond capability of 32-bit indexing.



Extended Data Figure 9. Scaling in number of eigenmodes in Chebyshev-filtered Thick Restart Lanczos Method. Up to 1000 or slightly more eigenpairs were calculated using our INCHING-CTRLM-HalfSparse implementation of the Chebyshev-filtered Thick Restart Lanczos Method with radius of

interaction R_C = 8Å. The coloring of the lines, as shown in the color bar, refers to the number of atoms in the system. (a) Run time to complete until the indexed eigenvalue (b) Optimized polynomial degrees involved in calculating the indexed eigenvalue. (c) Peak memory consumed when calculating the spectrum slice containing the indexed eigenvalue. (d) Residual error in each of the eigenvalues.


a
Eigenfrequency $\sqrt{\omega}$
0.016
0.014
0.012
0.010
0.008
0.006
0
10
20
30
40
50
60
5j7v Eigenvalue Index
b
Transformed Eigenvalue
1.00
0.75
0.50
0.25
0.00
0.000
0.001
0.002
0.003
0.004
0.005
3j3q Eigenvalue
c
Run Time (second)
Eigenvalue Index
Number of atoms
Polynomial Degree
Eigenvalue Index
Peak Memory (MB)
Eigenvalue Index
Residual Error $||r||_2$
Eigenvalue Index

988

Extended Data Figure 10. Results in applying Chebyshev-filtered Thick Restart Lanczos Method. (a) The first 64 eigenfrequency of the Faustovirus capsid (PDBID: 5J7V) coarse-grained as 5 million pseudoatoms. Inset shows the mode shapes of the first non-rigid mode. The left- and right-most insets are sideview and bird-eye view respectively; the middle inset is the sideview sectioned to show pattern inside. The arrow in magenta indicates an average direction for local clusters of the displacement field. (b) Mapping of the lowest 224 eigenvalues in batches of 28 eigenvectors when low- and band-pass Chebyshev filters were applied to the HIV capsid system (PDBID: 3J3Q) without coarse-graining. The calculation was done on a GPU RTX4090. Colors on the points indicate eigenvalues resolved in different batches of 64 eigenvalues. (c) Corresponding scaling in number of eigenmodes on GPU RTX4090 in batches of 28 eigenvectors. Up to 200 or slightly more eigenpairs were calculated using INCHING-CTRLM-HalfSparse with cutoff radius R_C = 8Å. The coloring of the lines, as shown in the color bar, refers to the number of atoms in the system. Run time to complete until the indexed eigenvalue, optimized polynomial degrees involved in calculating the indexed eigenvalue, peak memory consumed when calculating the spectrum slice containing the indexed eigenvalue and residual error in each of the eigenvalues were also shown.

Reference

1. Delarue, M. & Dumas, P. On the use of low-frequency normal modes to enforce collective movements in refining macromolecular structural models. *Proceedings of the National Academy of Sciences* **101**, 6957–6962 (2004).
2. Tama, F., Miyashita, O. & Brooks III, C. L. Normal mode based flexible fitting of high-resolution structure into low-resolution experimental data from cryo-EM. *Journal of Structural Biology* **147**, 315–326 (2004).

3. Gur, M., Madura, J. D. & Bahar, I. Global Transitions of Proteins Explored by a Multiscale Hybrid Methodology: Application to Adenylate Kinase. *Biophysical Journal* **105**, 1643–1652 (2013).
4. Franklin, J., Koehl, P., Doniach, S. & Delarue, M. MinActionPath: maximum likelihood trajectory for large-scale structural transitions in a coarse-grained locally harmonic energy landscape. *Nucleic Acids Research* **35**, W477–W482 (2007).
5. Bakan, A. & Bahar, I. The intrinsic dynamics of enzymes plays a dominant role in determining the structural changes induced upon inhibitor binding. *Proceedings of the National Academy of Sciences* **106**, 14349–14354 (2009).
6. Shrivastava, I. H. & Bahar, I. Common Mechanism of Pore Opening Shared by Five Different Potassium Channels. *Biophysical Journal* **90**, 3929–3940 (2006).
7. Lezon, T. R. & Bahar, I. Constraints Imposed by the Membrane Selectively Guide the Alternating Access Dynamics of the Glutamate Transporter GltPh. *Biophysical Journal* **102**, 1331–1340 (2012).
8. Bahar, I., Lezon, T. R., Bakan, A. & Shrivastava, I. H. Normal Mode Analysis of Biomolecular Structures: Functional Mechanisms of Membrane Proteins. *Chem. Rev.* **110**, 1463–1497 (2010).
9. Levitt, M. Conformation analysis of proteins. (University of Cambridge, 1972).
10. Levitt, M., Sander, C. & Stern, P. S. The normal modes of a protein: Native bovine pancreatic trypsin inhibitor. *International Journal of Quantum Chemistry* **24**, 181–199 (1983).
11. Brooks, B. & Karplus, M. Normal modes for specific motions of macromolecules: application to the hinge-bending mode of lysozyme. *Proceedings of the National Academy of Sciences* **82**, 4995–4999 (1985).
12. Brooks, B. & Karplus, M. Harmonic dynamics of proteins: normal modes and fluctuations in bovine pancreatic trypsin inhibitor. *Proc. Natl. Acad. Sci. U.S.A.* **80**, 6571–6575 (1983).

13. Ma, J. Usefulness and Limitations of Normal Mode Analysis in Modeling Dynamics of Biomolecular Complexes. *Structure* **13**, 373–380 (2005).
14. Hayward, S. & Groot, B. L. de. Normal Modes and Essential Dynamics. in *Molecular Modeling of Proteins* 89–106 (Humana Press, 2008). doi:10.1007/978-1-59745-177-2_5.
15. Tirion, M. M. Large Amplitude Elastic Motions in Proteins from a Single-Parameter, Atomic Analysis. *Phys. Rev. Lett.* **77**, 1905–1908 (1996).
16. Hinsen, K., Thomas, A. & Field, M. J. Analysis of domain motions in large proteins. *Proteins: Structure, Function, and Bioinformatics* **34**, 369–382 (1999).
17. Hinsen, K. Analysis of domain motions by approximate normal mode calculations. *Proteins: Structure, Function, and Bioinformatics* **33**, 417–429 (1998).
18. Tama, F. & Sanejouand, Y.-H. Conformational change of proteins arising from normal mode calculations. *Protein Engineering, Design and Selection* **14**, 1–6 (2001).
19. Brink, J. *et al.* Experimental Verification of Conformational Variation of Human Fatty Acid Synthase as Predicted by Normal Mode Analysis. *Structure* **12**, 185–191 (2004).
20. Schilbach, S. *et al.* Structures of transcription pre-initiation complex with TFIIH and Mediator. *Nature* **551**, 204–209 (2017).
21. Jin, Q. *et al.* Iterative Elastic 3D-to-2D Alignment Method Using Normal Modes for Studying Structural Dynamics of Large Macromolecular Complexes. *Structure* **22**, 496–506 (2014).
22. Krieger, J. M., Sorzano, C. O. S., Carazo, J. M. & Bahar, I. Protein dynamics developments for the large scale and cryoEM: case study of ProDy 2.0. *Acta Cryst D* **78**, 399–409 (2022).
23. Zhang, Y. *et al.* State-dependent sequential allostery exhibited by chaperonin TRiC/CCT revealed by network analysis of Cryo-EM maps. *Progress in Biophysics and Molecular Biology* **160**, 104–120 (2021).
24. Vuillemot, R., Miyashita, O., Tama, F., Rouiller, I. & Jonic, S. NMMD: Efficient Cryo-EM Flexible Fitting Based on Simultaneous Normal Mode and Molecular Dynamics atomic displacements. *Journal of Molecular Biology* **434**, 167483 (2022).

25. Atilgan, A. R. *et al.* Anisotropy of Fluctuation Dynamics of Proteins with an Elastic Network Model. *Biophysical Journal* **80**, 505–515 (2001).
26. Tama, F., Gadea, F. X., Marques, O. & Sanejouand, Y.-H. Building-block approach for determining low-frequency normal modes of macromolecules. *Proteins: Structure, Function, and Bioinformatics* **41**, 1–7 (2000).
27. Li, G. & Cui, Q. A Coarse-Grained Normal Mode Approach for Macromolecules:An Efficient Implementation and Application to Ca2+-ATPase. *Biophysical Journal* **83**, 2457–2474 (2002).
28. Perahia, D. & Mouawad, L. Computation of low-frequency normal modes in macromolecules: Improvements to the method of diagonalization in a mixed basis and application to hemoglobin. *Computers & Chemistry* **19**, 241–246 (1995).
29. Marques, O. & Sanejouand, Y.-H. Hinge-bending motion in citrate synthase arising from normal mode calculations. *Proteins: Structure, Function, and Bioinformatics* **23**, 557–560 (1995).
30. Koehl, P. Large Eigenvalue Problems in Coarse-Grained Dynamic Analyses of Supramolecular Systems. *J. Chem. Theory Comput.* **14**, 3903–3919 (2018).
31. Sleijpen, G. L. G. & Van der Vorst, H. A. A Jacobi--Davidson Iteration Method for Linear Eigenvalue Problems. *SIAM Rev.* **42**, 267–293 (2000).
32. Sleijpen, G. L. G., Booten, A. G. L., Fokkema, D. R. & van der Vorst, H. A. Jacobi-davidson type methods for generalized eigenproblems and polynomial eigenproblems. *Bit Numer Math* **36**, 595–633 (1996).
33. Wall, M. R. & Neuhauser, D. Extraction, through filter-diagonalization, of general quantum eigenvalues or classical normal mode frequencies from a small number of residues or a short-time segment of a signal. I. Theory and application to a quantum-dynamics model. *The Journal of Chemical Physics* **102**, 8011–8022 (1995).

34. Li, R., Xi, Y., Vecharynski, E., Yang, C. & Saad, Y. A Thick-Restart Lanczos Algorithm with Polynomial Filtering for Hermitian Eigenvalue Problems. *SIAM J. Sci. Comput.* **38**, A2512–A2534 (2016).
35. Zhou, Y. & Saad, Y. A Chebyshev–Davidson Algorithm for Large Symmetric Eigenproblems. *SIAM J. Matrix Anal. Appl.* **29**, 954–971 (2007).
36. Parlett, B. N. 5. Deflation. in *The Symmetric Eigenvalue Problem* 87–92 (Society for Industrial and Applied Mathematics, 1998). doi:10.1137/1.9781611971163.ch5.
37. Berman, H. M. *et al.* The Protein Data Bank. *Nucleic Acids Research* **28**, 235–242 (2000).
38. Zhao, G. *et al.* Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. *Nature* **497**, 643–646 (2013).
39. Townshend, R. J. L. *et al.* Geometric deep learning of RNA structure. *Science* **373**, 1047–1051 (2021).
40. Baek, M. *et al.* Accurate prediction of protein structures and interactions using a three-track neural network. *Science* **373**, 871–876 (2021).
41. Papadimitriou, Ch. H. The NP-Completeness of the bandwidth minimization problem. *Computing* **16**, 263–270 (1976).
42. Cuthill, E. & McKee, J. Reducing the bandwidth of sparse symmetric matrices. in *Proceedings of the 1969 24th national conference* 157–172 (Association for Computing Machinery, 1969). doi:10.1145/800195.805928.
43. George, A. & Liu, J. W. Computer Solution of Large Sparse Positive Definite Systems (Alan George and Joseph W. Liu. *SIAM Rev.* **26**, 289–291 (1984).
44. Azad, A., Jacquelin, M., Buluç, A. & Ng, E. G. The Reverse Cuthill-McKee Algorithm in Distributed-Memory. in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* 22–31 (2017). doi:10.1109/IPDPS.2017.85.
45. Mlakar, D., Winter, M., Parger, M. & Steinberger, M. Speculative Parallel Reverse Cuthill-McKee Reordering on Multi- and Many-core Architectures. in *2021 IEEE International*

Parallel and Distributed Processing Symposium (IPDPS) 703–713 (2021). doi:10.1109/IPDPS49936.2021.00080.

46. Karantasis, K. I., Lenharth, A., Nguyen, D., Garzarán, M. J. & Pingali, K. Parallelization of Reordering Algorithms for Bandwidth and Wavefront Reduction. in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* 921–932 (2014). doi:10.1109/SC.2014.80.
47. Bentley, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM* **18**, 509–517 (1975).
48. Bakan, A., Meireles, L. M. & Bahar, I. ProDy: Protein Dynamics Inferred from Theory and Experiments. *Bioinformatics* **27**, 1575–1577 (2011).
49. Lehoucq, R. B. & Sorensen, D. C. Deflation Techniques for an Implicitly Restarted Arnoldi Iteration. *SIAM J. Matrix Anal. Appl.* **17**, 789–821 (1996).
50. Lehoucq, R., Sorensen, D. & Yang, C. ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. *SIAM* (1998).
51. Anderson, E. *et al. LAPACK Users' Guide*. (Society for Industrial and Applied Mathematics, 1999). doi:10.1137/1.9780898719604.
52. Yu, V. W. *et al.* GPU-acceleration of the ELPA2 distributed eigensolver for dense symmetric and hermitian eigenproblems. *Computer Physics Communications* **262**, 107808 (2021).
53. Yu, V. W. *et al.* ELSI — An open infrastructure for electronic structure solvers. *Computer Physics Communications* **256**, 107459 (2020).
54. Wu, X., Davidović, D., Achilles, S. & Di Napoli, E. ChASE: a distributed hybrid CPU-GPU eigensolver for large-scale hermitian eigenvalue problems. in *Proceedings of the Platform for Advanced Scientific Computing Conference* 1–12 (Association for Computing Machinery, 2022). doi:10.1145/3539781.3539792.

55. Sgherzi, F., Parravicini, A. & Santambrogio, M. D. A Mixed Precision, Multi-GPU Design for Large-scale Top-K Sparse Eigenproblems. in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)* 1259–1263 (2022). doi:10.1109/ISCAS48785.2022.9937893.
56. Anzt, H. *et al.* Optimizing Krylov Subspace Solvers on Graphics Processing Units. in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops* 941–949 (2014). doi:10.1109/IPDPSW.2014.107.
57. Li, R., Xi, Y., Erlandson, L. & Saad, Y. The Eigenvalues Slicing Library (EVSL): Algorithms, Implementation, and Software. *SIAM J. Sci. Comput.* **41**, C393–C415 (2019).
58. Wu, K. & Simon, H. Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems. *SIAM J. Matrix Anal. Appl.* **22**, 602–616 (2000).
59. Bell, N. & Garland, M. Efficient Sparse Matrix-Vector Multiplication on CUDA | Research. *NVIDIA Technical Report NVR-2008-004* https://research.nvidia.com/publication/2008-12_efficient-sparse-matrix-vector-multiplication-cuda (2008).
60. Okuta, R., Unno, Y., Nishino, D., Hido, S. & Loomis, C. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)* (2017).
61. Nickolls, J., Buck, I., Garland, M. & Skadron, K. Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue* **6**, 40–53 (2008).
62. cuBLAS. https://docs.nvidia.com/cuda/cublas/.
63. cuSPARSE. https://docs.nvidia.com/cuda/cusparse/.
64. Pornillos, O., Ganser-Pornillos, B. K. & Yeager, M. Atomic-level modelling of the HIV capsid. *Nature* **469**, 424–427 (2011).
65. Caspar, D. L. D. & Klug, A. Physical Principles in the Construction of Regular Viruses. *Cold Spring Harb Symp Quant Biol* **27**, 1–24 (1962).

66. Mosalaganti, S. *et al.* AI-based structure prediction empowers integrative structural analysis of human nuclear pores. *Science* **376**, eabm9506 (2022).
67. Huang, C.-M., Kucinic, A., Johnson, J. A., Su, H.-J. & Castro, C. E. Integrated computer-aided engineering and design for DNA assemblies. *Nat. Mater.* **20**, 1264–1271 (2021).
68. Klose, T. *et al.* Structure of faustovirus, a large dsDNA virus. *Proceedings of the National Academy of Sciences* **113**, 6206–6211 (2016).
69. Hoffmann, A. & Grudinin, S. NOLB: Nonlinear Rigid Block Normal-Mode Analysis Method. *J. Chem. Theory Comput.* **13**, 2123–2134 (2017).
70. Pandey, M. *et al.* The transformational role of GPU computing and deep learning in drug discovery. *Nat Mach Intell* **4**, 211–221 (2022).
71. Ikegami, T., Sakurai, T. & Nagashima, U. A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method. *Journal of Computational and Applied Mathematics* **233**, 1927–1936 (2010).
72. Gulati, A. *et al.* Structural studies on chimeric Sesbania mosaic virus coat protein: Revisiting SeMV assembly. *Virology* **489**, 34–43 (2016).
73. Yang, Z., Bahar, I. & Widom, M. Vibrational Dynamics of Icosahedrally Symmetric Biomolecular Assemblies Compared with Predictions Based on Continuum Elasticity. *Biophysical Journal* **96**, 4438–4448 (2009).
74. Uto, S. *et al.* Mutual relationships between structural and functional changes in a PsbM-deletion mutant of photosystem II. *Faraday Discuss.* **198**, 107–120 (2017).
75. Lee, D. T. & Wong, C. K. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica* **9**, 23–29 (1977).
76. Hales, T. *et al.* A FORMAL PROOF OF THE KEPLER CONJECTURE. *Forum of Mathematics, Pi* **5**, e2 (2017).

77. Cuppen, J. J. M. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.* **36**, 177–195 (1980).
78. Francis, J. G. F. The QR Transformation A Unitary Analogue to the LR Transformation—Part 1. *The Computer Journal* **4**, 265–271 (1961).
79. Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards* **45**, 255–282 (1950).
80. Sorensen, D. C. Implicit Application of Polynomial Filters in a k-Step Arnoldi Method. *SIAM J. Matrix Anal. Appl.* **13**, 357–385 (1992).
81. Paige, C. C., Parlett, B. N. & van der Vorst, H. A. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numerical Linear Algebra with Applications* **2**, 115–133 (1995).
82. Saad, Y. 7. Filtering and Restarting Techniques. in *Numerical Methods for Large Eigenvalue Problems* 163–191 (Society for Industrial and Applied Mathematics, 2011). doi:10.1137/1.9781611970739.ch7.
83. Weiße, A., Wellein, G., Alvermann, A. & Fehske, H. The kernel polynomial method. *Rev. Mod. Phys.* **78**, 275–306 (2006).
84. Geus, R. The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities. (ETH Zurich, 2002). doi:10.3929/ethz-a-004469464.
85. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).
86. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods* **17**, 261–272 (2020).
87. Poppleton, E., Mallya, A., Dey, S., Joseph, J. & Šulc, P. Nanobase.org: a repository for DNA and RNA nanostructures. *Nucleic Acids Research* **50**, D246–D252 (2022).

88. Suma, A. *et al.* TacoxDNA: A user-friendly web server for simulations of complex DNA structures, from single strands to origami. *Journal of Computational Chemistry* **40**, 2586–2595 (2019).

Scalable anisotropic vibrations of megascale macromolecules Algorithms

Jordy Homing Lam

December 27, 2023

This is a supporting document submitted to support our article "Scalable anisotropic vibrations of megascale macromolecules". It contains the pseudocodes for the following algorithms:

1 3-D Reverse Cuthill McKee

A Reverse Cuthill McKee algorithm supplemented with a k-D tree data structure where k=3. Analysis of a k-D tree can be found in the work of 'Bentley, J. L. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 509–517 (1975)'. An empty 1-d array of length l is notated as 0^l. Time complexity marked in comment.

Algorithm 1 3-D Reverse Cuthill McKee

procedure 3DRCM($X \in \mathbb{R}^{N\times 3}$, $R_C \in \mathbb{R}^1$)
▷ (a) Prepare a 3-D tree $O(NlogN)$
  $3dTree \leftarrow KdTree(X)$
▷ (b) Find peripheral node $O(N \times 3N^{2/3})$
  Initialize $D = 0^N$ ▷ Degrees.
  for $i = 0, ..., N-1$ **do**
    $|\mathcal{N}(X_i)| = 3dTree(R_C, X_i)$
    $D[i] \leftarrow |\mathcal{N}(X_i)|$
  $Inds = Argsort(D)$; $IndsRev = Argsort(Inds)$ ▷ Quicksort.
  $m = max(D)$
▷ (c) Breadth-first loop $O(3N^{5/3} + 2cm^2N)$
  Initialize $L = 0^N$; $\tilde{D} = 0^m; n = 0$
  for $z = 0, ..., N-1$ **do**
    if $Inds[z] == -1$ **then**
      continue
    $L[n] \leftarrow Inds[z]$; $n \mathrel{+}= 1$
    $Inds[IndsRev[Inds[z]]] \leftarrow -1$ ▷ Indicate visited.
    $Level_s = n-1; Level_t = n$
    while $Level_s < Level_t$ **do**
      for $\tilde{i} = Level_s, ..., Level_t - 1$ **do**
        $i \leftarrow L[\tilde{i}]$; $n_{old} = n$
        $\mathcal{N}(X_i) = 3dTree(R_C, X_i)$ ▷ Neighbors in R_C.
        for $j = 0, ..., |\mathcal{N}(X_i)| - 1$ **do**
          $\tilde{j} = \mathcal{N}(X_i)[j]$
          if $Inds[IndsRev[\tilde{j}]] == -1$ **then**
            continue
          $Inds[IndsRev[\tilde{j}]] \leftarrow -1$
          $L[n] \leftarrow j$; $n \mathrel{+}= 1$
        $l_s = 0$
        for $k = n_{old}, ..., n-1$ **do** ▷ Insertion Sort.
          $\tilde{D}[l_s] \leftarrow D[L[k]]$; $l_s \mathrel{+}= 1$
        for $k = 1, ..., l_s - 1$ **do**
          $\tilde{d} \leftarrow \tilde{D}[k]$
          $\tilde{l} \leftarrow L[n_{old} + k]$
          $l_t \leftarrow k$
          while $l_t > 0$ and $\tilde{d} < \tilde{D}[l_t - 1]$ **do**
            $\tilde{D}[l_t] \leftarrow \tilde{D}[l_t - 1]$
            $L[n_{old} + l_t] \leftarrow L[n_{old} + l_t - 1]$; $l_t \mathrel{-}= 1$
          $\tilde{D}[l_t] \leftarrow \tilde{d}$
          $L[n_{old} + l_t] \leftarrow \tilde{l}$
      $Level_s \leftarrow Level_t$; $Level_t \leftarrow n$
▷ (d) Reverse
  return $L[:: -1]$

2 Modified Gram-Schmidt Vector

A Modified Gram-Schmidt (MGS) algorithm to orthogonalize a vector u against the basis $V = [v_0, ..., v_m]$.

Algorithm 2 Modified Gram-Schmidt Vector

procedure MGSV($u \in \mathbb{R}^n; V \in \mathbb{R}^{m \times n}$)
  for $i = 0, ..., m-1$ **do**
    $u \leftarrow u - (u^\top v_i)v_i$
  return u

3 Iterative Classical Gram-Schmidt

An Iterative Classical Gram-Schmidt (ICGS) algorithm to orthogonalize a vector u against the basis $V = [v_0, ..., v_m]$. A break is triggered when the deflation is revoked.

Algorithm 3 Iterative Classical Gram-Schmidt

procedure ICGS($u \in \mathbb{R}^n; V \in \mathbb{R}^{m \times n}$)
  $r_0 = ||u||_2$
  for $i_{iter} = 1, 2, 3$ **do**
    $u \leftarrow u - VV^\top u$
    $r_1 = ||u||_2$
    if $r_1 > r_0/2$ **then**
      Break
    $r_0 \leftarrow r_1$
  if $r_1 \leq r_0/2$ **then**
    Warning! Loss of orthogonality
  return u

4 p-step Lanczos Factorization

A p-step Lanczos Factorization (PLF) to produce p basis between index j_s and j_t. Algorithm 4 PLF is where a polynomial filter can be applied, for example, in Algorithm 8 CF with a first-kind Chebyshev polynomial. Step 9 and 10 are optional; a proof of the spectrum bound can be found in 'Zhou et al, Bounding the spectrum of large Hermitian matrices, Linear Algebra and its Applications 435(3), 2011, p.480-493, https://doi.org/10.1016/j.laa.2010.06.034'. In our case, the lower bound is well known.

Algorithm 4 p-step Lanczos Factorization

procedure PLF($A \in \mathbb{R}^{n \times n}, V \in \mathbb{R}^{(m+1) \times n}, \alpha \in \mathbb{R}^m, \beta \in \mathbb{R}^m, j_s = 0, j_t = m$)
  for $j = j_s, ..., j_t - 1$ **do**
    $r = Av_j$ ▷ $r = p(A, v_j, a, b)$ if filter in use.
    $\alpha_j \leftarrow v_j^\top r$
    $r \leftarrow r - \alpha_j v_j$
    $r \leftarrow MGSV(r, V[:j]); r \leftarrow MGSV(r, V[:j])$ ▷ Full Reorthogonalization (FRO).
    $\beta_j \leftarrow ||r||_2$
    $v_{j+1} = r/\beta_j$
  $T[j, j] = \alpha; T[j, j-1] = T[j-1, j] = \beta$
  Solve $TQ = QD$ ▷ Spectrum upper bound as $D_{max} + \beta_j ||e_j^\top Q_j||_\infty$
  return V, α, β

5 Implicitly Restarted Lanczos Method

An Implicitly Restarted Lanczos Method (IRLM) to compute k smallest eigenpairs. This algorithm calls Algorithm 2 MGSV and Algorithm 4 PLF. At max, 15000 restarts were allowed.

Algorithm 5 Implicitly Restarted Lanczos Method

procedure IRLM($A \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{(m+1)\times n}, \alpha \in \mathbb{R}^m, \beta \in \mathbb{R}^m, \epsilon = 1e-8$)
  Initialize $v_0 \leftarrow v_{rand}/||v_{rand}||_2$
  $j_s = 0$
  for $i_{iter} = 1, ..., 15000$ **do**
    ▷ (a) Initial p-steps Lanczos Factorization
    PLF(A,V,α,β, $j_s = j_s$, $j_t = m$)
    ▷ (b) Implicit Shift
    $S, W = eig(\alpha, \beta[:m-1])$
    $\theta = Sort(Diag(S))[::-1][:(m-k)]$
    $\tilde{\beta} = \beta_{k+p-1}$
    $Q = I$
    for $i = 0...p-1$ **do**
      $T = Tridiag(\alpha, \beta[:m-1])$
      $\tilde{Q}\tilde{R} = T - \theta_i I$
      $T \leftarrow \tilde{Q}^\top T \tilde{Q}$
      $\alpha, \beta \leftarrow T$
      $Q \leftarrow Q\tilde{Q}$
    $\beta \leftarrow [\beta, \tilde{\beta}]$
    ▷ (c) Implicit Restart
    $\sigma = Q[k-1, m-1]$
    $V[:k] \leftarrow Q[:k, :]V[:m, :]$
    $v_k \leftarrow \beta_{k-1}Q[k, :]V[:m, :] + \sigma\beta_{m-1}v_m$
    $\beta_{k-1} \leftarrow ||v_k||_2$
    $v_k \leftarrow v_k/\beta_{k-1}$
    $v_k \leftarrow MGSV(v_k, V[:k]); v_k \leftarrow MGSV(v_k, V[:k])$
    $v_k \leftarrow v_k/||v_k||_2$
    $j_s \leftarrow k$ ▷ Reset the PLF
    ▷ (d) Estimate Convergence
    if $|\beta_{k-1}| < \epsilon$ **then**
      Break
  $S, W = eig(Tridiag(\alpha[:k], \beta[:k-1]))$
  return $S[:k], W^\top V[:k, :]$

6 Thick Restart Lanczos Method

A Thick Restart Lanczos Method to compute the k smallest eigenpairs. This algorithm calls Algorithm 2 MGSV and Algorithm 4 PLF. Algorithm 4 PLF is where a filter can be applied through the matrix-vector multiplication, for example, in Algorithm 9, a filter is constructed using first-kind Chebyshev polynomial basis. At max, 15000 restarts were allowed.

Algorithm 6 Thick Restart Lanczos Method

procedure TRLM($A \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{(m+1)\times n}, \alpha \in \mathbb{R}^m, \beta \in \mathbb{R}^m, \epsilon = 1e-12$)
  Initialize $v_0 \leftarrow v_{rand}/||v_{rand}||_2$
  $j_s = 0; b = 0; S = 0; r = v_0$
  for $i_{iter} = 1, ..., 15000$ **do**
    ▷ (a) Reinitialize
    $\beta[:k] \leftarrow 0$
    $\alpha[:k] \leftarrow Diag(S)$
    $r \leftarrow MGSV(r, V[:k])$
    $r \leftarrow r/||r||_2$
    $v_k \leftarrow r$
    ▷ (b) p-step Lanczos Factorization, filter inside.
    PLF(A,V,α,β, $j_s = j_s$, $j_t = m$)
    ▷ (c) Thick Restart
    $T = Tridiag(\alpha, \beta[:m-1]); T[k,:k] \leftarrow b; T[:k,k] \leftarrow b$
    $S, W = eig(T)$
    $W \leftarrow W[:m, Argort(Diag(S))[:k]]$
    $S \leftarrow Diag(S)[Argort(Diag(S))[:k]]$
    $V[:k,:] \leftarrow W^\top V$
    $b = \beta_{m-1} W[m-1,:k]$
    $j_s \leftarrow k$ ▷ Reset the PLF
    ▷ (d) Check Convergence
    if $||b||_2 < \epsilon$ **then**
      Break
  return S, V

7 Jacobi-Davidson Method

A Jacobi-Davidson Method to compute k smallest eigenpairs. This algorithm calls Algorithm 2 MGSV and Algorithm 3 ICGS. The correction z is approximated with Generalized Minimal Residual Iteration (GMRES) routine provided in CuPy; unless otherwise stated, the iteration is stopped if residual error in GMRES reach 1e-6 or the number of iterations exceeds 20. Note that this can be replaced by a Minimal Residual Iteration (MIRES) routine, which processes symmetric matrices. j_s counts towards the desired number of converged Ritz pair k. Some preconditioners (e.g. ILU(k)) were considered in solving the correction equation, but their size and density become prohibitive as n increases; trivial preconditioner e.g. $diag(A)^{-1}$ does not improve performance.

Algorithm 7 Jacobi-Davidson Method

procedure JDM($A \in \mathbb{R}^{n \times n}, \epsilon = 1e-12, k = 64$)
  Initialize $v_0 = v_{rand}/||v_{rand}||_2$
  $G = v_0^\top A v_0$
  $V = [v_0,]$
  $Q = [0^n,]; \Lambda = []$
  $j_s = 0$

  for $i_{iter} = 1, ..., 15000$ **do**
    $S, W = eig(G)$
    while True **do**
                                                ▷ (a) Get Residual
      $u = VW[:, 0]$; $\theta = S[0, 0]$
      $r = Au - \theta u$
      $\sigma = \theta$ ▷ Propose shift.
      $\tilde{Q} = [Q, u]$

      if $(||r||_2 > \epsilon) \cup ((dim(S)[0] <= 1) \cap (j_s != k - 1))]$ **then** ▷ Non-convergence.
        Break.
                                                ▷ (b) Update Projections
      $\Lambda \leftarrow [\Lambda, \theta]$
      $Q \leftarrow \tilde{Q}$
      $V \leftarrow VW[:, 1 : j]; S \leftarrow S[1 : j, 1 : j]$
      $G \leftarrow S; W \leftarrow I$
      $j_s \mathrel{+}= 1$
      if $j_s == k$ **then**
        Return Λ, Q ▷ All converged.

    if $dim(S)[0] == 2k$ **then**   ▷ (c) Restart if workspace is full
      $V \leftarrow VW[:, 0 : k]; S \leftarrow S[0 : k, 0 : k]$
      $G \leftarrow S; W \leftarrow I$
                                                ▷ (d) Correction equation with GMRES.
    $(I - uu^\top)(A - \theta I)(I - uu^\top)z = -r$
    $z \leftarrow MGSV(\tilde{Q}, z)$
    $z \leftarrow ICGS(V, z)$; $z \leftarrow ICGS(V, z)$
    $z \leftarrow z/||z||_2$
                                                ▷ (e) Update Projections
    $\tilde{z} = Az$
    $V \leftarrow [V, z]$
    $G \leftarrow [G, V^\top \tilde{z}; \tilde{z}^\top V, z^\top \tilde{z}]$
                                                ▷ (f) Guard not all converged.
  return Λ, Q

8 Chebyshev-Davidson Method

A Chebyshev-Davidson Method to compute k smallest eigenpairs. This algorithm calls Algorithm 2 MGSV and Algorithm 3 ICGS. λ_{min} is the lower bound of spectrum. λ_{max} is upper bound of the spectrum obtained from Algorithm 4 PLF. θ_s is the 'squeezing' moving lower bound.

Algorithm 8 Chebyshev-Davidson Method

procedure CDM($A \in \mathbb{R}^{n\times n}, \lambda_{min}, \lambda_{max}, \epsilon = 1e-12, k = 64,$)
  Initialize $v_0 = v_{rand}/||v_{rand}||_2$
  $G = v_0^\top A v_0$
  $V = [v_0,]$
  $Q = [Av_0,]; \Lambda = []$
  $j_s = 0; j = 1$
  $u = V[:, j_s]$
  $\theta_s = (\lambda_{max} + G[0,0])/2$

  for $i_{iter} = 1, ..., 15000$ **do**
    $z = p(A, u, \theta_s, \lambda_{max})$ ▷ (a) Low-pass Chebyshev filter
    $z \leftarrow MGSV(\tilde{Q}, z)$
    $z \leftarrow ICGS(V, z)$
    $z \leftarrow z/||z||_2$
    ▷ (b) Update Projections
    $\tilde{z} = Az$
    $V[:, j] \leftarrow z$
    $Q[:, j] \leftarrow \tilde{z}$
    $G[j, j_s : j] \leftarrow Q[:, j]^\top V[:, j_s : j+1]$
    $G[j_s : j, j] \leftarrow G[j, j_s : j]^\top$
    $S, W = eig(G)$
    ▷ (c) Restart if workspace is full
    $j_r = j + 1$
    if $j + 1 >= 2k$ **then**
      $j_r = max(j_s + 1, k + 5, min(k + j_s, 2k - 5))$
    $V[:, j_s : j_r] \leftarrow V[:, j_s : j+1]W[:, 0 : j_r - j_s + 1]$
    $Q[:, j_s : j_r] \leftarrow Q[:, j_s : j+1]W[:, 0 : j_r - j_s + 1]$
    $G[j_s : j_r, j_s : j_r] \leftarrow S[0 : j_r - j_s + 1, 0 : j_r - j_s + 1]; W \leftarrow I$
    ▷ (d) Get Residual
    $\theta = S[0]$
    $r = Q[:, j_s] - \theta V[:, j_s]$
    $u = V[:, j_s]$ ▷ Next u
    if $||r||_2 < \epsilon$ **then**
      $\Lambda \leftarrow [\Lambda, \theta]; j_s += 1$
      if $j_s >= k$ **then**
        Return Λ, Q ▷ All converged.
      $u = V[:, j_s - 1]$ ▷ Next u
    $j = j_r$
    ▷ (e) Update moving lower bound
    $\theta_s = max(S_{median}, \lambda_{min})$
    ▷ (f) Guard not all converged.
  return Λ, Q

9 Chebyshev Filtered Matrix Vector Product

Matrix vector product filtered on the basis of first-kind Chebyshev polynomial. $p(t) = \sum_{j=0}^{j=M} \kappa_j T_j(t)$. Note that (1) κ_j is a set of user-defined coefficients e.g. the scaled coefficients in equation 28 of the main text. (2) In Algorithm 8 CDM, only the M-th degree polynomial is considered.

Algorithm 9 Chebyshev Filtered Matrix Vector Product

procedure ChebAv($A \in \mathbb{R}^{n\times n}, v \in \mathbb{R}^{n\times 1}, a, b$)
  $e =: (b-a)/2; c =: (b+a)/2$
  $y = \kappa_0 v$
  $v_+ = \kappa_1(Av - cv)/e$
  $y \mathrel{+}= \kappa_1 v_+$
  $v_- = v; v \leftarrow v_+;$
  for $j = 2, ..., M$ **do**
    $v_+ = 2/e(Av - cv) - v_-$
    $y \mathrel{+}= \kappa_j v_+$
    $v_- \leftarrow v; v \leftarrow v_+$
  return y