Structure-Aware Calculation of Many-Electron Wave Function Overlaps on Multicore Processors

Davor Davidović[1], https://orcid.org/0000-0003-2649-9236 and Enrique S. Quintana-Ortí[2], https://orcid.org/0000-0002-5454-165X

[1] Centre for Informatics and Computing, Ruđer Bošković Institute, Bijenička cesta 54, 10000 Zagreb, Croatia. davor.davidovic@irb.hr

[2] Depto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46.022–València, Spain. quintana@disca.upv.es

Abstract. We introduce a new algorithm that exploits the relationship between the determinants of a sequence of matrices that appear in the calculation of many-electron wave function overlaps, yielding a considerable reduction of the theoretical cost. The resulting enhanced algorithm is embarrassingly parallel and our comparison against the (embarrassingly parallel version of) original algorithm, on a computer node with 40 physical cores, shows acceleration factors which are close to 7 for the largest problems, consistent with the theoretical difference.

Keywords: Wawe functions · LU factorization · multicore processors

1 Introduction

Many-electron wave function (MEWF) overlaps are extensively used in the nonadiabatic dynamics and have significant importance in photochemical studies. Concretely, the overlap functions provide a straightforward mechanism to record the electronic states along different nuclear geometries. Therefore, they can be leveraged for constructing multi-state, multi-dimensional potential energy surfaces for quantum dynamics [6]. In the context of nonadiabatic dynamics simulations, for example, MEWF overlaps yield an approximation of time-derivative couplings (TDCs) in fewest-switch surface hopping (FSSH) calculations [11, 5]. The main drawback of the approaches based on MEWF overlaps lies in their high computational complexity and poor scaling with the system size. Thus, accelerating the calculation of the overlap functions can significantly augment the dimension of the systems to which the FSSH method can be applied.

In this work, we improve the algorithm presented in [10] for computing the overlaps between excited states using CIS-type wave functions. In particular, we optimize the algorithm denoted there as OL2M –an approach based on the level-2 minors obtained from Laplace's recursive formula, or in other words, the minors obtained by removing two rows and two columns from the input referent matrix. In rough detail, our optimization targets the part of the OL2M algorithm in which the determinants of all the level-2 minors are computed and stored, introducing a structure-aware variant of the method that reduces the

theoretical cost of that part of the algorithm by an order of magnitude via an *update of the LU factorization*; see, e.g., [4, 8]. This enhancement results in significantly shorter execution times for large problems solved in parallel on a multicore processor.

The rest of the paper is structured as follows. In Section 2 we offer a brief introduction to the computations of MEWF overlaps. The columnwise structure-aware algorithm and its parallelization are described in Section 3; and the parallel experimental results are presented in 4. Finally, a few concluding remarks and future research directions close the paper in Section 5.

2 Problem definition

Given two many-electron wave-functions, denoted by Ψ_I and Ψ_J, the overlap between these functions is expressed as:

$$S_{IJ} = \langle \Psi_I | \Psi_J \rangle \,, \tag{1}$$

using bra-ket $\langle * | * \rangle$ notation [2], a common notation used in quantum mechanics to describe quantum spaces. The S_{IJ} is the (I, J) element of the overlap matrix S, N_A is the number of states, and the indices $I, J \in \{1, 2, \ldots, N_A\}$. The N_A states are described by CIS-type wave functions and, therefore, they can be expanded using Slater determinants. In the Slater determinants expansion, the electrons are divided into n_σ and m_σ respectively, representing the number of occupied and virtual orbitals for each spin σ.

The mathematical-physics problem can be reformulated into a matrix representation as that in Figure 1. The main computational problem consists in obtaining the determinants of all the matrices constructed such that a row i and column j from the referent matrix A_{ref} are replaced with the contents of row i_β and column j_α, respectively, from matrices WF_β and WF_α, for all possible combinations of i, j, i_β, j_α.

The total number of possible matrices is $n_\sigma^2 m_\sigma^2$ for each spin $\sigma \in \{\alpha, \beta\}$. Furthermore, if an LU factorization [4] is applied to compute these determinants, which requires $2/3n_\sigma^3$ flops (floating-point operations), this operation becomes the main bottleneck of the entire MEWF, yielding a total cost of $O(n_\sigma^5 m_\sigma^2)$ flops.

It is worth to note that, for a fixed row and column (i, j) of A_{ref}, numerous rows and columns from WF_β and WF_α are to be tested, while all other rows and columns in the referent matrix remain unchanged. Thus, the matrices, for which we have to compute the determinants differ only in one row and one column. In order to decrease the computational cost, the challenge is to reuse the unchanged parts of referent matrix for computing other determinants.

The first approach to exploit this property was presented in [7]. There, the authors reported a significant speedup by exploiting similarities between the consecutive matrices differ in only one row/column. Concretely, the referent matrix A_{ref} was expanded by columns using the Laplace transformation. The determinants of the minors were then stored and reused to compute determinants of matrices which differ only in one column from the baseline determinant. With

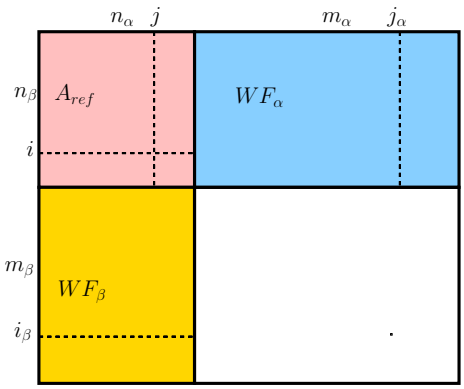


Fig. 1: Matrix representation of the computational problem. The referent matrix is square with dimension $n_\beta \times n_\alpha$. Rows/columns i/j are replaced with rows/columns from the panels WF_β and WF_α, respectively.

this strategy, replacing any column for WF_α is simply a linear combination of factors from j_α with pre-computed determinants of the minors.

The work presented in [10] improved the idea in [7] by computing the determinants of the minors obtained from the second-level recursive Laplace expansion. There, the minors were constructed by expanding the referent matrix along both rows and columns; then the LU factorization was repeatedly applied to obtain the determinants; and finally these results were reused to compute the overlap of the corresponding Slater determinants. Hereafter, we will refer to this algorithmic approach as DL2M. Although this strategy yields a considerable reduction in the computational complexity compared with the method in [7], obtaining the determinants of the minors remains the major computational bottleneck.

A related topic is that of computing the LU factorization of a large collection of small matrices. This problem has been tackled, on graphics processors, as part of an effort to develop a batched version of the Basic Linear Algebra Subprograms as well as in the framework of computing a block-Jacobi preconditioner for the iterative solution of sparse linear systems [3, 1]. However, in those applications the matrices are independent and no structure-aware explotation of the problem is possible.

3 Algorithms for DL2M Calculations

3.1 Original

Consider the matrix $A \in \mathbb{R}^{n \times n}$ representing the referent matrix A_{ref} and assume, for simplicity, that $n = n_\alpha = n_\beta = m_\alpha = m_\beta$. The calculation of the overlap between any two MEWFs, described in Section 2, requires the computation of the determinants for all possible submatrices of A where any two rows/-columns have been eliminated from the matrix. Let us denote by $A_{r_1,r_2||c_1,c_2}$ the

submatrix that results from eliminating rows r_1, r_2 and columns c_1, c_2 from A. The straightforward solution to obtain the determinants is to explicitly construct all possible submatrices with $m = n - 2$ rows/columns of A, and then compute the LU factorization (with partial pivoting) of each submatrix, as shown in the naive algorithm (NA) in Listing 1.1. For brevity, hereafter we employ pseudo-Matlab notation in the presentation of the algorithms, and we do not consider the effect of the row permutations obtained from the LU factorization on the determinant sign. All our actual realizations of the algorithms include partial pivoting to ensure numerical stability in practice.

The computational cost of the NA realization is, approximately,

$$\sum_{r_1=1}^{n} \sum_{r_2=1}^{r_1-1} \sum_{c_1=1}^{n} \sum_{c_2=1}^{c_1-1} 2n^3/3 \approx n^7/6 \text{ flops.}$$

```
1 for r1=1:n
2   for r2=1:r1-1
3     for c1=1:n
4       for c2=1:c1-1
5         [L,U,P] = lu(A_{r1,r2||c1,c2});
6         d[r1][r2][c1][c2] = prod(diag(U));
```

Listing 1.1: Naive algorithm for DL2M calculations.

3.2 Columwise re-utilization

The naive algorithm in Listing 1.1 exposes that, between any two iterations of the two inner loops (that is, those indexed by `c1`, `c2`), the matrix that needs to be factorized only differs in two columns. A natural question is thus how to exploit the fact that all other matrix columns remain the same between these two iterations. To illustrate the response, let us define $A_r = A_{r_1,r_2||\_} \in \mathbb{R}^{m \times n}$ as the submatrix with $m = n-2$ rows and n columns that results from eliminating only rows r_1, r_2 from A. Next, consider the LU factorization of this submatrix:

$$L^{-1}PA_r \;=\; L^{-1}P\,[a_1, a_2, \ldots, a_n] \;=\; [u_1, u_2, \ldots, u_n] \;=\; U, \qquad (2)$$

where $L \in \mathbb{R}^{m \times m}$ is a unit lower triangular comprising the Gauss transforms that are applied to annihilate the subdiagonal entries of the matrix, P is the $m \times m$ permutation matrix due to the application of partial pivoting during the factorization, and $U \in \mathbb{R}^{m \times n}$ is the resulting upper triangular factor [4]. The answer we are searching for should state the relationship between the factorization of A_r in (2) and that of the submatrix

$$A_{r_1,r_2||c_1,c_2} = [a_1, a_2, \ldots, a_{c_2-1}, a_{c_2+1}, \ldots, a_{c_1-1}, a_{c_1+1}, \ldots, a_n], \qquad (3)$$

for any two column values c_1, c_2. Since a Gauss transform, applied to a matrix from the left, simply performs an independent linear combination of each matrix

column, the application of the factors L and P from the LU factorization in (2) to $A_{r_1,r_2||c_1,c_2}$ results in:

$$L^{-1}PA_{r_1,r_2||c_1,c_2} = [u_1, u_2, \ldots, u_{c_2-1}, u_{c_2+1}, \ldots, u_{c_1-1}, u_{c_1+1}, \ldots, u_n]\,; \qquad (4)$$

which corresponds to the columns of U in (2), except for u_{c_1} and u_{c_2}, which have disappeared. The result is thus already upper triangular in the leftmost $c_2 - 1$ columns, but it contains zeros only below the first and second subdiagonals in columns $[u_{c_2+1}, u_{c_2+2}, \ldots, u_{c_1-1}]$ and $[u_{c_1+1}, \ldots, u_n]$, respectively; see the example in Figure 2.

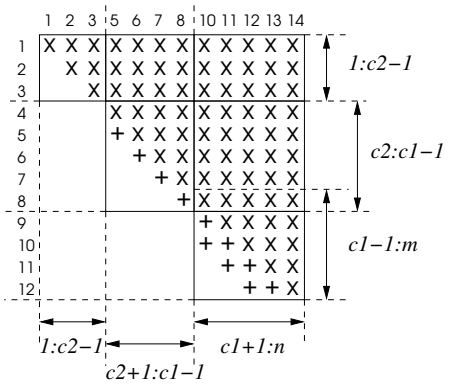


Fig. 2: Structure of the upper triangular matrix in (4), with $n = 14$, $m = n-2 = 12$, $c_2 = 4$ and $c_1 = 9$. Nonzero entries below the main diagonal are identified with the symbol '+'. Columns are numbered taking into account that c_1, c_2 were eliminated.

In consequence, in order to obtain the desired upper triangular matrix, which yields the determinant for $A_{r_1,r_2||c_1,c_2}$ we need to apply Gauss transforms (or, alternatively, any type of orthogonal transform [4]) to eliminate the nonzero entries below the main diagonal of this matrix. Let us consider the partitioning of the quasi-upper triangular factor in (4) as follows:

$$L^{-1}PA_{r_1,r_2||c_1,c_2} = \begin{array}{|c|c|c|l} \hline U_{11} & U_{12} & U_{13} & 1 : c_2 - 1 \\ \hline & U_{22} & U_{23} & c_2 : c_1 - 1 \\ \hline & & U_{33} & c_1 : m \\ \hline \end{array} \qquad (5)$$

$$\underbrace{1 : c_2 - 1}\ \underbrace{c_2 + 1 : c_1 - 1}\ \underbrace{c_1 + 1 : n}$$

where U_{11} is upper triangular; and U_{22}/U_{33} contain zeros below the first/second subdiagonal. Furthermore, consider $U_{23} = \left[\frac{U_T}{u_B}\right]$, where u_B corresponds to the bottom row of U_{23}. The algorithm that exploits the relationship between the matrices factorized in the inner two loops is shown in Listing 1.2. Naturally, the LU factorizations involving the blocks $[U_{22}, U_{23}]$ and U_{33} leverage the special quasi-upper triangular structure of these submatrices to reduce the cost of this alternative method.

```
1  for r1=1:n
2    for r2=1:r1-1
3      [L,U,P] = lu(A_{r1,r2||-});
4      for c1=1:n
5        for c2=1:c1-1
6          Partition U as in equation (5) → U11, U22, U23, U33
7          [L2,U2,P2] = lu([U22, U23]);% Exploit zeros below first subdiagonal
8          [L3,U3,P3] = lu([uB;
9                           U33]);     % Exploit zeros below second subdiagonal
10         d[r1][r2][c1][c2] = prod(diag(U11))
11                           * prod(diag(U2))
12                           * prod(diag(U3));
```

Listing 1.2: Algorithm for DL2M calculations with columnwise structure-aware reutilization.

The cost of the columnwise "structure-aware" realization is approximately given by

$$\sum_{r_1=1}^{n}\sum_{r_2=1}^{r_1-1}\left(\underbrace{2m^3/3}_{\text{LU of } A_r} + \sum_{c_1=1}^{n}\sum_{c_2=1}^{c_1-1}\left(\underbrace{\sum_{j_1=c_2+1}^{c_1-1} 6(n-j_1)}_{\text{LU of } [U_{22},U_{23}]} + \underbrace{\sum_{j_2=c_1+1}^{n-1} 12(n-j_2)}_{\text{LU of } [u_B;U_{33}]}\right)\right) \approx n^6/2 \text{ flops},$$

where the sum for j_1 corresponds to the cost of the LU factorization for $[U_{22},\ U_{23}]$ and the sum for j_2 to that for U_{33}. Taking into account that $m \approx n$, and neglecting the lower order terms, the cost for this approach is $n^6/2$ flops. Compared with NA, this represents a reduction in the cost of one order of magnitude.

3.3 Parallel implementation

Both the original NA and the columnwise structure-aware algorithm (CSA), described in the previous subsections, are embarrassingly parallel. An analysis of dependencies and concurrency is, therefore, trivial. From Listing 1.1, we observe that the LU factorizations of the minors (and the accumulation of the diagonal elements of the resulting triangular factors), in the inner-most loop, are completely independent. In other words, each minor can be constructed independently of other minors, and the corresponding calculations can proceed in

parallel. Although this approach exhibits a much larger memory footprint, because the minors are explicitly constructed, it accommodates a straight-forward parallelization. In contrast, in real-world use cases, the dimension of the initial matrix is up to $n = 200$, and exploiting only the parallelism intrinsic to the operations involved in a single LU factorization will surely exhibit very low performance. Therefore, our approach computes single-threaded (i.e. sequential) LU factorizations, but combines this with a parallelization of the outer loops around these calculations to compute several decompositions concurrently.

For the parallelization on multicore processors, in this work we leverage OpenMP. Concretely, in the DL2M NA case, in Listing 1.1, an OpenMP parallel for pragma is applied to the outer-most loop of the algorithm –that is, the loop over index r_1– which corresponds to the first row to be removed from the initial matrix. The work corresponding to the three inner loops (over r_2, c_1, and c_2) is then executed in parallel for each iteration of the row index r_1.

For the CSA variant, in Listing 1.2, the inner-most LU factorizations and the products of diagonal elements of the factors $U_{\{11,2,3\}}$ are independent for each combination of the columns c_1 and c_2. They require only the U factor obtained from the two outer-most loops (a unique combination of r_1 and r_2), as in Line 3 of Listing 1.2. In consequence, the parallelization of DL2M with columnwise reuse is done by distributing the iteration space across loop c_1; that is over the first column to be removed from the minor (without rows r_1 and r_2). For this variant, it is also possible to parallelize across the outer-most loop r_1 like it was done for L2M NA. That approach can be followed, for example, to parallelize the outer-most loop across multiple nodes, while the parallelization over c_1 can be applied at the node level. This multi-level parallel alternative for clusters is part of ongoing work.

4 Experimental Results

In this section we illustrate the gains in performance attained by the DL2M structure-aware algorithm, with columnwise re-utilization, for computing the determinants of the level-2 minors of the referent matrix. For this purpose, the new algorithm, CSA, is compared against the original NA realization, described in subsection 3.1. In addition, in the final part of this section, we compare the overall performance of the MEWF algorithms for computing the overlap of the wave functions, using our DL2M algorithm with columnwise re-utilization, with the algorithm OL2M, described in [10].

Set up. The experiments in this section were obtained on the Juwels cluster from Juelich Supercomputing Center. Each node consists of two Intel Xeon Platinum 8168 processors, running at 2.7 GHz, with 24 cores each and 96 GB of RAM. The code was written in Fortran and C programming languages, compiled with GCC 8.3.0 and linked against the Intel MKL 2019.3.199 (sequential) library. All experiments are run on a single node and employ double precision arithmetic.

Algorithmic improvements. The first experiment assesses the speed up gains achieved only by introducing algorithmic changes, without any parallelisation strategies. The columnwise reutilization strategy yields a significant performance gain up to 2.5× and 5× compared with the OL2M and NA variants, respectively, even when executed sequentally, as illustrated in Figure 3. Note that the difference between CSA and other variants increases with the problem size. That is because of much lower computational cost of CSA compared to NA and OL2M variants. As an example, for $n = 100$ the flops count for NA is $10^14/6$ while CSA exhibits $10^12/2$ flops, that is approximately 33× less flops for CSA variant. However, in NA and OL2M variants more flops are "fast" flops based on LU and BLAS-3 operations, while in CSA, a part of flops, due to columnwise reutilization strategy, are based on BLAS-1 operations (the update step).

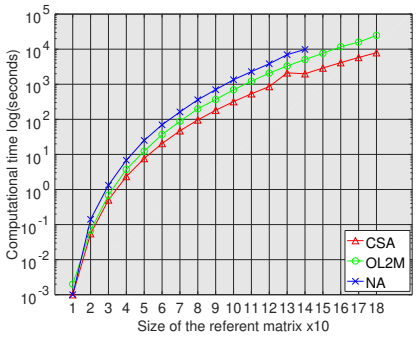
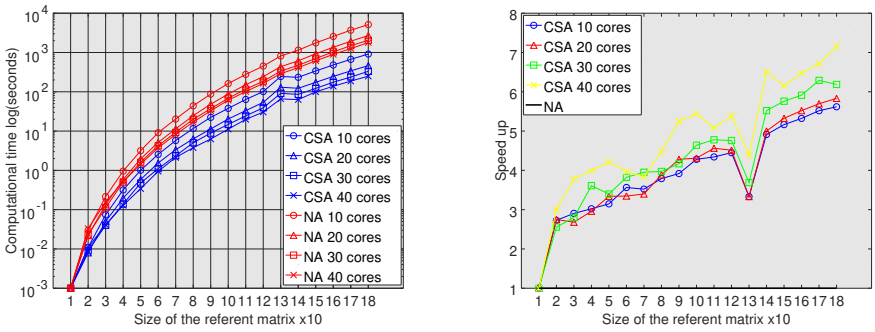


Fig. 3: Execution time of the sequential NA, OL2M and CSA algorithms.

NA vs CSA. This experiment compares the DL2M CSA variant (Listing 1.2) and the DL2M NA implementation (Listing 1.1). The input matrix for this test was generated with random entries and a number of columns/rows ranging from $n = 10$ to 180. The lower theoretical cost of the new CSA approach (of an order of magnitude) becomes evident in Figure 4 (left), which shows a reduction of the execution time by approximately one order of magnitude for sufficiently large matrices. When increasing the matrix size though, the speedup compared to NA significantly increases, because of the much lower computational cost of the CSA variant. Figure 4 (right) shows that the speed up of CSA (with respect to NA) is consistent, independently of the number of cores in the test, and roughly grows by a factor of up to 7 for the largest test matrices.

We can observe that the CSA algorithm is superior in performance to the naive version for all configurations and matrix sizes, as presented in Figure 4 (right). It can been seen that the speedup is similar, no matter how many cores we used in test configuration, and is increasing up to more than 7× for larger test matrix sizes.

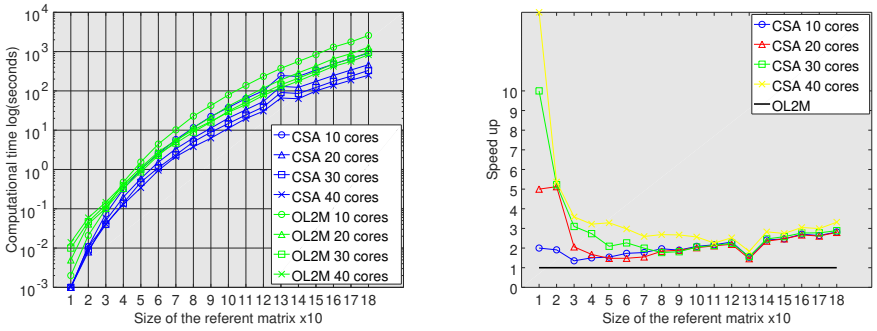


(a) Execution time (b) Speedup

Fig. 4: Execution time and speed-up (left and right plots, respectively) for the parallel DL2M CSA and NA algorithms for varying #cores and matrix size.

OL2M vs CSA. In Figure 5 (left) we compare the new DL2M CSA with the OL2M algorithm [10] (considering only the part that computes the determinants of the minors). In OL2M, the complete LU is computed inside the c_1 loop, and the accumulated U factors are reused only inside the c_2 loop. By computing the LU before the start of the c_1 loop, the CSA variant offers a speed up of up to 3.3× for the largest problems, Figure 5 (right).

Note that the speedup of CSA w.r.t. OL2M is also independent on the number of cores (as in the case of NA) and that is almost the same for different test configurations and larger test matrix sizes.



(a) Execution time (b) Speedup

Fig. 5: Execution time and speed-up (left and right plots, respectively) for the parallel DL2M CSA and OL2M algorithms for varying #cores and matrix size.

Scalability Although CSA re-uses the columns from the U factor for computing determinants of the subsequent minors, the algorithm achieves a good scalability with increasing number of cores, as presented in Figure 6.

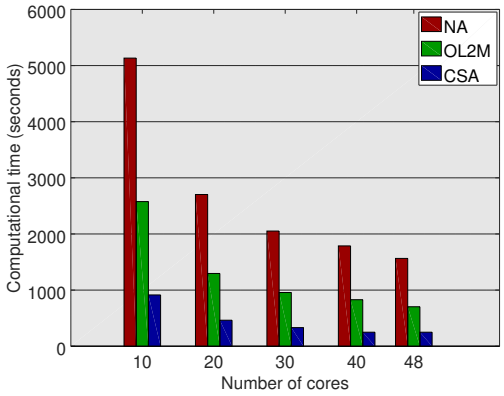


Fig. 6: Strong scaling of CSA variant with the number of processor for $n = 180$ (number of rows/columns of the referent matrix).

Real use-case The final experiment compares DL2M CSA and the OL2M algorithm on real test cases corresponding to the excited state wave functions of poly-Alanine systems (with 100 and 195 occupied orbitals, i.e. the sizes of the referent matrices) obtained using different basis sets. The results of this test in Table 1 show that DL2M CSA offers higher scalability and achieves a speed up factor above 5× over DL2M, on 48 cores, which is aligned with the results reported in Figure 5 (right). For the Alanine-100 use-case, by increasing the number of cores over 40 a drop in performance is occured. That is expected since the problem becomes to small for the given number of cores in which the communication overhead, due the increased paralelism, becomes more significant in the total execution time.

5 Conclusion and Future Work

In this work, we leverage the connection between the level-2 minors of the referent matrix appearing in MEWFs to save a considerable part of the computations required to obtain the corresponding determinants. For that purpose, we use an updating technique for the LU factorization, in order to reduce the cost by an order of magnitude. Our tests show that the new approach considerably accelerates the computation of the MEWF overlaps, by a factor of up to 7, in principle allowing the solution of larger problems.

As part of future work, we plan to explore the parallelization of this algorithm using different approaches and/or tools (e.g., to exploit task-level parallelism, or

Table 1: Execution times (in seconds) of DL2M CSA and OL2M for Alanine-100 and Alanine-195 on varying #cores.

#cores	Algorithm	Alanine-100		Alanine-195	
		Minors	Total	Minors	Total
10	OL2M	78.02	92.57	4,233.16	4,432.2
	CSA	37.61	51.96	1,364.06	1,568.2
20	OL2M	39.47	53.00	2,117.29	2,303.6
	CSA	20.26	33.89	691.78	896.39
30	OL2M	29.29	43.74	1,557.32	1,755.0
	CSA	14.13	28.79	478.16	674.98
40	OL2M	28.33	44.11	1,306.56	1,508.5
	CSA	11.61	26.83	410.27	628.87
48	OL2M	29.41	44.92	1,142.5	1,365.7
	CSA	10.72	26.46	364.56	581.14

to combine a cluster-level parallelization with a finer grain concurrent execution). In addition, we will investigate how to exploit the connection between the level-2 minors across other dimensions to further reduce the theoretical cost of this type of computations.

The source code is available at [9] and is part of the *cto-nto* library for computing natural transition orbitals for CIS type wave functions.

Acknowledgement

This research was performed under project HPC-EUROPA3 (INFRAIA-2016-1-730897) and supported by Croatian Science Fundation under grant HRZZ IP-2016-06-1142, the Foundation of the Croatian Academy of Science and Arts, and the European Regional Development Fund under grant KK.01.1.1.01.0009 (DATACROSS). Enrique S. Quintana-Ortí was supported by project TIN2017-82972-R of the MINECO and FEDER. The authors gratefully acknowledge the computer resources provided by the Juelich Supercomputing center, and to BSC where the initial testings and the code development were performed.

References

1. Anzt, H., Dongarra, J., Flegar, G., Quintana-Ortí, E.S.: Variable-size batched LU for small matrices and its integration into block-Jacobi preconditioning. In: 2017 46th International Conference on Parallel Processing (ICPP). pp. 91–100 (2017). https://doi.org/10.1109/ICPP.2017.18, doi.ieeecomputersociety.org/10.1109/ICPP.2017.18
2. Dirac, P.A.M.: A new notation for quantum mechanics. Mathematical Proceedings of the Cambridge Philosophical Society **35**(3), 416–418 (1939). https://doi.org/10.1017/S0305004100021162
3. Dong, T., Haidar, A., Luszczek, P., Harris, J.A., Tomov, S., Dongarra, J.: Lu factorization of small matrices: Accelerating batched dgetrf on the gpu. In: 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE

6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS). pp. 157–160 (Aug 2014). https://doi.org/10.1109/HPCC.2014.30
4. Golub, G.H., Loan, C.F.V.: Matrix Computations. The Johns Hopkins University Press, Baltimore, 3rd edn. (1996)
5. Hammes-Schiffer, S., Tully, J.C.: Proton transfer in solution: Molecular dynamics with quantum transitions. J. of Chemical Physics **101**(6), 4657–4667 (sep 1994). https://doi.org/10.1063/1.467455
6. Li, S.L., Truhlar, D.G., Schmidt, M.W., Gordon, M.S.: Model space diabatization for quantum photochemistry. J. of Chemical Physics **142**(6), 064106 (feb 2015)
7. Plasser, F., Ruckenbauer, M., Mai, S., Oppel, M., Marquetand, P., González, L.: Efficient and flexible computation of many-electron wave function overlaps. Journal of chemical theory and computation **12**(3), 1207–1219 (2016)
8. Quintana-Ortí, E.S., Van De Geijn, R.A.: Updating an LU factorization with pivoting. ACM Trans. Math. Softw. **35**(2), 11:1–11:16 (2008)
9. Sapunar, M.: Natural transition orbitals for CIS type wave functions. https://github.com/marin-sapunar/cis_nto, accessed: Oct 24, 2019
10. Sapunar, M., Piteša, T., Davidović, D., Došlić, N.: Highly efficient algorithms for CIS type excited state wave function overlaps. Journal of chemical theory and computation (2019)
11. Tully, J.C.: Molecular dynamics with electronic transitions. J. of Chemical Physics **93**(2), 1061–1071 (jul 1990). https://doi.org/10.1063/1.459170