Journal of Computational Physics 309 (2016) 314–323

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A fast band–Krylov eigensolver for macromolecular functional motion simulation on multicore architectures and graphics processors

José I. Aliaga[a,*], Pedro Alonso[b], José M. Badía[a], Pablo Chacón[c], Davor Davidović[d], José R. López-Blanco[c], Enrique S. Quintana-Ortí[a]

[a] *Depto. Ingeniería y Ciencia de Computadores, Universitat Jaume I, Castellón, Spain*
[b] *Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Spain*
[c] *Dept. Biological Chemical Physics, Rocasolano Physics and Chemistry Institute, CSIC, Madrid, Spain*
[d] *Rudjer Bošković Institute, Centar za Informatiku i Računarstvo – CIR, Zagreb, Croatia*

A R T I C L E I N F O

Article history:
Received 3 April 2015
Received in revised form 28 September 2015
Accepted 6 January 2016
Available online 8 January 2016

Keywords:
Computational biology
Macromolecular machines
Eigenvalue problems
Krylov subspace methods
Multicore processors
Graphics processors

A B S T R A C T

We introduce a new iterative Krylov subspace-based eigensolver for the simulation of macromolecular motions on desktop multithreaded platforms equipped with multicore processors and, possibly, a graphics accelerator (GPU). The method consists of two stages, with the original problem first reduced into a simpler band-structured form by means of a high-performance compute-intensive procedure. This is followed by a memory-intensive but low-cost Krylov iteration, which is off-loaded to be computed on the GPU by means of an efficient data-parallel kernel.
The experimental results reveal the performance of the new eigensolver. Concretely, when applied to the simulation of macromolecules with a few thousands degrees of freedom and the number of eigenpairs to be computed is small to moderate, the new solver outperforms other methods implemented as part of high-performance numerical linear algebra packages for multithreaded architectures.

1. Introduction

Living cells consist of long chains of aminoacids and nucleotides, usually assembled into large macromolecular machines that support the main biological functions. At the molecular level, the biological activity of these components can be studied via Molecular Dynamics (MD) simulations of their dynamics and interactions. These studies provide detailed information on the fluctuations and conformational changes of protein and nucleic acids macromolecules using available 3D atomic structures. Unfortunately, the large size of these macromolecules and the long time scale of their motion turn MD simulations too costly or even prohibitive in practice.

Coarse-grain (CG) models combined with normal mode analysis (NMA) compose a powerful alternative to simulate collective motions of macromolecular complexes at extended time scales [1,2]. For example, the experiments in [3–8] demonstrate that CG-NMA can efficiently model molecular flexibility, thus becoming an appealing alternative for expen-

[*] Corresponding author.
E-mail address: aliaga@uji.es (J.I. Aliaga).

http://dx.doi.org/10.1016/j.jcp.2016.01.007
0021-9991/© 2016 Elsevier Inc. All rights reserved.

sive atomistic simulations. Moreover, reformulating NMA in internal coordinates (ICs) [9] greatly reduces the number of degrees of freedom (DOF), extending the applicability of NMA to even larger macrocomplexes.

The diagonalization of a large-scale matrix pair (i.e., the solution of a generalized eigenproblem) is the most expensive operation in IC-NMA. In [10] we compared two approaches to compute a fraction of the spectrum of moderate-scale dense symmetric definite generalized eigenproblems, based respectively on the reduction to tridiagonal form and the Krylov subspace iteration. In [11] we particularized this analysis for molecular dynamics (MD) simulations, showing that the iterative Krylov subspaces are an attractive solution for large-scale macromolecules, with up to 150,000 degrees of freedom (DOFs), when tackled on a cluster of multicore processors.

In this paper we further enhance the appeal of the Krylov subspace-based methods by introducing a new iterative solver that combines an initial transformation of the original data into a band-structured problem, followed by the classical Krylov iteration applied to this reduced problem. The advantages of the new "band-Krylov" method lie in that *i)* it transfers a large fraction of the iteration cost into the initial efficient reduction to band form; and *ii)* it then operates the low-performance Krylov iteration on a smaller problem which implies a lower cost of this stage and, additionally, the possibility of accommodating the data closer to the processor floating-point units (FPUs). For the second stage, as an additional contribution of this work, we develop an efficient implementation of a symmetric band matrix-vector product that significantly accelerates the computation of the Krylov iteration on a graphics processing unit (GPU).

Overall, we expect that the combination of these three factors (fast and efficient reduction to band form, iterative procedure with the reduced band matrix, and acceleration of the iteration by means of a fast GPU kernel) offers a more efficient solution for moderate- to large-scale problems as the number of iterations required for convergence grows. The results with large-scale macromolecules, featuring several tens of thousands DOFs, expose the advantages of the new band-Krylov solver compared with the original "full-Krylov" method as well as the methods included in linear algebra libraries for high-performance multicore servers with or without GPUs.

The rest of the paper is structured as follows. In Section 2 we summarize the principles behind macromolecular motion simulation in internal coordinates, identifying the key numerical problem appearing in this approach. In Section 3 we revisit some popular numerically-stable methods for the solution of eigenvalue problems, and we provide a list of existing libraries and packages that implement these methods for current multicore architectures and GPUs. This section also discusses the properties of these methods from the point of view of computational cost and performance, motivating the introduction of the new fast cache-efficient band–Krylov eigensolver in Section 4. The experimental results follow in Section 5 using a recent hybrid platform consisting of a 6-core Intel Xeon "Sandy Bridge" processor, connected to an NVIDIA "Kepler" graphics card. Finally, the paper is closed with a few concluding remarks in Section 6.

2. Capturing macromolecular dynamics in internal coordinates

Macromolecular motions can be described in normal mode analysis (NMA) by approximating the potential (Hessian) and kinetic energies as quadratic functions of the atomic positions and velocities, respectively. This in turn allows the decomposition of the motion into a series of vectors that encode the potential displacement directions and can be obtained from the modes (i.e., the eigenvalues) of the second derivative of the potential and kinetic energy matrices. The low frequency modes, in particular, correspond to soft collective conformational changes, which are related to functional motions [12,13], and feature a close relation with atomistic MD [14–16].

iMod [9] exploits classical NMA formulations in internal coordinates (ICs) while extending them to model large-scale macromolecular structures. In this representation, the potential energy of the system is expressed as

$$A = \frac{1}{2} q\, H q^T, \tag{1}$$

where the vector q contains the displacement from the equilibrium conformation ($q = q^e - q^0$);

$$H = (H_{\alpha,\beta}) = \frac{\partial^2 V}{\partial q_\alpha \partial q_\beta} \tag{2}$$

is the Hessian matrix; and the α and β subindices represent any IC to compute the partial derivatives. Furthermore,

$$V = \sum_{i<j} F_{ij} (r_{ij}^e - r_{ij}^0)^2, \tag{3}$$

where F_{ij} is the spring stiffness matrix, and r_{ij} denotes the distance between atoms i and j.

Additionally, the kinetic energy is given by:

$$B = \frac{1}{2} \bar{q}\, T\, \bar{q}^T, \tag{4}$$

where $\bar{q} = dq/dt$, and the kinetic energy matrix is defined as

$$T = (T_{\alpha,\beta}) = \sum_i m_i \frac{\partial r_i}{\partial q_\alpha} \cdot \frac{\partial r_i}{\partial q_\beta}. \tag{5}$$

Here, the mass of the i-th atom is m_i, while r_i stands for the corresponding Cartesian position vector. In general, the solution of the Lagrange equation of motion ($L = T - V$) can be encoded in the lowest frequency components of the eigenproblem defined by the matrix pair (A, B); see [11] for details.

3. Solution of symmetric definite eigenproblems

The IC-NMA method described in Section 2 requires the solution of a generalized symmetric definite eigenproblem of the form

$$AX = BX\Lambda, \tag{6}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times n}$ are both dense symmetric positive definite, corresponding respectively to the Hessian and kinetic energy matrices that control the dynamics of the macromolecular complex; $\Lambda \in \mathbb{R}^{s\times s}$ is a diagonal matrix with the s sought-after smallest eigenvalues (or modes) of the matrix pair (A, B) on its diagonal entries; and the columns of $X \in \mathbb{R}^{n\times s}$ contain the associated unknown eigenvectors [17]. Furthermore, for accurate macromolecular motion simulation $n \geq 10{,}000$, but typically only the s smallest eigenpairs (i.e., the eigenvalues and eigenvectors for the low energy modes) are necessary, with $s \approx 2$–10% of n.

Numerically-reliable eigensolvers for (6) initially transform this generalized equation into a standard symmetric eigenproblem:

$$CY = Y\Lambda, \tag{7}$$

where $C = U^{-T}AU^{-1} \in \mathbb{R}^{n\times n}$ is symmetric, $Y = UX \in \mathbb{R}^{n\times s}$, and $U \in \mathbb{R}^{n\times n}$ is the Cholesky factor of B (i.e., $B = U^TU$ with U upper triangular) [17]. Thus, the standard eigenproblem (7) shares its eigenvalues with those of (6), while the original eigenvectors can be easily recovered from $X = U^{-1}Y$, which only requires the solution of a few triangular linear system.

Since all the methods described next share the initial transformation into a standard eigenvalue problem and the back-transform to obtain the original eigenvectors, we will omit these two operations from the following discussion.

3.1. Direct eigensolvers

A popular approach to solve (7) commences by initially reducing C to a symmetric tridiagonal matrix $T \in \mathbb{R}^{n\times n}$ via an eigenvalue-preserving similarity transform:

$$Q^TCQ = T, \tag{8}$$

defined by the orthogonal matrix $Q \in \mathbb{R}^{n\times n}$. This is followed by a subsequent refinement to extract the eigenvalues from the associated symmetric tridiagonal eigenproblem:

$$TZ = Z\Lambda, \tag{9}$$

where $Z \in \mathbb{R}^{n\times s}$ comprises the sought-after eigenvectors of T. Finally, the eigenvectors of the standard problem are recovered from $Y = QZ$. From the numerical perspective, the most involved part of this approach is the solution of the tridiagonal eigenproblem (9), which can be reliably tackled, e.g., via the QR algorithm or the MRRR method [17], both available as part of LAPACK [18]. On the other hand, from the computational point of view, the most expensive component in terms of floating-point arithmetic operations (flops) is the reduction of C to tridiagonal form (8). In particular, this reduction costs $O(n^3)$ flops while, in general, the subsequent solution of the tridiagonal eigenproblem only requires $O(n^2)$ flops [17]. Recovering Y from Z only adds $O(n^2s)$ flops to these figures.

We next describe two alternative approaches to obtain the tridiagonal matrix T and the associated orthogonal factor Q, and refer to numerical software packages for this purpose.

3.1.1. One-stage reduction to tridiagonal form

LAPACK's routine SYTRD performs the transformation (8) in a *single stage* via Householder reflectors. Specifically, at a given iteration $j = 1, 2, \ldots, n-2$, this procedure computes a reflector H_j that annihilates all elements below the first subdiagonal of the j-th column of the current matrix $C^{(j-1)}$, with $C^{(0)} = C$, accumulating this transform as

$$C^{(j)} = H_j^T C^{(j-1)} H_j = H_j^T\,(H_{j-1}^T H_{j-2}^T \cdots H_1^T C H_1 \cdots H_{j-1})\,H_j, \tag{10}$$

so that $Q = H_1 \cdots H_{n-3}H_{n-2}$ yields the desired reduction in (8). To attain high performance, the application of these reflectors is partially delayed, aggregating groups of b (block size) transforms, so as to permit a more efficient update of C via a blocked algorithm [17]. The overall cost of performing the reduction (8) using routine SYTRD is $4n^3/3$ flops, provided $b \ll n$. Furthermore, the back-transform $Y = QZ$ can be computed using LAPACK's routine ORMTR, at a cost of $2n^2s$ additional flops, without ever forming Q.

Following the traditional approach, the LAPACK routines for this procedure extract parallelism for (general-purpose) multicore processors by relying on a multithreaded implementation of BLAS. Alternative multithreaded implementations of these procedures have been recently proposed for (manycore) GPUs as part of the MAGMA [19] library.

3.1.2. Multi-stage reduction to tridiagonal form

The problem with the one-stage reduction just described is that half of the flops performed by routine SYTRD are cast in terms of the symmetric matrix-vector product (SYMV), which delivers a small fraction of the peak performance on current multicore and manycore architectures.

The *multi-stage methods* shift a major part of the computations necessary for the reduction into symmetric rank-$2b$ updates (SYR2K), basically equivalent to the efficient matrix-matrix product. In exchange, these methods incur a considerable increase of the computational cost. Let us consider, for simplicity, a two-stage algorithm. The idea is to initially transform C into a symmetric band matrix $W \in \mathbb{R}^{n\times n}$, with bandwidth w, via an orthogonal similarity transform Q_1, to then further reduce W into the tridiagonal matrix T using an additional orthogonal similarity transform Q_2. The first stage mostly consists of fast rank-$2b$ stages, while the type of operations appearing in the second stage are the same as those in the slow reduction to tridiagonal form. Provided $w \ll n$, this two-stage calculation of T roughly costs $4n^3/3$ flops, i.e., the same as the one-stage algorithm. However, the construction of the orthogonal matrix $\hat{Q} = Q_1Q_2$ that performs the full transformation $Q_2^T(Q_1^TCQ_1)Q_2 = Q_2^TWQ_2 = T$, necessary to recover the eigenvectors of Y from those of Z, becomes considerably more expensive, requiring $2n^3 + 2n^2s$ additional flops.

There exists a complete implementation of the two-stage method in the SBR toolbox [20], which can leverage a multithreaded implementation of BLAS to exploit the hardware concurrency of current multicore processors. For these same architectures one can also leverage the codes in the PLASMA library [21]. A reimplementation of this approach to tackle large-scale problems, which do not fit into the memory of the GPU, was done in [22]. ELPA [23] and MAGMA [24] implement a variant of the two-stage method, for clusters and multi-GPU platforms, respectively. Both implementations exhibit significantly lower computational cost compared with the original SBR algorithm, but they are still significantly more expensive than the one-stage approach.

3.2. Iterative (Krylov subspace-based) eigensolvers

Alternatively, the solution of symmetric eigenproblems can be tackled by means of a Krylov subspace-based method [17], an inexpensive choice for the solution of large-scale dense eigenproblems on multicore architectures and GPUs [25].

When applied to C, starting from an initial random vector $v_1 \in \mathbb{R}^n$, all Krylov subspace-based methods construct a basis of the Krylov subspace spanned by this matrix, defined as

$$\mathcal{K}(C, v_1, n) = \text{span}\{v_1, Cv_1, \ldots, C^{n-1}v_1\} = \text{span}\{v_1, v_2, \ldots, v_n\}. \tag{11}$$

If C is symmetric and positive definite, this matrix can be reduced to tridiagonal form via similarity orthonormal transform defined by $V = [v_1, v_2, \ldots, v_n]$; i.e.,

$$V^TCV = T = \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ 0 & & \beta_{n-1} & \alpha_n \end{bmatrix}, \quad V^TV = I. \tag{12}$$

Thus, equating the columns in $CV = VT$, the Lanczos three-term recurrence is obtained

$$Cv_k = \beta_{k-1}v_{k-1} + \alpha_k v_k + \beta_k v_{k+1}, \tag{13}$$

with $\beta_0 v_0 = 0$. Given the orthonormality of V, and operating properly on the recurrence, we obtain the expressions to compute the components of matrix T, which are the foundation of the Lanczos method:

$$\begin{aligned} \alpha_k &= v_k^T A v_k, \\ r_k &= \beta_k v_{k+1} = (C - \alpha_k I)v_k - \beta_{k-1}v_{k-1}, \\ \beta_k &= ||r_k||_2 \end{aligned} \tag{14}$$

such that the reduction in the m-th iteration is given, in matrix form, by

$$CV_m = V_mT_m + r_m e_m^T \tag{15}$$

This method exhibits low computational cost per iteration (in general, $2n^2$ flops) and, moreover, does not require an appreciable additional storage space. But the main feature of the Lanczos method is that the extremal eigenvalues of matrix T_m are good approximation of the extremal eigenvalues of matrix C, specially if there are not clustered, improving this condition when m increases. Empirically, it has been proved that if the number of sought-after eigenvalues of C is equal to s the size of the Krylov subspace has to be $m \geq 2s$. For macromolecular simulations it is convenient to compute the largest s eigenvalues of the pair (B, A), instead of the smallest s eigenvalues of the correlated problem (A, B) as, for these particular applications, the former have the appealing property of being extremal and well-separated, ensuring fast convergence of the iteration.

Krylov subspace methods are implemented as part of ARPACK [26]. This numerical package features a “reverse communication” interface so that the user is in charge of providing an efficient implementation of SYMV. We note that this is the only operation that “interacts” with the matrix in ARPACK, concentrating the bulk of the operations performed in this library. Efficient implementations of SYMV are offered as part of most tuned implementations of BLAS, including Intel MKL for multicore processors and CUBLAS for GPUs.

3.3. Discussion of existing methods

The methods described in the previous two subsections present a number of advantages and drawbacks that we next review in order to motivate our alternative solver.

The algorithms based on the two-stage (or multi-stage) reduction to tridiagonal form are mostly composed of compute-bound kernels (concretely, the level 3 BLAS SYR2K) which, in general, attain very high performance on current multicore and manycore architectures. However, their high cost often turns them too expensive for the solution of eigenproblems when only a reduced number of eigenvalues/eigenvectors is required [10]. A large fraction of this overhead is rooted in the need to explicitly build the orthogonal matrix $\hat{Q}$, which requires the additional $2n^3$ flops.

On the other hand, the one-stage reduction to tridiagonal form avoids the explicit construction of the orthogonal matrix, offering a much reduced computational cost compared with its two-stage counterparts. Nevertheless, half of the operations in this method are cast in terms of a memory-bound kernel (specifically, the level 2 BLAS SYMV), which offers very low performance in today’s architectures. Furthermore, the method incurs in the same cost independently of the number of eigenvalues that are required ($4n^3/3$ flops to reduce C to T), though the eigenvectors can be computed at a cost that is proportional to their number ($2n^2s$ flops to obtain Y from Z).

The Krylov subspace methods perform the bulk of its computations in terms of the memory-bound SYMV, and therefore can be expected to achieve only a small fraction of the processor’s peak performance. On the positive side, they are often the cheapest methods, with a cost of roughly $2n^2$ flops per iteration and a fast convergence rate for macromolecular motion simulations [11].

Consider the SYMV $y = Cx$, where $C \in \mathbb{R}^{n\times n}$ is symmetric and $x, y \in \mathbb{R}^n$, and let us denote the (i, j) entry of C by $c_{i,j}$ and the k-th entry of x/y as x_k/y_k. The low-performance of this kernel is due to the fact that there is no reuse of the matrix elements. In particular, for each element of C that is brought into the FPUs, the operation performs 4 flops ($y_i \mathrel{+}= c_{i,j} \cdot x_j$ and $y_j \mathrel{+}= c_{i,j} \cdot x_i$), rendering a ratio of 4 flops per memory operation (memop). On current architectures, where the memory bandwidth is much lower than the FPUs’ peak performance, the result is a memory-bound operation that is strongly constrained by the speed of the memory level where the data (i.e., the matrix C) resides.

A particularly harmful scenario for the SYMV kernel and, in consequence, for the Krylov subspace methods (as well as the one-stage reduction to tridiagonal form) is a problem where the data matrix C is too large to fit into the main memory and, therefore, has to be retrieved from disk at each iteration of the method. In this situation, the disk (I/O) bandwidth strictly dictates the performance of the Krylov subspace-based solver, while other parameters such as the number of cores, frequency, SIMD organization, etc. of the target architecture play a minor role. Similarly, in a problem with data too large to fit into the memory of a GPU, the matrix has to be transferred from the main memory to the graphics accelerator, via the PCI-e bus, once per iteration of the Krylov solver. In these circumstances, it is the PCI-e bandwidth that determines the performance of the solver, independently e.g. of the number of cores in the GPU.

Our approach to tackle this problem consists in reducing the dimension of the problem matrix involved in the Krylov subspace method so as to be able to fit the result closer to the FPUs, e.g. in the main memory instead of the disk, the GPU memory instead of the main memory, or the L3 cache instead of the main memory. Note that this will not improve the data reuse factor (flops to memops), but at least it will allow that the memops proceed at the speed of a faster memory level.

4. A fast cache-efficient band–Krylov eigensolver

Our new iterative algorithm is a “mixed” band–Krylov method that initially performs a reduction of C to a symmetric band matrix $\bar{W}$, much like the two-stage direct eigensolver, to then apply a Krylov subspace method on the reduced problem.

Concretely, the new method commences by transforming the standard problem into a symmetric band one:

$$\bar{W}\bar{Y} = \bar{Y}\Lambda \tag{16}$$

where $\bar{Q}^T C \bar{Q} = \bar{W}$ is an orthogonal similarity transform that produces the band matrix $\bar{W}$, and $Y = \bar{Q}\bar{Y}$. A major difference with the two-stage direct approach is that, in the direct case, the bandwidth of W was chosen to be small (usually, around 128–256, depending on the target architecture) so as to decrease the number of flops in the subsequent stage (low-performance reduction of band to tridiagonal form) while still ensuring high performance during the first stage.

Our goal and, consequently, our approach differs in that we choose the bandwidth w as large as possible, with the only limitation that the resulting band matrix $\bar{W}$, when stored in compact form as an $n \times (w + 1)$ dense matrix, fits into the “target” memory level. The cost for this initial reduction is thus $2((n - w)^2 w + (n - w)^3/3)$ flops, which are cast in terms of efficient Level 3 BLAS kernels. (Compare it with the $4n^3/3$ flops required for the calculation of the narrow-banded W in the direct case.)

In the second stage of our band–Krylov approach, we simply tackle the reduced problem (16) by applying a Krylov subspace-based eigensolver, with a cost of $2nw$ flops per iteration. Note that with this hybrid methodology, $\bar{Q}$ does not

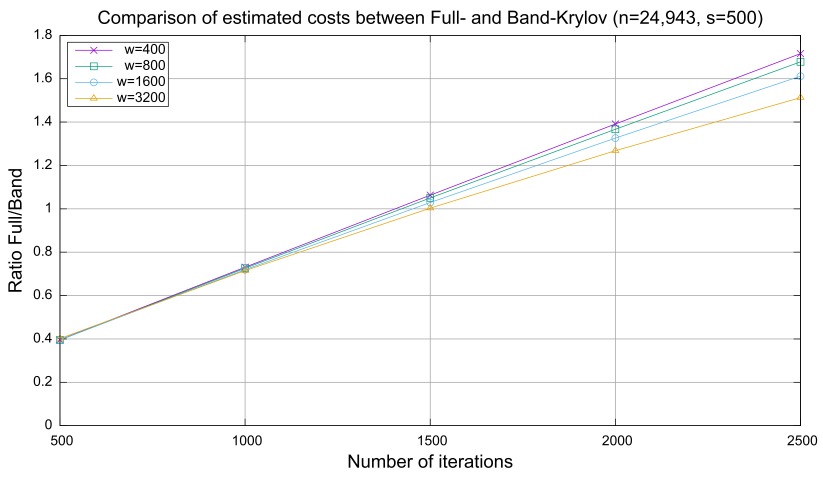

Fig. 1. Theoretical cost of the new band–Krylov eigensolver compared with the conventional Krylov subspace-based method.

need to be explicitly constructed but simply applied to $\bar{Y}$ in order to obtain the standard eigenvectors, at a cost of only $2(n-w)^2s$ flops. Thus, compared with the original Krylov method, we reduce the cost per iteration from $2n^2$ to $2nw$ flops, at the expense of the initial transform to band form. On the other hand, compared with the two-stage direct method, we avoid the costly formation of $\hat{Q}$.

In summary, there exists a trade-off between the cost of the two stages of our band–Krylov eigensolver and the bandwidth. Small/large values of w shift part of this cost towards/away from the initial transform but yield a cheap/costly Krylov iteration. Therefore, this parameter has to be chosen taking into account the convergence rate of the Krylov method.

Let us illustrate this balance via the computation of the s smallest eigenvalues of a random problem of "moderate" dimension n =24,943. Fig. 1 compares the estimated (or "expected") cost of the full- and band-Krylov solvers for this particular case. To perform this comparison, we consider the theoretical number of floating-point arithmetic operations (flops) for each solver:

- $2n^2$ flops for the full-Krylov solver (all of them casted as BLAS-2 kernels); and
- $2((n-w)^2w + (n-w)^3/3)$ flops for the initial reduction to band form (BLAS-3 kernels), with bandwidth w, plus $2nw$ flops per iteration (BLAS-2 kernels), in the band-Krylov solver.

Next, we assume that a BLAS-3 flop is $6\times$ faster than a BLAS-2 flop, and obtain estimations in the figure using this rate and the theoretical flops as the number of iterations varies. (We note that the performance factor between BLAS-2 and BLAS-3 was obtained by comparing the experimental flops/second rates of the matrix-vector product and the matrix multiplication on the same platform.)

The results in that figure show that the cost ratio between the two Krylov subspace-based eigensolvers strongly depends on the convergence rate of the problem and, to a less extent, on the bandwidth w selected for the band–Krylov method. In general, the theoretical cross-over point between the two eigensolvers is in the range of 1500 iterations, occurring a bit below that number in case $w = 3200$ or above it for narrower bandwidths.

5. Performance evaluation

5.1. Experimental setup

The following experiments were all performed using IEEE double-precision arithmetic on a server equipped with an Intel Xeon i7-3930K processor (6 cores at 3.2 GHz) and 24 GB of DDR3 RAM, connected to an NVIDIA Tesla K20 accelerator (2496 CUDA cores at 706 MHz, with 5 GB of GDDR5 RAM) via a PCI-e 16× bus.

The codes/libraries were processed with the Intel compiler `icc` (`composerve` 2011_sp1.9.293, `icc 12.1.3 20120212`), and linked to the implementation of LAPACK/BLAS implementations in Intel MKL (v10.3 update 9) and NVIDIA CUBLAS (v5.0). The MPI eigensolvers were linked to OpenMPI (1.6).

5.2. Macromolecules

For the comparison, we employ a case study corresponding to a microtubule (MT) 13:3 macromolecule, with three different instances, leading to eigenproblems of size n =12,469, 24,943 and 31,178. The atomic structure of MT was generated from the experimental coordinates (PDB-ID 1TUB), optimized using molecular dynamics, and kindly provided by M. Deriu [27]. For details on the biological significance of these cases, see [11]. NMA was performed under the elastic network model approximation with iMod [9], i.e. without any further energy minimization and considering that the optimized structure is at the minimum energy conformation. For each one of these cases, we compute the $s = 100, 200, 300, 400, 500$ and 1000 smallest eigenvalues and the associated eigenvectors of the pair (A, B). As argued earlier, in the Krylov subspace-based eigensolvers, we obtain this information from the s largest eigenvalues/eigenvectors of the pair (B, A).

The experimentation with other macromolecules offered similar qualitative results. We note that the execution time of ELPA and MAGMA basically depends on the problem dimension and the number of eigenvalues, but very little on the specific numerical data. For both Krylov-based eigensolvers, the execution time per iteration (and the reduction to band form for the band-Krylov solver) is independent of the numerical values, while the data itself can in principle affect the convergence rate. However, we did not observe large variations among different biological problems.

Collective motions of small proteins can often be captured with a dozen of modes, but this is not the case with large-sized computational challenging systems, so that the actual number of modes which are necessary depends on the problem dimension. We have reported that computing a number of modes that is around 2–10% of the problem dimension were needed to account for 70–90% of the conformational changes between atomic structures in different conformations [9]. In practical applications, such as morphing between atomic structures in different conformations [9,28] as well as for the flexible fitting of atomic structures into electron microscopy density maps [29], we found that, in general, these small percentages of modes suffices to obtain satisfactory results. The experimentation thus covers the practical application scenario for extracting the collective motions of relative large systems hardly accessible with Cartesian approaches.

5.3. *Eigensolvers*

We include the following eigensolvers in the experimental evaluation:

- `FKrylov`. The conventional Krylov subspace-based solver described in Section 3, with the SYMV products proceeding in the multicore processor. In the calls to ARPACK's routine DSAUPD, we set the following parameters: `nev`=100, 200, 300, 400, 500, 1000 (number of eigenvalues), `ncv`=2.5∗`nev` (number of vectors of the orthogonal matrix V_m), `tol`=1.0E−12 (stopping criterion), and `iparam[2]`=100 (maximum number of Arnoldi update iterations allowed). We also developed a version that computes the SYMV products on the GPU, using NVIDIA's CUBLAS routine for this operation. However, the performance advantage of this alternative implementation was limited: For the smallest problem ($n = 12{,}469$), the SYMV products proceed about 9% faster in the GPU. However, these benefits are in practice much lower as the SYMV only amounts for part of the computations during the iteration. On the other hand, the largest two problem sizes did not fit into the GPU memory and, therefore, the SYMV products could not proceed in the accelerator.
- `BKrylov`. The new band–Krylov subspace-based solver introduced in Section 4. The calls to ARPACK were performed using the same parameters as in the previous case. The bandwidth w is chosen in this case so that as to balance the cost of the initial reduction and the symmetric band matrix-vector (SBMV) products during the iteration. In other words, among the possible values of the bandwidth, we select that which optimizes the performance (i.e., execution time) of the global solver. A specialized GPU implementation of the SBMV product was developed as part of our optimization effort, showing a significant acceleration of this operation over the multi-core counterpart available in Intel MKL and the version of this operation in NVIDIA's CUBLAS. For example, for the smallest problem size and $w = 6400$, our GPU routine required, respectively, about 63% and 44% of the time employed by Intel MKL and NVIDIA CUBLAS implementations of SBMV.
- `ELPA1` and `ELPA2`[1] (Eigenvalue SoLvers for Petaflop-Applications, release 2013.11 v8). These message-passing eigensolvers exploit the multicore processors of the platform using one MPI rank per core, and do not off-load any part of the computation to the GPU. `ELPA1` performs a direct reduction to tridiagonal form, in a single stage, and then solves the associated tridiagonal eigenproblem. `ELPA2` proceeds in two-stages: reduction to band form and from there to tridiagonal form.
- `MAGMA`[2] (Matrix Algebra on GPU and Multicore Architectures, release 1.6.2). This hybrid CPU-GPU two-stage eigensolver exploits both the multicore processor and the GPU.

For the eigensolvers that employ the GPU (`BKrylov` and `MAGMA`) the results include the cost of transferring the input data and the results between main memory and GPU.

An additional advantage of the Krylov-based methods is that they consume considerably less memory than their direct alternatives. In particular, the two largest cases included in the experimentation could not be solved with `ELPA` and `MAGMA` due to lack of memory in the target server. To tackle this, we executed these solvers for several problem sizes of the same macromolecule (with $n \leq 15{,}000$), and then we performed a polynomial approximation of the execution time using a cubic function in the problem dimension.

1 http://elpa.rzg.mpg.de/.
2 http://icl.cs.utk.edu/magma/.

Table 1
Execution times (in seconds) of the different eigensolvers. The cells in red and blue identify, respectively, the best and second-to-best solver.

Size n	Solver	Number of eigenvalues s					
		100	200	300	400	500	1000
12,469	FKrylov	5.78	11.51	18.78	28.74	44.86	129.00
	BKrylov	9.63	11.23	15.84	21.65	31.02	103.60
	ELPA1	114.95	115.35	115.65	116.10	116.41	118.36
	ELPA2	67.95	80.41	107.02	116.90	146.35	219.29
	MAGMA	22.16	23.64	22.22	23.89	24.29	22.66
24,943	FKrylov	22.98	40.36	64.29	90.45	121.97	426.39
	BKrylov	58.32	56.18	62.45	70.43	102.31	285.01
	ELPA1	881.48	871.76	873.00	868.57	869.47	880.73
	ELPA2	396.42	433.95	529.21	584.80	704.89	993.46
	MAGMA	97.51	99.57	127.96	123.56	120.95	115.19
31,178	FKrylov	30.99	60.68	95.42	132.74	173.62	426.39
	BKrylov	105.28	113.39	120.95	136.83	155.12	285.01
	ELPA1	1714.47	1688.21	1689.41	1674.20	1675.82	1696.50
	ELPA2	717.79	764.79	903.62	1005.14	1197.91	1661.71
	MAGMA	157.25	158.17	237.92	218.83	210.24	202.40

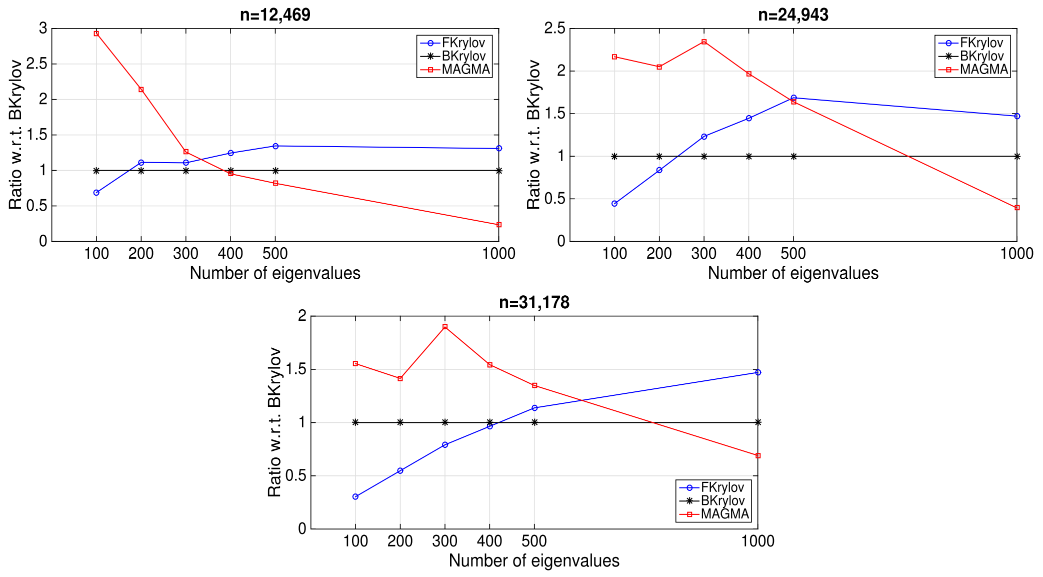

Fig. 2. Experimental comparison of the different eigensolvers. Performance ratios correspond to execution times normalized with respect to that of the band–Krylov eigensolver.

5.4. *General comparison of the methods*

Table 1 reports the execution of the eigensolvers for all three instances of the macromolecule MT and required number of eigenvalues s. These results expose that the band–Krylov eigensolver is the best or second-to-best option for all practical scenarios. In general, the fastest eigensolver is FKrylov for small values of s, BKrylov for moderate s, and MAGMA for large s, with the threshold values that define “small”, “moderate” and “large” depending on the problem size n. ELPA1 exhibits execution times which are mostly independent of the number of required eigenvalues. ELPA2 outperforms ELPA1 for small values of s but its cost rapidly grows with the number of eigenvalues, in the end being clearly slower than any other eigensolver.

Fig. 2 performs a quantitative comparison of the new band–Krylov eigensolver against the remaining implementations, plotting the ratios between the execution times of the latter normalized to those of BKrylov. We discard ELPA1 and ELPA2 from the following discussion since, as argued earlier, they are clearly inferior. For the smallest problem size, FKrylov offers an execution time that is only 60% that of BKrylov for $s = 100$ eigenvalues. However, as s grows to 200, 300, 400, 500 and 1000, FKrylov is outperformed by BKrylov in ratios that vary as 1.03, 1.19, 1.33, 1.45 and 1.25, respectively. The ratios between the execution times of the band–Krylov solver and the MAGMA routine for this small problem move in the opposite direction, decreasing with $s = 100 \rightarrow 1000$ as 2.30, 2.10, 1.40, 1.10, 0.78 and 0.21. Furthermore, the differences between these two eigensolvers in general decrease as the problem dimension grows and, thus, for the largest problem size, the ratios for growing values of $s = 100 \rightarrow 1000$ become 1.49, 1.39, 1.96, 1.59, 1.35 and 0.71. In summary, these results illustrate that the new eigensolver outperforms alternative existing methods for problems of moderate dimension when the number of eigenpairs to be computed is small to moderate.

6. Concluding remarks

We have presented a mixed eigensolver that combines the high-performance reduction to band form typical of direct multi-stage methods with the low-cost iteration of Krylov subspace methods. By appropriately choosing the bandwidth of the intermediate matrix that represents the problem, our hybrid band–Krylov eigensolver can thus shift part of its cost towards (away from) the initial band reduction and away from (towards) the Krylov iteration. The balance between these two stages is achieved taking into account the problem dimensions, convergence rate of the problem, target architecture, and efficiency of the underlying method. The solver is completed with a tailored symmetric band matrix-vector product that off-loads this operation to the GPU, resulting in a significant acceleration of the Krylov iteration.

The results using a server equipped with recent multi-core Intel Xeon technology and an NVIDIA Tesla “Kepler” show the performance superiority of the new band–Krylov eigensolver for the simulation of macromolecules with a few thousands degrees of freedom when the required number of eigenpairs is small to moderate. As an additional advantage, the new band–Krylov solver can handle larger problems on the GPU, as the initial reduction to band form can be performed out-of-core from the perspective of the accelerator, while the subsequent Krylov iteration proceeds in-core on a band matrix of reduced size.

Acknowledgements

This work was supported by the CICYT projects TIN2011-23283 and TIN2014-53495-R of the MINECO and FEDER, and project P1-1B2013-20 of the Fundació Caixa Castelló-Bancaixa and UJI.

References

[1] I. Bahar, A.J. Rader, Coarse-grained normal mode analysis in structural biology, Curr. Opin. Struct. Biol. 15 (5) (2005) 586–592.
[2] V. Tozzini, Minimalist models for proteins: a comparative analysis, Q. Rev. Biophys. 43 (3) (2010) 333–371.
[3] C.N. Cavasotto, J.A. Kovacs, R.A. Abagyan, Representing receptor flexibility in ligand docking through relevant normal modes, J. Am. Chem. Soc. 127 (26) (2005) 9632–9640.
[4] P. Chacon, F. Tama, W. Wriggers, Mega-Dalton biomolecular motion captured from electron microscopy reconstructions, J. Mol. Biol. 326 (2) (2003) 485–492.
[5] M. Delarue, P. Dumas, On the use of low-frequency normal modes to enforce collective movements in refining macromolecular structural models, Proc. Natl. Acad. Sci. USA 101 (18) (2004) 6957–6962.
[6] J. Franklin, P. Koehl, S. Doniach, M. Delarue, Minactionpath: maximum likelihood trajectory for large-scale structural transitions in a coarse-grained locally harmonic energy landscape, Nucleic Acids Res. 35 (2007) W477–W482 (Web Server issue).
[7] K. Hinsen, E. Beaumont, B. Fournier, J.J. Lacapere, From electron microscopy maps to atomic structures using normal mode-based fitting, Methods Mol. Biol. 654 (2010) 237–258.
[8] A. May, M. Zacharias, Energy minimization in low-frequency normal modes to efficiently allow for global flexibility during systematic protein–protein docking, Proteins 70 (3) (2008) 794–809.
[9] J.R. Lopez-Blanco, J.I. Garzon, P. Chacon, IMod: multipurpose normal mode analysis in internal coordinates, Bioinformatics 27 (20) (2011) 2843–2850.
[10] J. Aliaga, P. Bientinesi, D. Davidović, E.D. Napoli, F. Igual, E.S. Quintana-Ortí, Solving dense generalized eigenproblems on multi-threaded architectures, Appl. Math. Comput. 218 (22) (2012) 11279–11289.
[11] J.R. López-Blanco, R. Reyes, J.I. Aliaga, R.M. Badia, P. Chacón, E.S. Quintana, Exploring large macromolecular functional motions on clusters of multicore processors, J. Comput. Phys. 246 (2013) 275–288.
[12] W.G. Krebs, V. Alexandrov, C.A. Wilson, N. Echols, H. Yu, M. Gerstein, Normal mode analysis of macromolecular motions in a database framework: developing mode concentration as a useful classifying statistic, Proteins 48 (4) (2002) 682–695.
[13] L. Yang, G. Song, R.L. Jernigan, How well can we understand large-scale protein motions using normal modes of elastic network models? Biophys. J. 93 (3) (2007) 920–929.
[14] L. Orellana, M. Rueda, C. Ferrer-Costa, J.R. Lopez-Blanco, P. Chacón, M. Orozco, Approaching elastic network models to molecular dynamics flexibility, J. Chem. Theory Comput. 6 (9) (2010) 2910–2923.
[15] M. Rueda, P. Chacon, M. Orozco, Thorough validation of protein normal mode analysis: a comparative study with essential dynamics, Structure 15 (5) (2007) 565–575.
[16] A. Ahmed, S. Villinger, H. Gohlke, Large-scale comparison of protein essential dynamics from molecular dynamics simulations and coarse-grained normal mode analyses, Proteins 78 (16) (2010) 3341–3352.
[17] G.H. Golub, C.F.V. Loan, Matrix Computations, 3rd edition, The Johns Hopkins University Press, Baltimore, 1996.
[18] E. Anderson, Z. Bai, J. Demmel, J.E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A.E. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide, SIAM, Philadelphia, 1992.
[19] MAGMA project home page, http://icl.cs.utk.edu/magma/.
[20] C.H. Bischof, B. Lang, X. Sun, Algorithm 807: The SBR Toolbox—software for successive band reduction, ACM Trans. Math. Softw. 26 (4) (2000) 602–616. URL http://doi.acm.org/10.1145/365723.365736.
[21] PLASMA project home page, http://icl.cs.utk.edu/plasma/.
[22] D. Davidović, E.S. Quintana-Ortí, Applying OOC techniques in the reduction to condensed form for very large symmetric eigenproblems on GPUs, in: 20th Euro. Conf. PDP 2012, 2012, pp. 442–449.
[23] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, P. Willems, Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Comput. 37 (12) (2011) 783–794.
[24] A. Haidar, R. Solcà, M. Gates, S. Tomov, T. Schulthess, J. Dongarra, Leading edge hybrid multi-gpu algorithms for generalized eigenproblems in electronic structure calculations, in: J.M. Kunkel, T. Ludwig, H. Meuer (Eds.), Supercomputing, in: Lecture Notes in Computer Science, vol. 7905, Springer, Berlin/Heidelberg, 2013, pp. 67–80.

[25] P. Bientinesi, F.D. Igual, D. Kressner, M. Petschow, E.S. Quintana-Ortí, Condensed forms for the symmetric eigenvalue problem on multi-threaded architectures, Concurr. Comput. 23 (7) (2011) 694–707.
[26] ARPACK project home page, http://www.caam.rice.edu/software/ARPACK/.
[27] M.A. Deriu, M. Soncini, M. Orsi, M. Patel, J.W. Essex, F.M. Montevecchi, A. Redaelli, Anisotropic elastic network modeling of entire microtubules, Biophys. J. 99 (2011) 2190–2199.
[28] J.R. Lopez-Blanco, J.I. Aliaga, E.S. Quintana-Ortí, P. Chacon, IMODS: internal coordinates normal mode analysis server, Nucleic Acids Res. 42 (2014) W271–W276.
[29] J.R. Lopez-Blanco, P. Chacon, IMODFIT: efficient and robust flexible fitting based on vibrational analysis in internal coordinates, J. Struct. Biol. 184 (2013) 261–270.