CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **00**:1–12
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Out-of-Core Macromolecular Simulations on Multithreaded Architectures

José I. Aliaga[1], José M. Badía[1], Maribel Castillo[1], Davor Davidović[2*], Rafael Mayo[1], and Enrique S. Quintana-Ortí[1]

[1]*Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, 12.071–Castellón, Spain*
[2]*Institut Ruder Bošković, Centar za Informatiku i Računarstvo - CIR, 10000–Zagreb, Croatia*

SUMMARY

We address the solution of large-scale eigenvalue problems that appear in the motion simulation of complex macromolecules on multithreaded platforms, consisting of multicore processors and possibly a graphics processor (GPU). In particular, we compare specialized implementations of several high-performance eigensolvers that, by relying on disk storage and out-of-core (OOC) techniques, can in principle tackle the large memory requirements of these biological problems, which in general do not fit into the main memory of current desktop machines. All these OOC eigensolvers, except for one, are composed of compute-bound (i.e., arithmetically-intensive) operations, which we accelerate by exploiting the performance of current multicore processors and, in some cases, by additionally off-loading certain parts of the computation to a GPU accelerator. One of the eigensolvers is a memory-bound algorithm, which strongly constrains its performance when the data is on disk. However, this method exhibits a much lower arithmetic cost compared with its compute-bound alternatives for this particular application. Experimental results on a desktop platform, representative of current server technology, illustrate the potential of these methods to address the simulation of biological activity.

Received . . .

KEY WORDS: Macromolecular motion simulation, eigenvalue problems, out-of-core computing, multicore processors, GPUs.

1. INTRODUCTION

Coarse-grained models (CGM) combined with normal mode analysis (NMA) have recently arisen as a powerful approach to simulate biological activity at molecular level for extended time scales [5, 6, 21]. IMOD [16], for example, is a multipurpose tool chest that exploits the advantage of NMA formulations in internal coordinates (ICs), while extending them to cover multi-scale modeling [10, 15, 19]. Despite the reduction in degrees of freedom offered by ICs and reduced

*Correspondence to: Davor Davidović. Institut Ruder Bošković, Centar za Informatiku i Računarstvo - CIR, 10000–Zagreb, Croatia E-mail: ddavid@irb.hr.

approximations, the diagonalization step, and *the associated dense eigenproblem*, remain as the major computational bottleneck of this approach, especially for the simulation of large molecules.

In this paper we address the solution of large-scale eigenproblems arising in macromolecular simulation on current multithreaded desktop platforms, equipped with one or more multicore processors and, possibly, a graphics processing unit (GPU). In the target scenario, the data matrices are too large to fit into the graphics accelerator memory, or even the main memory of the platform, and therefore they have to be stored and moved from/to disk in pieces. Specialized algorithms are consequently required, which rely on so-called out-of-core (OOC) techniques to hide the disk latency when possible. Once a block of data is in the main memory, if the computation involving it proceeds on the GPU, similar techniques are applied to hide the latency due to the data transfers between main memory and the accelerator's memory via the PCI bus. Hereafter, an *OOC-GPU algorithm* refers to a method that performs most of its computations on the GPU, but operates with an amount of data that exceeds the capacity of the accelerator memory and, therefore, has to reside on the main memory of the system. The algorithm is therefore an OOC method from the GPU perspective. A true OOC algorithm is a method where the data do not fit into the main memory either, and have to be stored on disk.

The solution of large-scale generalized symmetric definite eigenproblems has been pursued, e.g., in [1, 4, 13, 17], on different types of high-performance platforms, for problems that fit in-core. (In particular, either in the GPU memory for those algorithms that operate with graphics accelerators, or in the main memory for those solutions that only exploit the multicore processors of the platform.) In [2] we addressed the solution of eigenproblems arising in the simulation of collective motions of macromolecular complexes using multicore platforms equipped with GPUs, making the following contributions compared to previous work:

- In [2], we assumed that the problem fitted into the main memory of the system but not into the accelerator memory. We then designed tailored OOC-GPU algorithms that, by leveraging OOC techniques [22], off-loaded the bulk of their computations to the GPU while amortizing the cost of data transfers between the GPU and the main memory with a sufficiently large number of floating-point arithmetic operations (flops).
- One of our algorithms was the first OOC-GPU implementation to employ spectral divide-and-conquer (SD&C) based on the polar decomposition [18]. We enhanced this algorithm with simple yet effective *ad-hoc* splitting strategies to reduce the number of SD&C iterations for our particular target application.
- We revisited an implementation of the two-stage reduction to tridiagonal form [8], where the first stage is also an OOC-GPU code while the subsequent stage operates on a much reduced compact matrix that fits in-core in the GPU memory.
- We compared these two OOC-GPU approaches using relatively large macromolecules [17].

In this paper, we extend our previous work [2] making the following original contributions:

- We consider macromolecular complexes involving matrices that do not fit into the main memory of the platform and, thus, require to operate with data on disk via true OOC methods.
- For the SD&C and two-stage algorithms, our results are based on the asymptotic performance observed for the OOC-GPU implementations evaluated in [2]. Following the conclusions in [20], due to the nature of the operations underlying all these algorithms (basically,

DOI: 10.1002/cpe

orthogonal factorizations and other simpler level-3 BLAS kernels), we can assume that these performance rates can be maintained when the data are on disk instead of main memory (i.e., they are encoded as OOC methods instead of simply OOC-GPU implementations). Therefore, we estimate the execution time on much larger problems using these values.

- We include in the comparison two recent in-core eigensolvers based on the two-stage approach [4, 13], with a considerably reduced computational cost, making the same considerations in order to extrapolate their execution time to an OOC scenario.
- We implement an OOC eigensolver based on a Krylov subspace iteration that operates with data on disk. For this memory-bound method, the disk bandwidth constraints the performance of the solver, so that the question is whether this is compensated by the much lower computational cost of this approach (quadratic per iteration vs cubic on the problem dimension for all the previous solvers), when the number of eigenvalues that are required is very small.
- We compare all these solvers using several datasets representative of much larger macromolecular complexes [17].

The rest of the paper is structured as follows. In Section 2 we briefly discuss the solution of generalized eigenproblems. In Section 3 we review the OOC implementation of the Krylov subspace-based method. The eigensolvers are evaluated next, in Section 4, using a collection of cases from biological sources. Finally, we close the paper in Section 5 with a few remarks.

2. SYMMETRIC DEFINITE EIGENPROBLEMS

The eigenproblem that has to be solved in the diagonalization step of CGM-NMA is given by

$$AX = BX\Lambda, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times n}$ correspond, respectively, to the Hessian and kinetic matrices that model the dynamics of the macromolecular complex, $\Lambda \in \mathbb{R}^{s\times s}$ is a diagonal matrix with the s sought-after eigenvalues (or modes) of the matrix pair (A, B), and $X \in \mathbb{R}^{n\times s}$ contains the corresponding unknown eigenvectors [11]. Furthermore, when dealing with large macromolecules, A, B are both dense symmetric positive definite, $n \geq$ 10,000, and typically only the $s \approx 100$ smallest eigenpairs in magnitude (i.e., the eigenvalues and eigenvectors for the low energy modes) are required.

All the eigensolvers considered next start by transforming the generalized eigenproblem (1) into an standard case, to then obtain the eigenpairs of the original problem from this compact form. Specifically, these methods initially compute the Cholesky factorization $B = U^T U$, where $U \in \mathbb{R}^{n\times n}$ is upper triangular [11], to then tackle the *standard* symmetric eigenproblem

$$CY = Y\Lambda \quad \equiv \quad (U^{-T}AU^{-1})(UX) = (UX)\Lambda, \tag{2}$$

where $C \in \mathbb{R}^{n\times n}$ is symmetric and $Y \in \mathbb{R}^{n\times s}$. Therefore, the standard eigenproblem (2) shares its eigenvalues with those of (1), while the original eigenvectors can be recovered from the simple back-transform $X := U^{-1}Y$. The initial Cholesky factorization, the construction of $C := U^{-T}AU^{-1}$

DOI: 10.1002/cpe

in (2), and the triangular solve for X are known to deliver high performance on a large variety of HPC architectures, including multicore processors and GPUs, and their functionality is covered by current numerical libraries (e.g., LAPACK, `libflame`, ScaLAPACK, PLAPACK, etc.) including some OOC extensions (SOLAR, POOCLAPACK). Therefore, we will not consider these operations any further but, instead, focus on the more challenging solution of the standard eigenproblem (2). Given that all eigensolvers included in the comparison perform the same sequence of computations to transform the generalized problem into an standard case, as well as recover the generalized eigenvectors from this solution, we consider fair to omit these transformation stages from the comparison.

Among the possible solvers for the symmetric eigenproblem, in [2] we presented an OOC-GPU implementation of the SD&C method [18], an algorithm with a high computational cost but which mainly consists of matrix-matrix operations that naturally render it as an appealing candidate for OOC-GPU strategies/platforms. Concretely, the SD&C method roughly requires $6n^3$ flops per iteration and, for the biological applications studied in [2], 7 iterative steps, which yields a total approximate cost of $42n^3$ flops. It comes as no surprise then that the general conclusion from the experiments in [2], from the point of view of performance, is that the OOC-GPU implementation of the SD&C method is not competitive with other OOC-GPU methods, for moderate-scale problems that fit into the main memory of the platform but not into the GPU memory.

In [8] we introduced a practical OOC-GPU implementation of a two-stage reduction-based eigensolver which first transforms the matrix C from dense to band form, to then refine this intermediate matrix to tridiagonal form, and finally obtain the eigenvalues using the MR^3 tridiagonal eigensolver [7, 9]. There, we demonstrated how, by carefully orchestrating the PCI data transfers between host and accelerator, in-core performance is maintained for the solution of large-scale generalized and standard eigenproblems via OOC-GPU algorithms. Provided the intermediate bandwidth is chosen large enough, this method casts most of its computations during the reduction to band form in terms of efficient BLAS-3, at the expense of a considerable computational cost. Specifically, $4n^3/3$ flops are required to obtain the band matrix and a lower-order amount for the subsequent refinement step. However, due to this double-step, recovering the original eigenvectors adds $2n^3$ flops to the method.

In [4], a variant of the two-stage method is proposed that exhibits significantly lower computational cost compared with the conventional algorithm. The implementation described there, and the associated library ELPA, targets clusters of nodes equipped with multicore processors, and assumes that the problem data are distributed among the main memories of the system nodes. In [13] and the MAGMA library the same variant is implemented on a desktop platform with multiple GPUs, distributing the problem data among the accelerators' memories. Thus, these two solutions differ from [8] not only in the variant of the two-stage method that they implement, but also in that they correspond to in-core algorithms (from the perspective of main memory and GPU memory in the case of ELPA and MAGMA, respectively).

An alternative for the solution of symmetric eigenproblems is given by the Krylov subspace-based methods [11]. This approach employs a Lanczos-based procedure [11] to iteratively construct an orthogonal basis of the Krylov subspace associated with C. Upon convergence, this procedure yields a tridiagonal square matrix $\tilde{T}$, of reduced dimension $m \geq 2s$, whose eigenvalues approximate those of C, and a matrix $\tilde{V}$, of conformal dimension, that contains the Krylov vectors. Given that in

DOI: 10.1002/cpe

practice $m \ll n$, obtaining the eigenvalues and eigenvectors from $\tilde{T}$ and $\tilde{V}$ adds a minor theoretical cost to the computations. For symmetric matrices, this process often exhibits fast convergence (especially in case the extremal eigenvalues of C are well separated), low computational cost per iteration (in general, $\mathcal{O}(n^2)$ flops) and, moreover, does not require an appreciable additional storage space. As our application requires the smallest s eigenvalues (in magnitude) of the pair (A, B), and they are close to zero, we compute these from the largest s eigenvalues (in magnitude) of the correlated problem (B, A) which, for these particular problems, have the appealing property of being extremal and well-separated, ensuring fast convergence of the iteration.

Practical in-core implementations of the Krylov subspace-based methods on multicore processors clearly outperform the previous compute-bound eigensolvers when the number of desired eigenvalues/eigenvectors is small relative to the problem size [1], even when the problem is dense and the compute-bound eigensolvers rely on a hardware accelerator. On the other hand, from an OOC viewpoint, the major drawback of the Krylov subspace-based methods is that they cast a major part of their computations in terms of the symmetric matrix-vector product (MVP). In particular, for a matrix of size $n \times n$, this kernel roughly performs $2n^2$ flops with $n^2/2$ numbers (only the upper or lower triangular part of the matrix needs to be accessed) clearly being a memory-bound operation as the rate of computation to data is only $2n^2/(n^2/2) = 4$ flops/number. Therefore, an implementation that operates with data stored on disk is strongly limited by the bandwidth to this device and will only attain a low fraction of the architecture's peak performance. For example, for a platform equipped with a disk that delivers a sustained bandwidth around 300 MBytes/s, we can expect to attain a maximum performance rate of just 300 MBytes/s $\times$ 1 double precision (DP) number/8 bytes $\times$ 4 flops/DP number = 150 DP MFLOPS (millions of flops per second), compared with the hundreds of DP GFLOPS that are easily delivered by current multicore processors. At this point, it is worth clarifying that the use of double- and triple-buffering techniques can only help to approximate the maximum performance dictated by the disk bandwidth in practice, but not exceed it. Furthermore, the integration of a GPU in the platform cannot be expected to deliver any increase in performance for Krylov methods that operate with OOC data, as the bottleneck is the disk bandwidth.

Therefore, the question we investigate is whether the cost advantage of the memory-bound Krylov subspace-based method is sufficient to compensate its much lower performance compared to the compute-bound eigensolvers when the data are stored on disk.

3. IMPLEMENTATION OF THE OOC EIGENSOLVERS

OOC-GPU eigensolvers based on the two-stage and SD&C algorithms were introduced in [8] and [2], respectively. Both implementations operate with data matrices that reside in the main memory but do not fit into the accelerator memory. Adding one more layer in the memory hierarchy of these eigensolvers, so that the data reside on disk instead of main memory, and are moved back and forth between there and the GPU, is conceptually equivalent. Furthermore, for compute-bound operations like those present in these two algorithms, in [20] we showed that the disk latency can be perfectly hidden via a careful organization of the data movements, analogous to that performed between the GPU and the main memory. Therefore, we can reasonably expect that the OOC-GPU

DOI: 10.1002/cpe

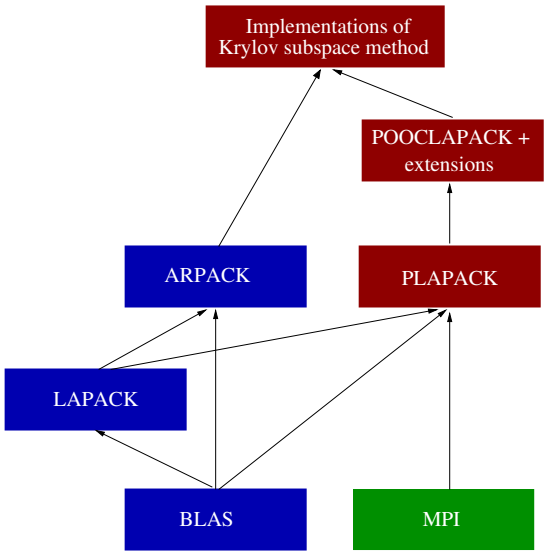

Figure 1. Libraries and routines for the construction of the Krylov subspace-based eigensolvers. (red: message-passing; blue: sequential/multithreaded; green: communication). All the OOC functionality is embedded inside POOCLAPACK and our own extensions to this libraries.

eigensolvers maintain their performance when operating with data that effectively resides on disk. A similar reasoning applies to the in-core two-stage variants implemented in ELPA and MAGMA.

In this section we present the new OOC implementation of the iterative eigensolver based on the computation of a Krylov subspace of C, which truly operates with data on disk. Our OOC Krylov subspace-based solver computes the main operations of the method using existing OOC libraries that orchestrate the access to the matrices on secondary storage as well as efficient dense linear algebra and communication libraries. To that end we leveraged the OOC implementation of the Cholesky factorization in POOCLAPACK [12], and extended this library with new OOC kernels for the triangular system solve with multiple right-hand sides (including several algorithmic variants of this operation), as well as the dense matrix-vector product (for symmetric matrices). While the use of an MPI-based library to tackle these problems on a desktop server introduces a certain overhead, this is clearly compensated by the advantages intrinsic to software composability and reusability. Furthermore, we employed ARPACK for the iterative construction of the Krylov subspace as well as several other complementary libraries. Figure 1 shows the organization of the different libraries and routines used/developed to implement our Krylov subspace-based eigensolver.

3.1. PLAPACK and POOCLAPACK

PLAPACK [23] is a parallel message passing library for dense linear algebra that includes a collection of routines to solve linear systems of equations, eigenvalue problems, etc. The library mirrors sequential dense linear algebra algorithms by adopting an "object-based" approach. Matrices and vectors are distributed among the processes that communicate using MPI primitives, but the distribution and indexing details are hidden to the user by means of opaque objects.

DOI: 10.1002/cpe

From the implementation point of view, we introduced several small modifications (patches) in the contents of PLAPACK (release 3.2) to be able to operate with very large dense matrices, both in-core and out-of-core. The most significant of these changes was the substitution of a considerable number of 32-bit integers (`int`) that were used for indexing purposes internally to the codes by 64-bit numbers (`long int`).

POOCLAPACK is the OOC extension of PLAPACK to solve linear algebra problems using algorithm-by-tiles. This kind of algorithms extends the traditional concept of algorithms-by-blocks used in in-core libraries by defining the *tile* (a square-like large block) as the unit of transference between (main) memory and disk. Thus, the library implicitly makes a hierarchical usage of two concepts: a matrix is composed by tiles of dimension $t \times t$, with each tile consisting of several blocks of size $b \times b$ (with $b \ll t$). Tiles are transferred from/to disk by an algorithm-by-tiles while computation is performed on the blocks that compose an in-core tile (i.e., a tile that resides in memory) by an algorithm-by-blocks from PLAPACK.

POOCLAPACK comprises several OOC routines, including a kernel to obtain the Cholesky factorization of a symmetric positive definite matrix (routine `PLA_OOC_Chol`). Additionally, we applied techniques analogous to those in POOCLAPACK to develop OOC kernels for the symmetric matrix-vector product (`PLA_OOC_Symv`) and the solution of triangular linear systems with multiple right-hand sides (`PLA_OOC_Trsm`), including several algorithmic variants of the latter.

3.2. ARPACK

The eigenpairs were computed using ARPACK (ARnoldi PACKage) [14]. Specifically, we applied the implicit restarted variant of the Lanczos process in this package. While there exists also a parallel version of this library, PARPACK [3], the latter uses a data distribution different from that of PLAPACK. Some initial experiments revealed that the redistribution of data required at each iteration by the interaction between PARPACK and PLAPACK was quite expensive. On the other hand, for large eigenproblems like those tackled in this work, the total computational cost of the ARPACK routines is negligible compared with that of the POOCLAPACK/PLAPACK kernels, so that we decided to use the sequential version of the ARPACK library. We emphasize that the redistributions involved in the interaction between PLAPACK and ARPACK were performed using PLAPACK routines. These routines allow us to gather (all-to-one) and scatter (one-to-all) objects of the library (`PLA_Obj`) from and to C-style matrices and vectors, resulting in a negligible cost compared to the rest of the algorithm.

3.3. The Krylov subspace-based eigensolver

A high-level (simplified) description of the eigensolver is given in Listing 1. The code assumes that the matrix pair (A, B) is passed as two objects (`PLA_Obj`), `A` and `B`, initially stored on disk following the standard data layout for POOCLAPACK. All data movement between disk and main memory is hidden (performed) inside the OOC routines `PLA_OOC_Chol`, `PLA_OOC_Trsm`, and `PLA_OOC_Symv`. Internally, these kernels invoke routines from PLAPACK to perform the appropriate computations on the in-core data (tiles) in parallel. We emphasize that this solver was actually invoked with the "swapped" pair (B, A) instead of (A, B) so as to compute the largest s eigenvalues of the former problem (indicated by parameter "`LM`" in the calls to routines `dsaupd`

DOI: 10.1002/cpe

and `dseupd`) and accelerate the convergence. The desired s eigenvalues (the smallest ones) are then obtained by inverting those that were actually computed.

```
// Krylov subspace-based eigensolver. Variant with explicit construction of C
// Inputs: Matrix pair (A,B), both of dimension n x n; number of requested eigenvalues s
// Outputs: eigenvalues in vector d, of length n, and eigenvectors in X of dimension n x s

   int    ido  = 0,       iter = 0,   info  = 0;
   double tol = 1.0e-12,  done = 1.0, dzero = 0.0, *w = NULL, *z = NULL;

// 1.  Transform the generalized eigenproblem into the corresponding standard case
// 1.1 Factor B = U^T U; overwrite upper triangular part of B with the Cholesky factor U
   PLA_OOC_Chol( "Upper", &n, B, ... ); // Cholesky factorization; n^3/3 flops

// 1.2 C := U^-T A U^-1; overwrite A with C. Upper triangle of B contains the Cholesky factor
   PLA_OOC_Trsm( "Right", "Upper", "No_Transpose", "No_Unit",
                 &n, &n, &done, B, ..., A, ... ); // Triangular solve; n^3 flops
   PLA_OOC_Trsm( "Left", "Upper", "Transpose", "No_Unit",
                 &n, &n, &done, B, ..., A, ... ); // Triangular solve; n^3 flops

// 2.  Krylov subpace iteration for the variant
// 2.0 Generate w_0; initial guess returned in WORKD
   dsaupd( &ido, ..., &n, 'LM', &s, &tol, ..., &m, V, ..., IPNTR, WORKD, ..., &info );

   while (ido != 99) {
      // 2.1 Form z_k+1 := C w_k; C is in A
      w = WORKD + IPNTR[0] - 1;
      z = WORKD + IPNTR[1] - 1;
      PLA_OOC_Symv( "Upper",
                    &n, &done, A, ..., w, ..., &dzero, z, ... ); // Symmetric matrix-vector
                                                                 // product; n^2 flops

      // 2.2 z_k+1 -> w_k+1; w and z in WORKD
      dsaupd( &ido, ..., &n, 'LM', &s, &tol, ..., &m, V, ..., WORKD, ..., &info );
   }

   // 2.3 S,V -> Lamba,Y; S in WORKD, eigenvalues in d, Ritz vectors in X
   dseupd( ..., d, X, ..., &s, 'LM', &tol, ..., &m, V, ..., IPNTR, WORKD, ..., &info );

// 3.  Back-transform to the generalized eigenproblem solution
// 3.1 X := U^-1 Y. Overwrite Ritz vectors in X with eigenvectors of the problem
   PLA_OOC_Trsm( "Left", "Upper", "No_Transpose", "No_Unit",
                 &n, &s, &done, B, ..., X, ... ); // Triangular solve; n^2s flops
```

Listing 1: Implementation of Krylov subspace-based eigensolver using POOCLAPACK and ARPACK (simplified).

Parallelism was evaluated for several configurations which placed c/h MPI ranks in the node, with c being the number of physical cores per node, and h the number of local threads for LAPACK and BLAS, with $1 \leq h \leq c$.

4. EXPERIMENTAL RESULTS

4.1. Macromolecular benchmarks

In the following experiments, we will consider the biological cases listed in Table I. The dimension of the eigenproblem associated with the `MT` macromolecule can be scaled to a variety of cases, e.g., from only 12,469 up to 908,954. For details on the biological significance of these cases, see [17].

DOI: 10.1002/cpe

Acronym	Name	PDB id.	n
MT	Microtubule 13:3	1tub	Scalable
ER	Eucariotic Ribosome*	2xsy,2xtg	36,488
CCMV	Cowpea chlorotic mottle virus	1cwp	56,454
VP_A	Vault protein	2zuo,2zv4,2zv5	75,518
VP_B			119,486
$HK97_A$	Hong Kong 97 virus Head II	2ft1	100,325
$HK97_B$			149,231

Table I. Benchmark of large-scale biological macromolecules. *In this case all heavy atoms were taken into account in the rest a C_a model was used.

4.2. Target platform and software

All the experiments were performed using IEEE double-precision arithmetic on a server equipped with two Intel Xeon E5520 quad-core processors (for a total of 8 cores @ 2.27 GHz), 48 GBytes of RAM, and connected to an NVIDIA Tesla C2050 GPU (2.6 GBytes of memory, ECC on). The local disk used in the computations of OOC routines was an Intel SSD 910 (400 GBytes of capacity).

The following eigensolvers were included in the experimentation:

- In-core compute-bound algorithms:

  `ELPA` [†] (Eigenvalue SoLvers for Petaflop-Applications, release 2013.11 v8). This in-core two-stage eigensolver exploits the multicore processors of the platform but does not offload any part of the computation to the GPU. Parallelism was exploited using 8 MPI ranks.

  `MAGMA` [‡] (Matrix Algebra on GPU and Multicore Architectures, release 1.4.1). This hybrid CPU-GPU two-stage eigensolver exploits both the multicore processors and the GPU.

- OOC-GPU compute-bound algorithms:

  `Two-stage`. A GPU implementation of the conventional two-stage algorithm introduced in [8] with several enhancements to reduce PCI-transfers.

  `SD&C`. The GPU implementation of the spectral divide-and-conquer based on the polar decomposition [2], with a splitting technique *ad-hoc* for macromolecular simulation.

- OOC memory-bound algorithm:

  `Krylov`. The Krylov subspace-based solver described in detail in Section 3, with the matrix-vector products proceeding in the multicore processors. Depending on the problem size, we configured this solver to employ a single MPI rank/8 OpenMP threads, 2 MPI ranks/4 OpenMP threads, or 8 MPI ranks with a single thread each. In the calls to ARPACK routine `dsaupd`, we set the following parameters: `nev`=100 (number of eigenvalues), `ncv`=250 (number of vectors of the orthogonal matrix V, `tol`=1.0e-12 (stopping criterion), and `iparam[2]`=100 (maximum number of Arnoldi update iterations allowed).

In all cases, the codes were compiled with Intel `icc` (`composerxe 2011_sp1.9.293`), linked to the implementation of LAPACK/BLAS implementation in Intel MKL (v10.3 update 9) and

[†] `http://elpa.rzg.mpg.de/`.
[‡] `http://icl.cs.utk.edu/magma/`.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

NVIDIA CUBLAS (v5.0). The MPI codes (`ELPA` and `Krylov`) were linked to `mvapich2 1.8a1p1`.

For all the compute-bound eigensolvers except ELPA, the results include the cost of transferring the input data and the results between main memory and GPU. (In ELPA there is no transfer between main memory and the GPU as this library is GPU-oblivious.) For the memory-bound Krylov subspace-based method, the cost of moving the data to/from disk is included in the reported results, but there is no movement of data between main memory and the GPU, because all operations proceed in this case the multicore processors.

The dimension of the largest problem that can be tackled by each one of these eigensolvers depends on the memory requirements of the method (software) and also on the "target memory level" (hardware). For the in-core ELPA and MAGMA, the hardware limit is determined by the capacity of the main memory and the GPU memory, respectively; for both OOC-GPU implementations the limit is dictated by the main memory; and for the Krylov eigensolver it is the capacity of the disk. For the latter implementation, the performance results reported next correspond to actual execution times, while for all other four solvers, we perform an extrapolation to estimate the execution time for the larger benchmark cases. In particular, these four eigensolvers feature a cubic computational cost on the problem dimension, requiring $O(n^3)$ flops. For the in-core and OOC-GPU algorithms, we ideally assume that the data transfers from/to disk (and, for MAGMA, also the PCI-transfers from/to the GPU) add no overhead to the algorithm, so that we extrapolate the performance of the largest problem size that could be actually tackled to elaborate the estimations. For example, consider the MAGMA in-core implementation. Given the memory requirements of this two-stage GPU eigensolver and the 2.6 GBytes of the target GPU, the largest macromolecule that could be tackled with the corresponding routine from this library was `MT` with $n = 12,469$, requiring a time of $t = 82.16$ s. The estimated execution time of this solver for a benchmark macromolecule of dimension $\hat{n}$ was then simply extrapolated as $t \times (\hat{n}/n)^3$.

4.3. Performance evaluation

Table II reports the execution times of the different eigensolvers using the collection of biological benchmarks. For the smaller cases, time is reported in seconds while for the larger ones we employ hours. We note that up to the `CCMV` benchmark (n=56,454), the problem fits into the main memory of the platform when solved via the `Krylov` code. This is transparently exploited by the operating system which actually stores the data there even though our OOC implementation attempts to retrieve the matrix from disk via the corresponding file system calls. From that problem dimension up, the data no longer fits into the main memory and therefore have to be read from disk at each iteration of the Krylov subspace-based solver. In any case, these results clearly reveal this solver as highly competitive against all other four alternatives for the largest cases, especially as the problem dimension grows beyond n=100,000.

5. CONCLUDING REMARKS

We have presented a new and competitive OOC eigensolver, based on the computation of Krylov subspaces, for the solution of generalized symmetric eigenproblems arising in macromolecular

DOI: 10.1002/cpe

Macromolecule	n	Execution time in seconds				
		in-core MAGMA	in-core ELPA	OOC-GPU Two-stage	OOC-GPU SD&C	OOC Krylov
MT	12,469	82.16	95.42	148.12	350.26	41.37
	15,588	160.52	168.49	270.31	623.42	58.90
	20,266	352.75	336.01	535.96	1,154.05	98.36
	21,824	440.52	406.89	652.62	1,653.41	108.02
	24,943	657.82	581.58	907.42	2,402.82	135.49
	29,622	1,101.60	974.11	1,475.61	3,871.20	187.03
	31,178	1,284.43	1,135,82	1,701.75	4,521.39	210.29
		Execution time in hours				
ER	36,488	0.57	0.50	0.75	2.01	0.08
CCMV	56,454	2.11	1.85	2.78	7.44	0.18
VP	75,518	5.05	4.43	6.65	17.82	0.93
$HK97_A$	100,325	11.85	10.39	15.59	41.78	10.39
VP_B	119,486	20.02	17.56	26.34	70.58	14.71
$HK97_B$	149,231	38.99	34.21	51.31	137.51	23.14
MT	195,805	88.09	77.27	115.90	310.62	40.87

Table II. Execution time of the eigensolvers. The results with black print are actual times; the results in blue print are estimated by extrapolating the largest in-core case that could be solved (e.g., for MAGMA, n=12,469); finally, the results in red print are extrapolated from the largest case that could be solved using the OOC-GPU implementation (e.g., for SD&C, n=31,178).

motion simulation. Additionally, we have compared the performance of this implementation against two OOC-GPU algorithms, based on the two-stage reduction to tridiagonal form and a recent spectral divide-and-conquer approach for the polar decomposition, and two in-core solvers based on a variant of the two-stage approach.

The Krylov subspace-based eigensolver presents a much lower theoretical cost than the in-core (ELPA and MAGMA) and OOC-GPU alternatives, but this method casts most of its computations in terms of a memory-bound symmetric matrix-vector product. Therefore, when operating with OOC data this method cannot benefit e.g. from the use of a hardware accelerator, as its performance is intrinsically limited by the disk bandwidth. On the other hand, the implementations of the spectral divide-and-conquer and two-stage counterparts (including ELPA and MAGMA) attain high performance by carefully amortizing the cost of data transfers with a large number of floating-point arithmetic operations so that the dimension of the macromolecular problems that can be tackled is not constrained by the capacity of the GPU memory nor the main memory.

Our experiments on an desktop platform with two Intel Xeon multicore processors and an NVIDIA “Fermi” GPU, representative of current server technology, illustrate the potential of all these methods to address the simulation of biological activity. These results reveal the asymptotic superiority of the Krylov approach for this application over all other libraries/methods as the problem size increases.

ACKNOWLEDGEMENTS

This research was supported by the CICYT project TIN2011-23283 of the *Ministerio de Economía y Competitividad*, the Fundación Caixa-Castelló/Bancaixa contract no. P1-1B2011-18 and FEDER.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

REFERENCES

1. J. Aliaga, P. Bientinesi, D. Davidović, E. Di Napoli, F.D. Igual, and E. S. Quintana-Ortí. Solving dense generalized eigenproblems on multi-threaded architectures. *Appl. Math. & Comp.*, 218(22):11279–11289, 2012.
2. J. I. Aliaga, D. Davidović, and E. S. Quintana-Ortí. Out-of-core solution of eigenproblems for macromolecular simulations on GPUs. In *Proc. 10th Int. Conf. Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science. Springer, 2013. To appear.
3. ARPACK project home page. `http://www.caam.rice.edu/software/ARPACK/`.
4. T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krmer, B. Lang, H. Lederer, and P.R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Computing*, 37(12):783 – 794, 2011.
5. G. S. Ayton and G. A. Voth. Systematic multiscale simulation of membrane protein systems. *Curr. Opin. Struct. Biology*, 19(2):138–44, 2009.
6. I. Bahar, T. R. Lezon, A. Bakan, and I. H. Shrivastava. Normal mode analysis of biomolecular structures: functional mechanisms of membrane proteins. *Chem. Rev.*, 110(3):1463–97, 2010.
7. P. Bientinesi, I. S. Dhillon, and R. van de Geijn. A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations. *SIAM J. Sci. Comput.*, 27(1):43–66, 2005.
8. D. Davidović and 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*, pages 442–449, 2012.
9. Inderjit S. Dhillon and Beresford N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications*, 387:1 – 28, 2004.
10. N. Go, T. Noguti, and T. Nishikawa. Dynamics of a small globular protein in terms of low-frequency vibrational modes. *Proc. Natl. Acad. Sci.*, 80(12):3696–3700, 1983.
11. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996.
12. Brian C. Gunter, Wesley C. Reiley, and Robert A. van de Geijn. Parallel out-of-core cholesky and qr factorization with pooclapack. In *IPDPS*, page 179. IEEE Computer Society, 2001.
13. Azzam Haidar, Raffaele Solcà, Mark Gates, Stanimire Tomov, Thomas Schulthess, and Jack Dongarra. Leading edge hybrid multi-gpu algorithms for generalized eigenproblems in electronic structure calculations. In Julian Martin Kunkel, Thomas Ludwig, and HansWerner Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 67–80. Springer Berlin Heidelberg, 2013.
14. R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users guide: Solution of large scale eigenvalue problems by implicitly restarted arnoldi methods., 1997.
15. M. Levitt, C. Sander, and P. S. Stern. Protein normal-mode dynamics: trypsin inhibitor, crambin, ribonuclease and lysozyme. *J. Mol. Biology*, 181(3):423–47, 1985.
16. J. R. Lopez-Blanco, J. I. Garzon, and P. Chacon. iMOD: multipurpose normal mode analysis in internal coordinates. *Bioinformatics*, 27(20):2843–50, 2011.
17. J. R. López-Blanco, R. Reyes, J. I. Aliaga, R. M. Badia, P. Chacón, and E. S. Quintana. Exploring large macromolecular functional motions on clusters of multicore processors. *J. Comp. Phys.*, 246:275–288, 2013.
18. Y. Nakatsukasa and N. J. Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD. Technical Report 2012.52, Manchester Inst. Math. Sci., The University of Manchester, 2012.
19. T. Noguti and N. Go. Dynamics of native globular proteins in terms of dihedral angles. *J. Phys. Soc. Jpn.*, 52(9):3283–3288, 1983.
20. G. Quintana-Ortí, F. D. Igual, M. Marqués, E. S. Quintana-Ortí, and Robert A. Van de Geijn. A run-time system for programming out-of-core matrix algorithms-by-tiles on multithreaded architectures. *ACM Trans. Math. Softw.*, 38(4):25:1–25:25.
21. L. Skjaerven, S. M. Hollup, and N. Reuter. Normal mode analysis for proteins. *J. Mol. Struct. (Theochem)*, 898(1-3):42–48, 2009.
22. Sivan Toledo. A survey of out-of-core algorithms in numerical linear algebra. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.
23. Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

DOI: 10.1002/cpe