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 (one or more) multicore processors and possibly a graphics processor (GPU). In particular, we compare specialized implementations of three high-performance eigensolvers that rely on disk storage and out-of-core (OOC) techniques to tackle the large memory requirements of these biological problems, which in general do not fit into the main memory of current desktop machines. Two of the OOC eigensolvers are composed of compute-bounded operations and we enhance their performance by leveraging hybrid CPU-GPU routines that off-load the arithmetically-intensive parts of the algorithms to a GPU accelerator. The third OOC eigesolver is a memory-bounded algorithm, which strongly constrains its performance when the data is on disk. However, this eigensolver exhibits a much lower arithmetic cost compared with its compute-bounded alternatives for this particular application. Experimental results on a desktop platform with two Intel Xeon multicore processors and an NVIDIA "Fermi" GPU, 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) has recently arisen as a powerful approach to simulate biological activity at molecular level for extended time scales [4, 5, 20]. iMOD [14], for example, is a multipurpose tool chest, based on the seminal works of Go and others [9, 18, 13], that exploits the advantage of NMA formulations in internal coordinates (ICs), while extending them to cover multi-scale modeling. Despite the reduction in the degrees of

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

freedom offered by ICs and reduced approximations, the diagonalization step, and *an associated dense eigenproblem*, remains the major computational bottleneck of this approach, especially for simulations involving large molecules.

In [2] we addressed the solution of generalized symmetric definite eigenproblems arising in the simulation of collective motions of macromolecular complexes using multicore platforms equipped with graphics processing units (GPUs), making the following contributions compared to [1, 15, 16]:

- The eigenproblems associated with this particular application involve dense matrices that are, in general, too large to fit into the memory of the GPU and, sometimes, even the main memory of the server. In [2] we considered the first case; i.e., we assumed that the problem fitted in the main memory of the system but not in the accelerator memory. We then designed specialized algorithms that, by adopting out-of-core (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 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 [17]. 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 [7], 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.
- We compared these two OOC-GPU approaches using relatively large macromolecules [15].

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

- We consider macromolecular complexes that involve matrices that do not fit into the main memory of the platform and, therefore, require to operate with data on disk.
- For the SD&C and two-stage OOC-GPU algorithms, our results are based on the asymptotic GFLOPS (i.e., billions of flops/sec.) rates observed for the implementations evaluated in [2]. Following the conclusions from [19], and due to the nature of the operations underlying these two algorithms (basically, orthogonal factorizations and other simpler level-3 BLAS kernels), in these two cases we assume that these performance rates can be maintained when the data is on disk instead of main memory, and estimate the execution time on much larger problems using these values.
- We implement an OOC eigensolver based on a Krylov subspace iteration that operates with data on disk. For this memory-bounded method, the disk bandwidth constraints the performance of the solver, so that the question we address is whether this is compensated by the much lower computation cost of this approach, even for dense eigensolvers, when the number of eigenvalues that have to be computed is very small.
- We compare the two OOC-GPU approaches with the OOC Krylov subspace-based algorithm using several datasets representative of much larger macromolecular complexes [15].

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 three eigensolvers are evaluated next, in Section 4, using a collection

Prepared using cpeauth.cls
DOI: 10.1002/cpe

of cases from biological sources. Finally, we close the paper in Section 5 with a few concluding 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 capture 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 [10]. 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 (i.e., the eigenvalues and eigenvectors associated with low energy modes) are required.

All the eigensolvers considered next start by explicitly transforming the generalized eigenproblem (1) into an standard one, 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 [10], to then tackle the equivalent *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 $Y \in \mathbb{R}^{n\times s}$. Here, 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 $C := U^{-T}AU^{-1}$ 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).

Among the conventional solvers for the symmetric eigenproblem, in [7] 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 [6, 8]. There, we demonstrated how, by carefully orchestrating the PCI data transfers between host and accelerator, in-core performance is maintained or even increased for the OOC solution of general large-scale eigenproblems on hybrid CPU-GPU platforms. 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 higher computational cost. Specifically, $8n^3/3$ 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 [2] we proposed an OOC-GPU implementation of the SD&C method [17], an algorithm with a much higher computational cost than the two-stage reduction-based approach describer earlier, but which also consists mainly of matrix-matrix operations that naturally render it as an appealing

DOI: 10.1002/cpe

candidate for OOC-GPU strategies/platforms. In particular, 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 its OOC-GPU two-stage counterpart, for moderate-scale problems that fit into the main memory of the platform but not in the GPU memory.

A third alternative for the solution of symmetric eigenproblems is given by the Krylov subspace-based methods [10]. This approach employs a Lanczos-based procedure [10] 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 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, low computational cost per iteration (in general, $\mathcal{O}(n^2)$ flops) and, moreover, does not require an appreciable additional storage space. Practical in-core implementations of these methods on multicore processors clearly outperform the previous two compute-bounded 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-bounded eigensolvers exploit 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 significant part of their computations in terms of the matrix-vector product (MVP). For a matrix of size $n \times n$, this kernel roughly performs $2n^2$ flops on n^2 numbers (i.e., a rate of computation to data of only $O(1)$), so that an implementation that operates with OOC data is intrinsically limited by data movement (i.e., it is memory-bounded) and will attain very low performance.

Therefore, the question to investigate is whether the cost advantage of Krylov subspace-based methods is sufficient to compensate their much lower performance compared to the compute-bounded eigensolvers when data is stored on disk.

3. IMPLEMENTATION OF THE OOC EIGENSOLVERS

OOC-GPU eigensolvers based on the two-stage and SD&C algorithms were introduced in [7] and [2], respectively. Both implementations operate with data matrices residing in the main memory, but which 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-bounded operations alike those present in these two algorithms, in [19] 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 expect that the implementations of the OOC-GPU eigensolvers maintain their performance when operating with data that effectively resides on disck.

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

DOI: 10.1002/cpe

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 [11], 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). 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.

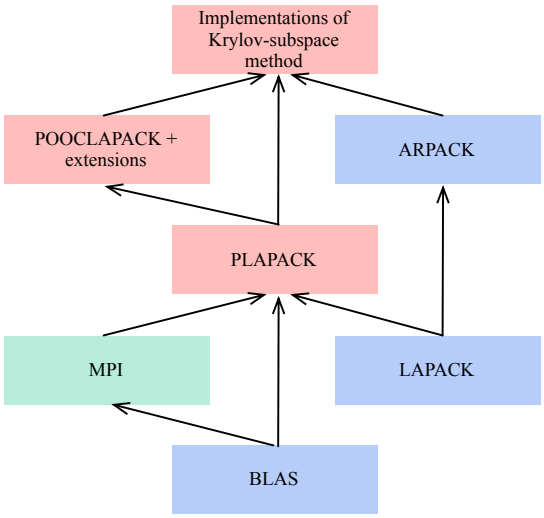

Figure 1. Libraries and routines for the construction of the Krylov subspace-based eigensolvers. (red: message-passing; blue: multithreaded; green: communication). All the OOC functionality is embedded inside POOCLAPACK and our own extensions to this libraries.

3.1. PLAPACK and POOCLAPACK

PLAPACK 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 [21], but the distribution and indexing details are hidden to the user by means of opaque objects.

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

DOI: 10.1002/cpe

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 features 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 underlying 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.

```
1  // Krylov subspace-based eigensolver. Variant with explicit construction of C
2  // Inputs: Matrix pair (A,B), both of dimension n x n; number of requested eigenvalues s
3  // Outputs: eigenvalues in vector d, of length n, and eigenvectors in X of dimension n x s
4
5     int    ido  = 0,   iter = 0,   info  = 0;
6     double dtol = 0.0, done = 1.0, dzero = 0.0;
7     double *w = NULL, *z = NULL;
8
9  // 1.  Transform the generalized eigenproblem into the corresponding standard case
10 // 1.1 Factor B = U^T U; overwrite upper triangular part of B with the Cholesky factor U
11
12    PLA_OOC_Chol( "Upper", &n, B, ... ); // Cholesky factorization; n^3/3 flops
13
14 // 1.2 C := U^-T A U^-1; overwrite A with C. Upper triangle of B contains the Cholesky factor
15
16    PLA_OOC_Trsm( "Right", "Upper", "No_Transpose", "No_Unit",
17                  &n, &n, &done, B, ..., A, ... ); // Triangular solve; n^3 flops
18    PLA_OOC_Trsm( "Left", "Upper", "Transpose", "No_Unit",
19                  &n, &n, &done, B, ..., A, ... ); // Triangular solve; n^3 flops
20
21 // 2.  Krylov subpace iteration for the variant
22 // 2.0 Generate w_0; initial guess returned in WORKD
23
24    dsaupd( ..., &ido, ..., &n, ..., &s, &dtol, ..., &m, Q, ..., IPNTR, WORKD, ..., &info );
25
26    while (ido != 99) {
27        // 2.1 Form z_k+1 := C w_k; C is in A
28
29        w = WORKD + IPNTR[0] - 1;
30        z = WORKD + IPNTR[1] - 1;
31        PLA_OOC_Symv( "Upper",
32                      &n, &done, A, ..., w, ..., &dzero, z, ... ); // Symmetric matrix-vector
33                                                                   // product; n^2 flops
34
35        // 2.2 z_k+1 -> w_k+1; w and z in WORKD
36
37        dsaupd( ..., &ido, ..., &n, ..., &s, &dtol, ..., &m, Q, ..., WORKD, ..., &info );
38    }
39    // 2.3 S,Q -> Lamba,Y; S in WORKD, eigenvalues in d, Ritz vectors in X
40
41    dseupd( ..., &s, ..., d, X, ..., &s, &dtol, ..., &m, Q, ..., IPNTR, WORKD, ..., &info );
42
43 // 3.  Back-transform to the generalized eigenproblem solution
44 // 3.1 X := U^-1 Y. Overwrite Ritz vectors in X with eigenvectors of the problem
45
46    PLA_OOC_Trsm( "Left", "Upper", "No_Transpose", "No_Unit",
47                  &n, &s, &done, B, ..., X, ... ); // Triangular solve; n^2s flops
```

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

DOI: 10.1002/cpe

3.2. ARPACK

The eigenpairs were computed using ARPACK (ARnoldi PACKage) [12]. Specifically, we applied the implicit restarted variant of the Lanczos process (implemented by routines `dsaupd` and `dseupd`) 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 note that the redistributions involved in the interaction between PLAPACK and ARPACK were realized 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.

Paralellism was exploited using two different approaches: pure-MPI and multithreaded. In the first one, we placed c MPI ranks in the node, with c being the number of physical cores, and set the number of threads for LAPACK and BLAS to one. In the second, we used a single MPI rank but set the number of threads for LAPACK and BLAS to c. Our experiments revealed that, for large-scale cases, the second approach was more efficient, and therefore we will only report results for this one.

4. EXPERIMENTAL RESULTS

All the experiments in this section were performed using IEEE double-precision arithmetic on a server equipped with two Intel Xeon E5520 quad-core processors (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, 1000MB/s for streamed reads, 750 MB/s for streamed writes). For the compute-bounded eigensolvers, i.e. SD&C and the two-stage reduction-based algorithm, the results include the cost of transferring the input data and the results between main memory and GPU (no disk is involved). For the memory-bounded 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 are performed in the multicore processors. In all cases, the codes were linked to NVIDIA CUBLAS (v5.0) and the BLAS implementation in GotoBLAS2 (v1.13).

Prepared using cpeauth.cls
DOI: 10.1002/cpe

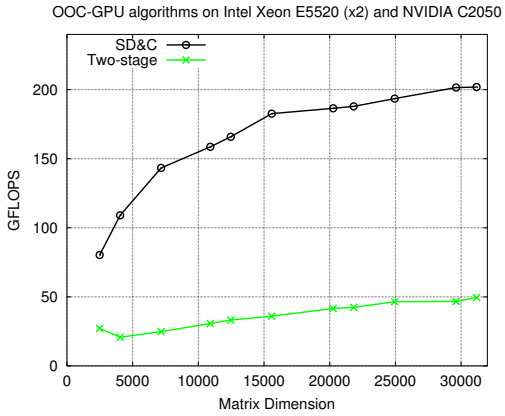

Figure 2. GFLOPS rate of the OOC-GPU eigensolvers applied to small to moderate-scale instances of the MT test case.

4.1. Performance of compute-bounded solvers

The goal of the first experiment is to estimate the performance of the OOC-GPU compute-bounded eigensolvers. For that purpose, we consider a scalable biological case, MT (see Table I), generated using IMOD, and tune it to generate problems that do not fit into the GPU but are "small" enough for the main memory. For matrix decompositions such as the QR factorization and other similar Level-3 BLAS-based kernels, like those appearing in the compute-bounded eigensolvers, disk latency can be mostly hidden by overlapping it with computation, even in platforms equipped with GPU accelerators [19]. Therefore, we expect these results to carry over to an execution where the problem data matrices are stored on disk, and have to be transferred between secondary storage and the GPU memory.

Our results in Figure 2 report the asymptotic performance of the OOC-GPU algorithms, determined as the highest GFLOPS rate when the problem dimension grows to fill the capacity of the main memory. At this point, it is worth noticing that the higher GFLOPS ratio for SD&C does not imply a shorter execution time, as this method also exhibits a much higher cost than the two-stage alternative (see Section 2 and [2]). In any case, the experiment serves its purpose, showing that the OOC-GPU SD&C and two-stage implementations deliver, respectively, sustained rates of 201.89 and 49.52 GFLOPS for the largest problem size. More importantly, the trends revealed by this experiment indicate we can hardly expect a raise in the GFLOPS rate by working with larger problems (even if they fitted into the main memory) as the performance rates are quite flat for the largest two problem sizes.

4.2. Comparison of the eigensolvers

We next compare the theoretical execution times of the OOC-GPU eigensolvers, when operating with much larger cases, against the actual execution time of the OOC Krylov subspace-based solver with data stored on disk. In particular, we estimate the execution time of the OOC-GPU compute-bounded eigensolvers from the relation between their theoretical cost (in flops, see Section 2) and the sustained GFLOPS rates observed earlier as time in seconds = cost in flops/(sustained GFLOPS rate $\cdot 10^9$). On the other hand, for the Krylov subspace method, we report the experimental execution

DOI: 10.1002/cpe

Acronym	Name	PDB id.	n
ER	Eucariotic Ribosome*	2xsy,2xtg	36,488
CCMV	Cowpea chlorotic mottle virus	1cwp	56,454
VP	Vault protein	2zuo,2zv4,2zv5	119,486
VP_A			75,518
VP_B			119,486
HK97	Hong Kong 97 virus Head II	2ft1	227,154
$HK97_A$			100,325
$HK97_B$			149,231
MT	Microtubule 13:3	1tub	908,954

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.

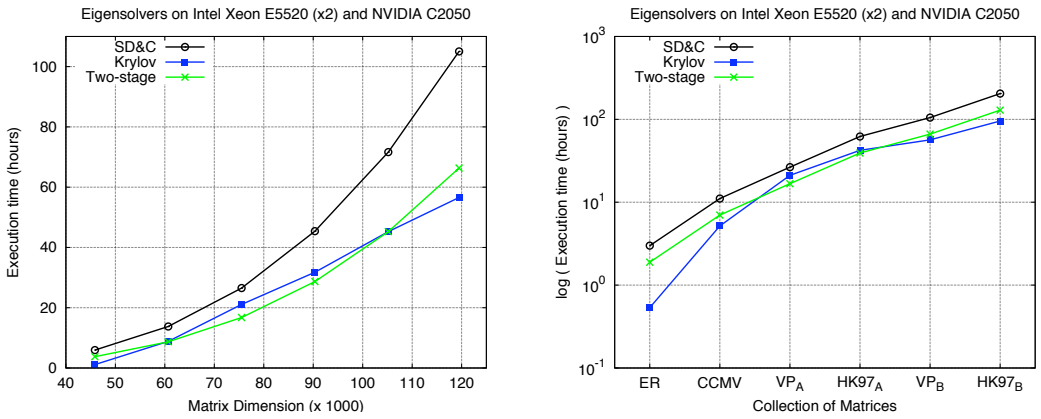

Figure 3. Estimated and experimental execution times of the eigensolvers applied to scaled instances of macromolecule VP (left plot) and the benchmark collection of cases (right plot).

time. Based on our experience with the SD&C eigensolver and the testbed considered in [2], we assume that this method converges in 7 iterations and, therefore, its cost is $42n^3$ flops. In all three cases, the eigensolvers include the cost of the initial transformation into a standard eigenproblem (Cholesky factorization and two triangular system solves), implemented as OOC algorithms that operate with data on disk and exploit the multicore processors available in the platform.

Figure 3 reports the performance, in terms of execution time, of the three eigensolvers for scaled instances of macromolecule VP (left plot) as well as all the cases included in the benchmark collection (right plot, with logarithmic scale for the y axis): ER, CCMV, VP_A, $HK97_A$, VP_B, and $HK97_B$; see Table I for details. Both experiments reveal the high performance of the two-stage solver for several of the cases which, on the other hand, is overcome by the Krylov method for the largest problems. Interestingly, even though the latter solver operates with data on disk, which limits its performance to that dictated by the data transfer rate between the disk and the processor floating-point units, the much lower theoretical cost of the Krylov method ($O(n^2k)$ flops, where k denotes the number of iterations for convergence; see the **while** loop in lines 26–38 of Listing 1) compared with that of the two-stage approach ($O(n^3)$ flops) determines this result. At this point, we note that nothing prevents us from being able to tackle much larger macromolecules using our Krylov eigensolver, in principle, as large as determined by the size of the platform disk. However, given the constant GFLOPS rate and disk bandwidth for the two-stage and Krylov eigensolvers, respectively,

Prepared using cpeauth.cls
DOI: 10.1002/cpe

we can only expect an increase in the performance gap in favor of the Krylov subspace-based method for those much larger cases.

As noted earlier, the performance of the Krylov subspace-based eigensolver depends on the number of iterations k of the **while** loop (lines 26–38) in Listing 1 which, in turn, depends on the clustering of the eigenproblem s smallest eigenvalues (i.e. those that are to be computed).

Figure 4 compares the cost and convergence of the Krylov method with the two-stage eigensolver. In the left plot we report the number of iterations that result in the Krylov method being as expensive as the two-stage solver for scaled instances of macromolecule VP. This experiment reveals a steady and rapid growth in the number of iterations as the problem size is increased (except for the smallest problem cases). In the right plot of Figure 4 we employ the largest case of this macromolecule and report the execution time of the Krylov method as the number of iterations grows, showing that the cross-over point between this method and the two-stage solver occurs around iteration 380.

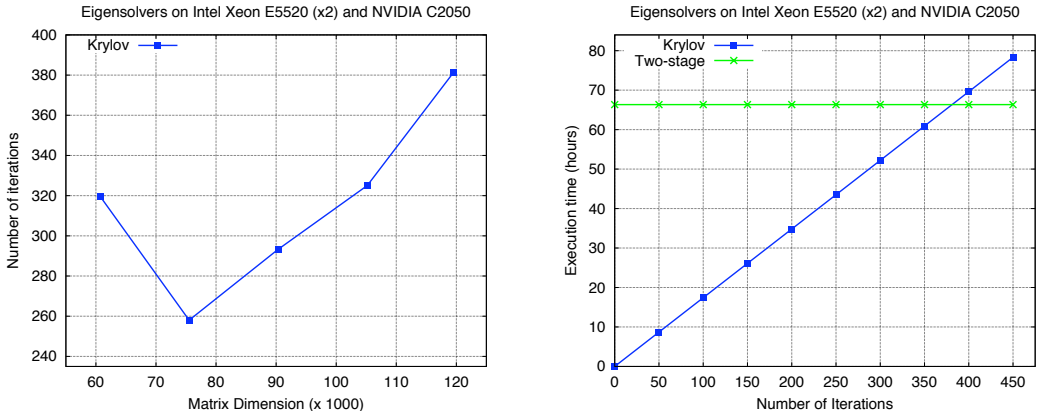

Figure 4. Relation between convergence rate and cost of the Krylov subspace-based eigensolver and execution time of the two-stage reduction-based eigensolver.

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 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.

The Krylov subspace-based eigensolver presents a much lower theoretical cost than the OOC-GPU alternatives, but this method casts most of its computations in terms of a memory-bounded operation like the (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-state reduction attain high performance by carefully amortizing the cost of the PCI 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

Prepared using cpeauth.cls
DOI: 10.1002/cpe

memory. We believe that this result carries over to an scenario where the problem matrices are stored on disk.

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 also show the superior performance of the OOC-GPU two-stage and Krylov approaches over the divide-and-conquer implementation for all problem sizes, the high correlation between the execution time of the Krylov subspace-based eigensolver and its convergence rate, and the asymptotic superiority of the Krylov approach as the problem size increases.

As part of future work, we plan to analyze more advanced Krylov eigensolvers, with faster convergence, than can turn this method even more competitive with the two-stage eigensolver.

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, and the EU Project FP7 318793 "EXA2GREEN".

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. *Applied Mathematics and Computation*, 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 *Proceedings of the 10th International Conference on 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. G. S. Ayton and G. A. Voth. Systematic multiscale simulation of membrane protein systems. *Curr. Opin. Struct. Biology*, 19(2):138–44, 2009.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.
11. 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.
12. R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users guide: Solution of large scale eigenvalue problems by implicitly restarted arnoldi methods., 1997.
13. 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.
14. J. R. Lopez-Blanco, J. I. Garzon, and P. Chacon. iMOD: multipurpose normal mode analysis in internal coordinates. *Bioinformatics*, 27(20):2843–50, 2011.
15. 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.

DOI: 10.1002/cpe

16. MAGMA project home page. http://icl.cs.utk.edu/magma/.
17. 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.
18. T. Noguti and N. Go. Dynamics of native globular proteins in terms of dihedral angles. *J. Phys. Soc. Jpn.*, 52(9):3283–3288, 1983.
19. Gregorio Quintana-Ortí, Francisco D. Igual, Mercedes Marqués, Enrique 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.
20. L. Skjaerven, S. M. Hollup, and N. Reuter. Normal mode analysis for proteins. *J. Mol. Struct. (Theochem)*, 898(1-3):42–48, 2009.
21. Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996.
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.

DOI: 10.1002/cpe