PARALLELIZED IMPLEMENTATION OF THE CCSD(T) METHOD IN MOLCAS USING OPTIMIZED VIRTUAL ORBITALS SPACE AND CHOLESKY DECOMPOSED TWO-ELECTRON INTEGRALS

Michal PITOŇÁK[a1,b,*], Francesco AQUILANTE[c], Pavel HOBZA[b1,d], Pavel NEOGRÁDY[a2], Jozef NOGA[e,f] and Miroslav URBAN[a3,g]

[a] *Department of Physical and Theoretical Chemistry, Faculty of Natural Sciences, Comenius University, Mlynská Dolina, SK-842 15 Bratislava, Slovak Republic; e-mail: [1] pitonak@fns.uniba.sk, [2] neogrady@fns.uniba.sk, [3] urban@fns.uniba.sk*

[b] *Institute of Organic Chemistry and Biochemistry, Academy of Sciences of the Czech Republic, v.v.i. and Center for Biomolecules and Complex Molecular Systems, Flemingovo nám. 2, 166 10 Prague 6, Czech Republic; e-mail: [1] pavel.hobza@uochb.cas.cz*

[c] *Department of Physical and Analytical Chemistry, Quantum Chemistry, Uppsala University, P.O. Box 518, SE-75120 Uppsala, Sweden; e-mail: francesco.aquilante@gmail.com*

[d] *Department of Physical Chemistry, Palacký University, 771 46 Olomouc, Czech Republic*

[e] *Department of Inorganic Chemistry, Faculty of Natural Sciences, Comenius University, SK-842 15 Bratislava, Slovak Republic; e-mail: jozef.noga@fns.uniba.sk*

[f] *Institute of Inorganic Chemistry, Slovak Academy of Sciences, SK-845 36 Bratislava, Slovak Republic*

[g] *Slovak University of Technology in Bratislava, Faculty of Materials, Science and Technology in Trnava, Institute of Materials, J. Bottu 24, SK-917 24 Trnava, Slovak Republic*

Received February 25, 2011
Accepted April 7, 2011
Published online May 4, 2011

This paper is dedicated to Dr. Zdeněk Havlas on the occasion of his 60th birthday. The novel, robust implementation of the CCSD(T) method, described in this paper, was motivated by the ambition to extend the applicability of the highly correlated calculations in the area of noncovalent interactions – one of the major fields of Dr. Havlas's scientific interest. It is indisputably his personal credit that these methods were successfully applied to challenging biological problems on the ground of the Institute of Organic Chemistry and Biochemistry of the Academy of Science of the Czech Republic. We appreciate his pioneering contribution in this field as well as his generous support for our projects during the past years.

Parallelized implementation of the coupled cluster singles doubles with non-iterative triples in the MOLCAS program suite is described. The code benefits from the Cholesky decomposition of two-electron integrals and the algorithm is particularly designed for calculations using reduced optimized virtual orbital space. Different aspects of parallelization and its efficiency are discussed based on our recent successful calculations for medium sized molecules involving more than 1000 basis functions.

Keywords: *Ab initio* calculations; Electronic structure; Quantum chemistry; Coupled cluster; Cholesky decomposition; Parallelization; Noncovalent interaction; MOLCAS.

doi:10.1135/cccc2011048

Without doubt, nowadays widely used coupled cluster (CC) approach including single, and double excitations with perturbative correction for triple excitations (CCSD(T))[1] proved to be one of the most reliable methods for a treatment of the electronic structure of molecules. This holds, despite its routine applicability is still limited. From the conceptual point of view it is a limitation to molecular ground states (or more generally, the lowest states within each symmetry representation) and to systems for which these states are strongly dominated by a single (reference) determinant[2]. From the computational point of view, the most severe limitation results from the steep scaling of CCSD(T) with respect to the number of basis functions (N) and number of correlated electrons involved in the calculation. Let O and V denote the number of correlated occupied orbitals (OOs) and the number of active virtual orbitals (VOs), respectively. It is notoriously known that CCSD involves steps that formally scale as $\propto O^3V^3$, $\propto O^2V^4$ and $\propto O^4V^2$ in the iterative manner and the correction for triples includes even more steep $\propto O^3V^4$ and $\propto O^4V^3$ dependences.

As discussed in several studies (see e.g. ref.[3]), this $\propto N^6$ (or $\propto N^7$) scaling does not reflect the physical nature of the interaction between electrons (and nuclei) and a natural way to tackle larger systems with the CC methods would be exploitation of the short-range/local nature of the electron-electron correlation. There has been a lot of work done in this field (see e.g. refs[4–9]) resulting in several quite successful implementations. However, these methods are still far from being "black-box", often primarily limited by not so straightforward control of accuracy. Their most effective use applies to systems with naturally localized bonds.

A different aspect related to the dynamical electron correlation is a very slow convergence of the results with the increasing one-particle (atomic) orbital basis sets (AO). This aspect can be quite effectively treated within the explicitly correlated CC R12 ansatz of the wave function[10,11]. Latest development of this theory using alternative Slater type geminal correlation factors has proven the potential of this theory, when using basis sets of triple-zeta quality provided results equivalent to a conventional treatment using pentuple-zeta basis sets[12–15]. At the same time, the overall computational costs in CCSD(T)-R12 can be kept comparable with the conventional CCSD(T) using the same AO basis.

Feasible future applications of the CC theory on large molecular systems will likely combine both of the aforementioned treatments. Immense development in computer technologies, especially parallel systems, opens a possibility to apply the CCSD(T) approach to medium sized systems without approximations. Until recently, however, the computer implementations of

this theory were far from being optimized for these new technologies. Though "medium-sized" systems (say ~50 correlated OOs, ~500 VOs) are still within the reach of the well established standard CC (usually AO integral-direct based) CC codes of the last two decades, their robustness, (parallel) efficiency and memory demands turns out to be inadequate. Nevertheless, some recent new implementations e.g. by Janowski et al.[16] or Olson et at.[17], still based on AO integral-direct approach, Valiev et al.[18] or Lotrich et al.[19] are capable to treat systems with more than 1000 basis functions with the mentioned number electrons in a feasible manner.

Successful applications during past few years[20–23] proved that our implementation within the MOLCAS program suite belongs to this category as well. Here, we shall describe our implementation that is aimed for "medium-sized" systems being slightly different from the aforementioned ones. This article complements the recent report on the MOLCAS7 release[24].

At variance to some other modern implementations, our remains in the frame of molecular orbitals (MO) based formulation. The I/O bottleneck potentially arising from the storage of 2-electron MO integrals over four VO indexes is overcome by using the idea of Rendell et al.[25], however, employing Cholesky decomposition instead of the density fitting approximation. The main reason why we decided to "stay on the MO ground", is the straightforward possibility to combine the CC code with the truncated sets of optimized virtual orbitals (OVOS)[26] (or other type of modified VOs, like frozen natural orbitals (FNO)[27,28]. Another, although not so crucial argument for staying with the MO approach is the numerical stability related to the problem of linear dependences within the AO basis set.

Our OVOS method was described and tested on various applications[29,30] from which clearly follows that indeed immense computational savings can be achieved due to the reduction of the number of VOs. This is true especially when the main computational AO set is larger, i.e. for instance starting from triple-zeta quality in the hierarchy of correlation consistent basis sets[31]. The main idea of the OVOS approach is to unitary transform the canonical VOs in such a way that the resulting optimized orbitals form (usually much) more efficient "basis" for the expansion of the first order correlated wave-function. As it has been shown in several qualitatively different applications, the desired high accuracy can be preserved if typically 30% of the optimized VOs is removed in calculations with (aug-)cc-pVTZ basis, 40% with (aug-)cc-pVQZ basis, and even more when larger basis sets are used. If moderate accuracy is sufficient, additional 10% of the OVOS can be still removed at each cardinal number of the (aug-)cc-pVXZ hierarchy.

CCSD ALGORITHM

Basic Equations

Since the details of the CCSD theory are known from many previous works, let us restrict ourselves to a schematic overview using spin orbitals. In our implementation, we closely followed the closed shell formulation by Scuseria et al.[32] that does not need to be repeated here. Using reference determinant Φ, the amplitudes (t) of the excitation operators $\hat{T}_1$ and $\hat{T}_2$ of the CCSD wave function

$$\Psi_{\mathrm{CCSD}} = \exp(\hat{T}_1 + \hat{T}_2)\Phi \tag{1}$$

can be iteratively determined as

$$D_i^a (t^{(k+1)})_a^i = (\tilde{H}_d^{(k)})_a^i \tag{2}$$

$$D_{ij}^{ab} (t^{(k+1)})_{ab}^{ij} = (\tilde{H}_{T_1}^{(k)})_{ab}^{ij} + \underbrace{\frac{1}{2}(\tilde{H}^{(k)})_{mn}^{ij}(t^{(k)})_{ab}^{mn}}_{A_{00}} + \underbrace{(\tilde{H}^{(k)})_{km}^{ce}(t^{(k)})_{cf}^{kn}\bar{\delta}_{mn}^{ij}\bar{\delta}_{ab}^{ef}}_{A_{v0}} +$$

$$+ \underbrace{\frac{1}{2}(\tilde{H}_{T_1}^{(k)})_{ab}^{cd}(t^{(k)})_{cd}^{ij}}_{A_{vv}} - \underbrace{(\tilde{H}_d^{(k)})_k^m (t^{(k)})_{ab}^{kn}\bar{\delta}_{mn}^{ij}}_{A_0} + \underbrace{(\tilde{H}_d^{(k)})_e^c t_{cf}^{ij}\bar{\delta}_{ab}^{ef}}_{A_v} \tag{3}$$

where we have used pertinent matrix elements of the effective Hamiltonians

$$\tilde{H}_{T_1}^{(k)} = \exp(-\hat{T}_1^{(k)})\hat{H}\exp(-\hat{T}_1^{(k)}) \tag{4}$$

$$\tilde{H}^{(k)} = \exp(-\hat{T}_1^{(k)} - \hat{T}_2^{(k)})\hat{H}\exp(-\hat{T}_1^{(k)} + \hat{T}_2^{(k)}) \tag{5}$$

$$\tilde{H}_d^{(k)} = \exp(-\hat{T}_1^{(k)} - \hat{T}_2^{(k)})(\hat{H} - \hat{F}_d)\exp(-\hat{T}_1^{(k)} + \hat{T}_2^{(k)}) \tag{6}$$

with $\hat{F}_d$ being the diagonal part of the Fockian with matrix elements f_p^p.

$$D_{ij\ldots}^{ab\ldots} = f_i^i + f_j^j + \ldots - f_a^a - f_b^b - \ldots \tag{7}$$

$$\bar{\delta}^{rs}_{pq} = \delta^r_p \delta^s_q - \delta^s_p \delta^r_q \tag{8}$$

Summation over the repeated indices is assumed on the r.h.s. of the equations. Indexes *i*, *j*, ..., *n* and *a*, *b*, ..., *f* denote OOs and VOs, respectively, while *p*, *q*, ..., *s* denote molecular orbitals in general.

Irrespectively from the details, for our typical target systems the computationally most demanding part is the A_{vv} term which involves 2-electron integrals over four virtual indices. If these are stored, the I/O and storage demands raise enormously with the size of the system as $\propto V^4$ and the calculations become soon prohibitive. One of the ways to overcome this bottleneck is to calculate the A_{vv} term using the 2-electron integrals over AO's generated "on-the-fly" whenever needed[33–35]. Actual number of calculated integrals can be efficiently reduced by applying prescreening techniques[36,37].

Cholesky Decomposition

A different approach to overcome I/O bottlenecks due to two-electron integrals goes through the concept of "density fitting" (DF)[38]. The idea is that of approximating the AO products $|\mu\nu)$ with an expansion over a set of auxiliary Gaussian basis functions, and it relies on the fact that the number of auxiliary basis functions required for a reasonable accuracy is much smaller than the total number of AO products. The expansion coefficients are obtained by minimizing the "distance" (in Hilbert space) between the fitted and the actual AO products. To define this distance, the coulombic metric ($1/r_{12}$) is the one typically used, whereas complete freedom is left in the choice of the auxiliary basis set. In a formulation of DF that has become popular over the years with the name of "resolution-of-the-identity" (RI)[39], the auxiliary basis sets are designed for each AO basis through data-fitting of specific energy contributions – e.g., second-order Møller–Plesset (MP2) correlation energy – and then used to calculate the two-electron integrals over AOs (μ, ν, ...) as

$$(\mu\nu|\,\lambda\sigma) \approx \sum_P R^P_{\mu\nu} R^P_{\lambda\sigma} \tag{9}$$

$$R^P_{\mu\nu} = \sum_Q (\mu\nu|Q)(G^{-1/2})_{PQ}\,; \quad G_{PQ} = (P|Q) \tag{10}$$

where *P*, *Q* denote functions in the auxiliary basis set. On the other hand, it has been shown[38] that the generation of the auxiliary basis set in DF can be made without data-fitting if one exploits the onset of numerical linear dependence in the AO product basis itself. This type of *ab initio* DF originated in the earlier idea[40,41] of approximating the two-electron integral matrix by an incomplete Cholesky decomposition (CD), thus Eq. (*9*)

$$(\mu\nu|\lambda\sigma) = \sum_{m=1}^{M} L_{\mu\nu}^{m} L_{\lambda\sigma}^{m} \qquad (11)$$

where *L*'s are the so-called Cholesky vectors (ChVs). The number of ChVs, *M*, varies between 4–6 times the number of functions in the AO basis set for very accurate calculations[42]. The main advantage of standard CD over RI approximation is that one can effectively control the accuracy of the integral representation by means of the threshold used for the incomplete decomposition. This is a very important property especially in the context of the CC hierarchy of methods, as it helps preserving the systematic improvability of such quantum chemical models. On the other hand, the generation of the ChVs through CD of the two-electron integral matrix is in general more time-consuming than computing the *R*'s vectors through Eq. (*9*). The use of *ab initio* DF removes this bottleneck by requiring only CDs of each atomic subblock of the integral matrix[43] to define the auxiliary basis set needed in Eq. (*9*). Next to the computational advantage, this type of *ab initio* DF guarantees in practice the same accuracy control as in standard CD, and it has opened the way for the definition of analytical derivatives of the ChVs [44].

In our current implementation of the CCSD algorithm, the standard CD approximation[41] is used. On the other hand, the algorithm is designed to work just as well with any DF integral representation if the **R** vectors are made available. As already mentioned, our algorithm is MO based which means that we work with ChVs transformed to the MO basis. Compared to much more demanding parts (*vide infra*), this $\propto N^3M$ scaling of the transformation procedure requires a negligible fraction of the computer time.

Unlike in Hartree–Fock (HF) and Kohn–Sham density functional theory (KS-DFT), the integral representations of Eq. (*9*) and Eq. (*11*) do not open a possibility to factorize the computationally most demanding terms in CCSD, and leave unaltered the formal $\propto N^6$ scaling of the number of floating point operations. This is due to the fact that, say in A_{vv}, reconstruction of integrals from the ChVs prior to the contraction with the *t* amplitudes is

more efficient than the contraction of t with ChVs followed by the subsequent contraction of the resulting intermediate again with the ChVs. Direct contraction of amplitudes with ChVs can reduce the scaling of the given term, if both molecular orbital indexes of $L^m_{p,q}$ are contracted simultaneously, what is however not in general the case for A_{vv}. It remains unexplored the idea[45] to reduce the computational scaling of CCSD by extracting a "nonredundant" subset of the doubles manifold through CDs of properly chosen (metric) matrices. Nevertheless, CD offers an interesting possibility to substantially alleviate the $\propto N^4$ scaling of the storage requirements. In particular, this is crucial when there is a good reasoning to work with the MO based algorithm (e.g. use of OVOS). For illustration, Table I shows the theoretical progression of the storage requirements for systems with increasing number of electrons and/or basis functions.

As it is evident, with nowadays standard computers (4-8Gb RAM) several $\propto O^2V^2$ size objects can be kept in core for calculations up to about 1000 basis functions and not too many electrons. About 800–1000 basis functions is more or less the upper limit for full (effective) storage of V^4 2-electron integrals on a disk. However, it is not only the storage itself, but mainly the related I/O operations count that slows down the calculations considerably. The difference with the demands to store the ChVs is more than evident.

Virtual Orbital Space Segmentation

Our basic philosophy in the present CCSD code can be summarized as "flexible adjustment to the available RAM and to the number of parallel processors". This is accomplished via splitting the VO space into N′ segments of V′ (or less) VOs. For instance, the t^{ij}_{ab} biexcitation amplitude matrix ($\mathbf{T}^{00}_{vv}$) then splits into N′(N′+1)/2 blocks.

TABLE I
Typical sizes of objects (in GB) appearing in the CCSD calculation for selected model systems

Objects | Scaling	# OOs # VOs ChV (M)	20 500 2500	40 500 2500	40 800 4000	80 1200 6000
T^{ab}_{ij}; (ai|bj) | $O^2V^2/2$		0.4	1.5	3.8	34.3
(ab|cd) | $V^4/8$		58.2	58.2	381.5	1931.2
ChV L^m_{pq} | $MN^2/2$		0.3	0.3	1.3	4.6

$$\mathbf{T}^{00}_{\mathrm{vv}} = \begin{pmatrix} \mathbf{T}^{00}_{11} & & & \\ \mathbf{T}^{00}_{21} & \mathbf{T}^{00}_{22} & & \\ \vdots & \vdots & \vdots & \\ \mathbf{T}^{00}_{\mathrm{N}\,1} & \mathbf{T}^{00}_{\mathrm{N}\,2} & \cdots & \mathbf{T}^{00}_{\mathrm{N}\,\mathrm{N}'} \end{pmatrix} \tag{12}$$

For a closed shell case, the lower triangle of the matrix is relevant because of the permutation symmetry of the amplitudes $t^{ij}_{ab} = t^{ji}_{ba}$. Each block of the amplitudes $\mathbf{T}^{00}_{\mathrm{A}\,\mathrm{B}'}$ contains all t^{ij}_{ab} with $a \in \langle (A'-1)*V'+1, A'*V' \rangle$ and $b \in \langle (B'-1)*V'+1, B'*V' \rangle$. Similarly, all other quantities are stored, e.g. ChVs transformed to MO basis for various combinations of occupied and virtual indices $\mathbf{L0}^{m}_{ij}$, $\mathbf{L1}^{m}_{ai}$, $\mathbf{L2}^{m}_{ab}$ (0, 1 and 2 denotes the number of VO indexes).

For certain terms, several V^4 integral/intermediate blocks should simultaneously be kept in core. Certainly, this part is (much) more memory demanding than other terms involving integrals that bear also indices of OOs. If we would restrict ourselves to a single segmentation of the virtual space, the maximum block of $(V')^4$ would determine the memory demand. However, in such case this would be an unnecessary fine segmentation for the aforementioned, less memory demanding quantities. As a consequence, the too small segments would decrease the efficiency of the related matrix multiplications and I/O traffic. In order to keep the possibly highest efficiency with the available core memory, we have introduced a second level segmentation, merely related to V^4 integrals (labeled N″). This means that each segment of VO space, containing V' orbitals can be further split into N″ parts.

Two Alternative Computational CCSD Schemes

Irrespective of the VO space segmentation, the closed shell CCSD formulation as given by Scuseria et al.[32] provides the base for minimal operation count algorithm. Disregarding the $\propto N^5$ dependences, the floating point operations count is dominated by $\propto 4\ O^3V^3$, and $\propto O^2V^4/4$ steps and a multiple reading of O^2V^2 size data. The real cost depends on the actual O/V ratio, and, if $O \geq 1/16\ V$ (for instance, O = 50 and V = 800), 4 O^3V^3 represents more arithmetical operations than $O^2V^4/4$ step. In addition to a standard algorithm[32], reconstruction of integrals over MOs from the ChVs must be considered. For our target systems it was feasible (and mostly advantageous) to store on disk all the MO integrals that bear one or more indices of occupied orbitals. Hence, those MO integrals are calculated in a single step prior

to the CC iterations and used whenever it is effective (up to terms, for which it is more efficient to refactorize them via direct contraction of the amplitudes and the ChVs). V^4 MO-integral block evaluation involves at least $\propto 1/8$ MV^4 operations. One has to keep in mind that as soon as M is greater than 1/2 O^2 – which is often the case – recalculation of V^4 integrals is more expensive than the subsequent $O^2V^4/4$ step. We have again adopted a flexible strategy by implementing two algorithms that can be selected according to the actual available computer resources. In the following we denote them as

– Algorithm "V" that starts from the MO integrals precalculated from ChVs and stored locally on disk. This is the most efficient choice whenever sufficient (and fast enough), preferably local, data storage is available. Needless to stress that the pre-CCSD steps, like SCF, integral transformation from AOs to MOs, and eventually OVOS (*vide infra*), are dramatically accelerated by the use of Cholesky decomposition. Moreover, the blocked-virtual structure mentioned above is very easily manageable using blocked ChVs, too. Efficient parallelization can be introduced, in particular regarding the steps that involve (ab|cd) type integrals. Different blocks of the (ab|cd) integrals can be calculated and stored on individual nodes and being used on these nodes in each iteration (*vide infra*).

– Algorithm "M" when the four virtual-index integrals are reconstructed "on-the-fly" from ChVs in each CCSD iteration. Of course, as compared to "V", this algorithm gives rise to a significant increase of the theoretical prefactor in timing of each iteration 1/2 MV^4 extra arithmetic operations. However, the advantage of this algorithm resides in the minimum storage requirements, absence of the preparation step present in algorithm "V" and more efficient utilization of shared-memory parallelization on multi-core/processor systems.

To illustrate the differences in implementations of algorithms "V" and "M", we can analyze the O^2V^4 term arising from A_{vv} part of the double-excitation equations, Eq. (*3*). According to algorithm of Scuseria et al.[32], product of (anti-)symmetrized (t_1-contracted) 2-electron integrals, b_{cd}^{ab},

$$b_{cd}^{ab}(\pm) = \frac{1}{2}(b_{cd}^{ab} \pm b_{dc}^{ab}) \tag{13}$$

and τ-biexcitation amplitudes τ_{cd}^{ij} ($\tau_{cd}^{ij} = t_{cd}^{ij} + t_c^i t_d^j$)

$$\tau_{cd}^{ij}(\pm) = \frac{1}{2}(\tau_{cd}^{ij} \pm \tau_{cd}^{ji}) \tag{14}$$

is calculated via a symmetric, **S**, and skew-symmetric matrix **A**

$$b_{cd}^{ab}\tau_{cd}^{ij} = S_{ab}^{ij} + A_{ab}^{ij} \tag{15}$$

defined as

$$S_{ab}^{ij} = 2\sum_{c>d} b_{cd}^{ab}(+)\tau_{cd}^{ij}(+) + \sum_{c} b_{cc}^{ab}(+)\tau_{cc}^{ij}(+) \tag{16}$$

$$A_{ab}^{ij} = 2\sum_{c>d} b_{cd}^{ab}(-)\tau_{cd}^{ij}(-)\,. \tag{17}$$

The left column in Table II schematically shows the "M" ("on-the-fly") algorithm, while the right column compares the integral precalculation algorithm "V". Obviously, there is a price to be pay for introducing the segmentation of VOs. As it can be partially deduced from Table II, in both algorithms certain I/O and CPU (arithmetic) operations overhead is unavoidable. This is summarized in Table III. In both algorithms, multiple reading and writing of the $\mathbf{T}_2$ amplitudes depends on N′, i.e. the "large" segmentation, while reading of V^4 integrals in algorithm "V" and recalculation of these integrals in algorithm "M" do not require any overhead. In the case of algorithm "V", there is a multiple reading of the OV^3 integrals which depends on both "large" and "small" segmentation, i.e. N′ and N″, while in algorithm "M" "large" segmentation causes multiple reading of the $\mathbf{L1}_{ai}^{m}$, and $\mathbf{L2}_{ab}^{m}$ ChVs. The O^3V^3 term in CCSD equation suffers from the same overhead for both algorithm, mainly 3N′ multiple reading of O^2V^2 arrays or (N′+1) multiple reading of the $\mathbf{L1}_{ai}^{m}$ ChVs of MOV size. The effect of increasing both segmentations on "real" wall-clock timings is discussed in Section CCSD – Parallelization Timings.

Parallelization

Parallelization of the code is based on the Global Arrays (GA)[46] package. In the present implementation we limited ourselves mostly to the GA "equivalents" of the standard MPI directives. GA allocation of the arrays distributed across the nodes was merely used in the perturbative triples section for cre-

ating "task lists". A great advantage of such a pragmatic approach is in its independence from the particular MPI implementation.

The first level parallelization does not rely on the standard "master-slave" scheme, instead, all of the computer nodes are considered as equivalent. If more cores/processors are available on a node, a second level parallelization

TABLE II
Schemes of the algorithms "M" ("on-the-fly" reconstruction of MO integrals) and "V" (integral precalculation) for calculation of the O^2V^4-scaling term. For simplicity some of the processes that require less than $\propto N^5$ arithmetical operations are omitted

Algorithm "M":	Algorithm "V":
do a'=1,N'	do a'=1,N'
read $\mathbf{L1}^{m}_{a'i}$	
do b'=1,a'	do b'=1,a'
read $\mathbf{T}^{ij}_{a'b'}$, $\mathbf{L1}^{m}_{b'i}$	read $\mathbf{T}^{ij}_{a'b'}$
make $\tau^{ij}_{a'b'} \leftarrow \mathbf{T}^{ij}_{a'b'} + \mathbf{T}^{i}_{a'}.\mathbf{T}^{j}_{b'}$	make $\tau^{ij}_{a'b'} \leftarrow \mathbf{T}^{ij}_{a'b'} + \mathbf{T}^{i}_{a'} . \mathbf{T}^{j}_{b'}$
do c'=1,N'	do c'=1,N'
do d'=1,c'	do d'=1,c'
read and update[a] $\mathbf{L2}^{m}_{a'd'}$, $\mathbf{L2}^{m}_{c'd'}$, $\mathbf{L2}^{m}_{a'c'}$, $\mathbf{L2}^{m}_{b'c'}$	
for c'' in c' segment	for c'' in c' segment
for d'' in d' segment (c''≥d'', if c'=d')	for d'' in d' segment (c''≥d'', if c'=d')
read $\mathbf{T}^{ij}_{c''d''}$,$\mathbf{T}^{ij}_{d''c''}$	read $\mathbf{T}^{ij}_{c''d''}$,$\mathbf{T}^{ij}_{d''c''}$
for a'' in a' segment	for a'' in a' segment
for b'' in b' segment (a''≥b'', if a'=b')	for b'' in b' segment (a''≥b'', if a'=b')
	read (b''d''|a''i),(a''c''|b''i),(a''d''|b''i),(b''c''|a''i)
	read (a''c''|b''d''),(b''c''|a''d'')
	update: (a''c''|b''d'') ← –(b''d''|a''i).$\mathbf{T}^{i}_{c''}$ –(a''c''|b''i).$\mathbf{T}^{i}_{d''}$
mult: (a''c''|b''d'')=$\mathbf{L2}^{m}_{a''c''}$. $\mathbf{L2}^{m}_{c''d''}$	update: (b''c''|a''d'') ← –(a''d''|b''i).$\mathbf{T}^{i}_{c''}$ –(b''c''|a''i).$\mathbf{T}^{i}_{d''}$
mult: (b''c''|a''d'')=$\mathbf{L2}^{m}_{b''c''}$. $\mathbf{L2}^{m}_{a''d''}$	
update: $\mathbf{T(+)}^{ij}_{c''d''} \leftarrow \mathbf{b(+)}^{c''d''}_{a''b''}\ \tau(+)^{ij}_{a''b''}$	update: $\mathbf{T(+)}^{ij}_{c''d''} \leftarrow \mathbf{b(+)}^{c''d''}_{a''b''}\ \tau(+)^{ij}_{a''b''}$
update: $\mathbf{T(-)}^{ij}_{c''d''} \leftarrow \mathbf{b(-)}^{c''d''}_{a''b''}\ \tau(-)^{ij}_{a''b''}$	update: $\mathbf{T(-)}^{ij}_{c''d''} \leftarrow \mathbf{b(-)}^{c''d''}_{a''b''}\ \tau(-)^{ij}_{a''b''}$
end of a'',b'' loops	end of a'',b'' loops
write $\mathbf{T}^{ij}_{c''d''}$,$\mathbf{T}^{ij}_{d''c''}$	write $\mathbf{T}^{ij}_{c''d''}$,$\mathbf{T}^{ij}_{d''c''}$
end of all other loops	end of all other loops

[a] Update **L2** by T_1 contraction, e.g. $\mathbf{L2}^{m}_{a'b'} \leftarrow \mathbf{L1}^{m}_{a'i}\,\mathbf{T}^{i}_{b'}$.

is achieved within each node by utilizing multi-thread BLAS routines. Doing so, OpenMP parallelization of the matrix–vector (dgemv) and matrix–matrix multiplication (dgemm) is automatically taken care of. Since the algorithm is dominated by matrix–matrix multiplications, relatively high efficiency can be achieved on multi-core/processor nodes.

The first level – distributed memory – parallelization is tightly related to the N′ segmentation of the VO space. As indicated in Table II, this is easily realized via splitting the loops over the VO segments into independent parallel tasks. For most of the cases the distribution of these tasks is driven by segmentation of the two outermost loops, however, for some $O^3V'^3$ terms it is controlled via splitting of the uppermost and the third loops. Naturally, all other loops over VOs within each parallel task are segmented as well. Hence, for a certain N′ VO space segmentation there are $(N')^2 \propto O^4V'^2$, $(N')^2 \propto O^3VV'^2$ and $(N')^2/2 \propto O^2V^2V'^2$ independent tasks available for parallelization. The terms scaling $\propto N^5$ (not analyzed in details in this article) are split into independent tasks analogously to the $\propto O^3V^3/\propto O^4V^2$ scaling terms, thus not affecting noticeably the overall parallel load-balancing. In order to accomplish an ideal task distribution a specific number of nodes is related to a given segmentation (Table IV). Each entry represents the relative efficiency with respect to the "ideal" job distribution. At the end of each CCSD iteration 1/2 O^2V^2 $\mathbf{T}_2$ amplitudes have to be gathered and redistributed over all the nodes. This step can be a bottleneck mainly for large number of

TABLE III
Summarization of the most demanding I/O and arithmetical operation overhead resulting from the VO segmentation in evaluation of O^2V^4-scaling terms for "V" and "M" algorithms. (R) and (W) stand for read and write, respectively

Algorithm "V"		Algorithm "M"	
I/O			
1/4 N′(N′+1).O^2V^2	(R+W)	1/4 N′(N′+1).O^2V^2	(R+W)
$\propto$N′N″.OV^3 [a]	(R)	(N′+1).MOV	(R)
1/2 V^4	(R)	$\propto$N′(N′+1).MV^2 [a]	(R)
Arithmetic			
–		1/2 MV^4	

[a] Prefactor cannot be determined exactly, since the algorithm exploits certain redundancies in reading, e.g. when the data are already loaded in the memory.

nodes connected via slow network, especially for smaller jobs with a relatively small number of arithmetic/CPU operations.

The crucial pre-CCSD step in algorithm "V" includes precalculation of integrals over the molecular orbitals from the ChVs. The idea behind the parallelization of Cholesky decomposed 2-electron integral based modules in MOLCAS (SCF, RASSCF, MP2, ...) is to generate, store and use the ChVs by splitting the ChV auxiliary index (m′) over parallel nodes, with a local dimension for m′ being M_{local}. For MO based CC calculation, however, alternative strategy of parallelization must be applied. The efficiency of the particular execution significantly depends on the number of nodes, network bandwidth and the number of cores/processors per node.

1) When e.g. V^4 MO integrals are required, they have to be reconstructed from the partial contributions produced on each node, $(a'b'|c'd')'$,

$$(a'b'|c'd')' = \sum_{m'=1}^{M_{\text{local}}} L^{m'}_{a'b'} L^{m'}_{c'd'} \qquad (18)$$

TABLE IV
Theoretical efficiency (in %) of the parallelization as a function of virtual orbital space segmentation and number of the parallel computational nodes

N′	# of task	Number of nodes										
		2	3	4	5	6	7	8	other			
2	4[a]	100	67	100								
	2[b]	100	67	50								
									9	10	12	16
4	16[a]	100	89	100	80	89	76	100	89	80	67	100
	8[b]	100	89	100	80	67	57	100	89	80	67	50
									12	16	32	64
8	64[a]	100	97	100	98	97	91	100	89	100	100	100
	32[b]	100	97	100	91	89	91	100	89	100	100	50
									16	32	64	128
16	256[a]	100	99	100	98	99	99	100	100	100	100	100
	128[b]	100	99	100	98	97	96	100	100	100	100	100

[a] Tasks scaling as $\propto O^3VV'^2$ or $\propto O^4V'^2$. [b] Tasks scaling as $\propto O^2V^2V'^2$.

and summed up afterwards across all computational nodes. This algorithm can be mainly efficient for modest number of nodes connected with fast network, or for very large number of ChVs. The main drawback is the necessity to transfer a huge number of V^4 MO integrals over the nodes.

2) In the second alternative, merely O^2V^2 or smaller blocks of 2-electron integrals are calculated in parallel as in i) and subsequently stored on each node. Other types of MO integrals (i.e. OV^3 or V^4) are precalculated on each node from the complete set of ChVs gathered from all nodes. Naturally, merely a fraction of these integrals is needed for a parallel subtask locked to a particular node. This is typically an optimal choice. Optionally, all the integrals can be precalculated in this manner. Obviously, by having the "needed" fraction of integrals on each node means that one has to give up the full integral permutation symmetry, which gives rise to doubled number of calculated integrals. This fact, together with some other unavoidable duplicities makes this option disadvantageous for small number of parallel nodes. By going to larger number of nodes the segmentation of the VO space is increased and the data storage reduction on each node begins to scale almost linearly with respect to N_{nodes}. In the limit of "infinite" number of nodes, which is practically achieved already at 32 nodes, only $V^4/(2N_{nodes})$ are needed to be stored on each node. As an example, the CCSD calculation of benzene dimer in aug-cc-pVQZ basis set required only 140 GB of the local scratch space on each of the 242 parallel nodes, instead of 4.8 TB the full set of the V^4 integrals. This argument raises doubts about the accepted opinion that large CCSD calculations can be solely carried out within the integral-direct AO based approach.

Perturbative Correction Due to Triple-Excitations

The idea of VO space blocking has been essentially adopted from the algorithm for triples as suggested earlier[47]. Despite formally the RHF based algorithm comprises less operations, it turned out that even for the closed shell case the UHF based algorithm can be more effective due to (i) the fact that handling the permutation symmetry is much simpler, (ii) the simpler structure gives rise to much more effective use of the (second parallelization level) BLAS3 matrix multiplications because of well defined and large enough dimensions of the matrices and (iii) the formal reduction of the number of operations via presummed matrices[47] can be to a large extent utilized without storing huge files.

In this section we recapitulate the basic feature of the implementation as it has been transfered to MOLCAS. The main contribution to the whole "(T)" correction of CCSD(T)[1]

$$E_{(T)} = E_{T(CCSD)} + E_{ST}^{[5]} + E_{DT}^{[4]} \tag{19}$$

is the T(CCSD)[48] given as

$$E_{T(CCSD)} = \sum_{i>j>k} \sum_{a>b>c} D_{ijk}^{abc} (t_{ijk}^{abc})^2 \tag{20}$$

completed by the 5th order contribution

$$E_{ST}^{[5]} = \sum_{i,a} t_a^i x_i^a ; \qquad x_i^a = \sum_{j>k,\, b>c} \bar{g}_{jk}^{bc} t_{abc}^{ijk} \tag{21}$$

and, when the $f_i^a \neq 0$, also the 4th order contribution[49]

$$E_{DT}^{[4]} = \sum_{i,a} f_i^a y_a^i ; \qquad y_a^i = \sum_{j>k,\, b>c} t_{bc}^{jk} t_{abc}^{ijk} . \tag{22}$$

Computational demands are dictated by determining the t_{abc}^{ijk} amplitudes, in our algorithm calculated according to a very simple expression using supermatrices

$$D_{ijk}^{abc} t_{abc}^{ijk} = \sum_{r}^{O+V} \sum_{def} \sum_{lmn} \bar{K}_{ab}^{lr} \bar{L}_{rc}^{mn} \bar{\delta}_{lmn}^{ijk} \bar{\delta}_{abc}^{def} \tag{23}$$

where matrices $\bar{\mathbf{K}}$ and $\bar{\mathbf{L}}$

$$\begin{aligned} \bar{K}_{ab}^{ir} &= -\bar{K}_{ba}^{ir} = -t_{ab}^{ir} && \text{for } r \in \langle 1, O \rangle \\ \bar{K}_{ab}^{ir} &= \bar{K}_{ba}^{ir} = \bar{g}_{ab}^{i(r-O)} && \text{for } r \in \langle O+1, O+V \rangle \end{aligned} \tag{24}$$

$$\begin{aligned} \bar{L}_{rc}^{jk} &= -\bar{L}_{rc}^{kj} = \bar{g}_{rc}^{jk} && \text{for } r \in \langle 1, O \rangle \\ \bar{L}_{rc}^{jk} &= -\bar{L}_{rc}^{kj} = t_{(r-O)c}^{jk} && \text{for } r \in \langle O+1, O+V \rangle \end{aligned} \tag{25}$$

and

$$\overline{\delta}^{pqr}_{stu} = \overline{\delta}^{pq}_{st}\delta^{r}_{u} + \overline{\delta}^{qr}_{st}\delta^{p}_{u} + \overline{\delta}^{pr}_{st}\delta^{q}_{u}. \tag{26}$$

Within the UHF, or more generally "different orbitals for different spins" based algorithm one has to calculate t^{ijk}_{abc} for ααα (βββ) and ααβ (ββα) spin case blocks while obviously those in parentheses are merely needed for the true open shell systems.

In order to distinguish the spinorbitals in this section, we shall use capital letters to denote α spinorbitals, whereas lower case letters are reserved to β spinorbitals. The ααβ block can be calculated as[47]

$$D^{ABc}_{IJk}\, t^{IJk}_{ABc} = \sum_{R} (\overline{K}^{IR}_{AB} + \overline{K}^{JR}_{AB})(L^{Jk}_{Rc} - L^{Ik}_{Rc}) + \tag{27}$$

$$+ \sum_{r} [(K^{Ir}_{Ac} + K^{Jr}_{Ac})(L^{kJ}_{rB} - L^{kI}_{rB}) - (K^{Ir}_{Bc} + K^{Jr}_{Bc})(L^{kJ}_{rA} - L^{kI}_{rA})] + \tag{28}$$

$$+ \sum_{R} (K^{kR}_{cA}\overline{L}^{IJ}_{RB} - K^{kR}_{cB}\overline{L}^{IJ}_{RA}) + \tag{29}$$

$$+ \sum_{R} (\overline{K}^{IR}_{AB} L^{Ik}_{Rc} - \overline{K}^{JR}_{AB} L^{Jk}_{Rc}) + \tag{30}$$

$$+ \sum_{r} (K^{Ir}_{Ac} L^{kI}_{rB} - K^{Ir}_{Bc} L^{kI}_{rA} - K^{Jr}_{Ac} L^{kJ}_{rB} + K^{Jr}_{Bc} L^{kJ}_{rA}). \tag{31}$$

At the first glance this unconventional formulation seems to be awkward and overcomplicated. However, it gives rise to a substantial reduction of floating point operations since individual terms under Eqs (*30*) and (*31*) scale as $\propto O^2V^3(O + V)$ and can be precalculated. By this trick the (leading) number of operations for this spin block reduces from $\propto 2O^3V^3(O+V)$ to $\propto 5O^3V^3(O+V)/4$, i.e. by a factor of 1.6. A similar trick can be done also for the ααα (βββ) block reducing the formal scaling from $\propto O^3V^3(O+V)/4$ to $\propto O^3V^3(O+V)/12$.

Provided the **K** and **L** matrices are available on each node, our parallelized algorithm for this part is schematically described in Table V. The main advantage is that storage of terms in (Eqs (*30*)) and (*31*)) on disk is

TABLE V
Schematic description of the algorithm for the ααβ block of the T_3 amplitudes and their contribution to the total energy

	convention: Z′ ∈ VO segment seg_Z′; Z=A,B,c
Step 1:	Creation of the **K** and **L** matrices:
	for fixed VO segments A',B': $\bar{K}^{RM}_{A'>B'}$ stored
	for fixed VO segments c', A': $K^{rM}_{c'A'}/K^{Rm}_{C'a'}$ stored
	for VO segment A′/a′: $\bar{L}^{M>N}_{RA'}$ stored; $L^{mM}_{rA'}/L^{Mm}_{Ra'}$ stored
Step 2:	do seg_A′ = 1,N′; do seg_B′ = 1,seg_A′; do seg_c′ = 1,n′
	parallelization here: round-robin - do_it on this node?
Step 3:	if do_it = .true. - this step carried out on the node
	loads $\bar{L}^{M>N}_{RA'}$; $\bar{L}^{M>N}_{RB'}$; $L^{mN}_{rA'}$; $L^{mN}_{rB'}$; $L^{Mn}_{Rc'}$
	loads $\bar{K}^{RM}_{A'>B'}$; $K^{rM}_{c'A'}$; $K^{rM}_{c'B'}$; $\bar{g}^{M>N}_{A'>B'}$; $g^{Mn}_{A'c'}$; $g^{Mn}_{B'c'}$
	do k=1, O_β ; loads $K^{Rk}_{A'c'}$; $K^{Rk}_{B'c'}$ (fixed k)
Step 3a	$\Lambda^{M}_{A'B'c'} \leftarrow \bar{K}^{MR}_{A'B'}L^{Mk}_{Rc'} + K^{Mr}_{A'c'}L^{kM}_{rB'} - K^{Mr}_{B'c'}L^{kM}_{rA'}$ [see Eq. (*30*)) and (Eq. (*31*))]
Step 3b:	do I = 2,O_α; do J = 1,O_α-1
	$T3(A',B',c') \leftarrow \Lambda^{I}_{A'B'c'} - \Lambda^{J}_{A'B'c'}$
	$T3(A',B',c') \leftarrow K^{kR}_{c'A'}\bar{L}^{Ij}_{RB'} - K^{kR}_{c'B'}\bar{L}^{Ij}_{RA'}$
	$P(A',B',R) \leftarrow \bar{K}^{IR}_{A'B'} + \bar{K}^{JR}_{A'B'}$ $Q(R,c') \leftarrow L^{Jk}_{Rc'} - L^{Ik}_{Rc'}$
	$T3(A',B',c') \leftarrow P(A',B',R)*Q(R,c')$
	$P(A',c',r) \leftarrow K^{Ir}_{A'c'} + K^{Jr}_{A'c'}$; $Q(r,B') \leftarrow L^{kJ}_{rB'} - L^{kI}_{rB'}$
	$T3(A',B',c') \leftarrow P(A',c',r)*Q(r,B')$
	$P(B',c',r) \leftarrow K^{Ir}_{B'c'} + K^{Jr}_{B'c'}$; $Q(r,A') \leftarrow L^{kI}_{rA'} - L^{kJ}_{rA'}$
	$T3(A',B',c') \leftarrow P(B',c',r)*Q(r,A')$
	Calculated energy contribution and $T3(A',B',c') \leftarrow T3(A',B',c')/D^{A'B'c'}_{Ijk}$
Step 3c:	Partial contributions to x(k,c′), x(I,A′), x(I,B′), x(J,A′), x(J,B′)
	from T3(A′,B′,c′) [see Eq. (*21*)] and $\bar{g}^{I>J}_{A'>B'}$; $g^{Ik}_{A'c'}$; $g^{Ik}_{B'c'}$; $g^{Jk}_{A'c'}$; $g^{Jk}_{B'c'}$
	Partial contributions to y(k,c′), y(I,A′), y(I,B′), y(J,A′), y(J,B′)
	from T3(A′,B′,c′) [see Eq. (*22*)] and $\bar{K}^{I>J}_{A'>B'}$; $K^{I}_{A'c'}$; $K^{I}_{B'c'}$; $K^{J}_{A'c'}$; $K^{J}_{B'c'}$
	enddo k; I; J
	enddo seg_A′; seg_B′; seg_c′
Step 4:	master: gathers energy contributions from nodes
	gathers x and y and contracts with t and f [see Eq. (*21*) and Eq. (*22*)]

avoided and at the same time no repeated calculation of those terms is needed. In the current version, the parallelization philosophy has been taken as used in the DIRCCR12-OS code[50], i.e. that the **K** and **L** matrices are available for each node and the outer loops over the VO segments are controlled via round-robin scheduling. Such algorithm is ideal even for heterogeneous computer grids or differently loaded nodes. Alternatively, **K** and **L** matrices could be also distributed as the MO integrals in the previous section, and the outer loops controlled via a preselected distribution of jobs.

For the ααα (βββ) block the situation is more complicated and usage of the aforementioned trick requires huge intermediate matrices and the formal efficiency partially breaks off upon too many memory loading operations that are slow. Since computation of the pertinent block even using an algorithm based on direct utilization of Eq. (*23*) represents merely a minor part of the whole contribution from triples, at the moment we have retained the standard algorithm and have not worked out the code with the above mentioned trick. The level of parallelization remains as demonstrated for the ααβ block.

OVOS Algorithm

Optimized virtual orbital space method developed by Adamowicz et al.[51,52] reformulated and reimplemented by Neogrády et al.[26] generates modified VOs that accommodate more compact expansion of the correlated wavefunction. The actual algorithm is based on the iterative search for VOs that maximize the overlap of the 1st order correlated wavefunction expanded in the set of original and the truncated optimized VO space. In each iteration, new optimized orbitals are obtained by diagonalizing of the matrix **R**

$$\mathbf{U}^{\mathrm{T}}\mathbf{R}\mathbf{U} = \lambda \tag{32}$$

where **R** implicitly depends on **U**

$$R_{a^*b^*} = \sum_{ij}\sum_{cd} t_{ij}^{ca^*} t_{ij}^{db^*} P_{cd} \tag{33}$$

$$P_{ab} = \sum_{p^*} U_{p^*a} U_{p^*b} \ . \tag{34}$$

Asterisk superscript denotes virtuals that belong to the truncated optimized VO space. Factorization of the Eq. (*33*) leads to $\propto N^5$ (i.e. $\propto O^2V^3$) scaling procedure, which is by an order of magnitude faster then the consequent CCSD step.

To be able to go beyond several hundreds of basis functions and routinely carry out calculations with more than 2000 basis functions, similar VO index segmentation technique as in the CCSD step had to be introduced. Since the OVOS procedure can be factorized to steps with a single internal summation over VOs and the largest quantities that must be processed are of O^2V^2 size, it was possible to use a scheme in which only a single virtual index was segmented, while the second one had been left unsegmented for these arrays. This enabled to reduce the I/O overhead and to increase the robustness and the efficiency of the parallelized implementation.

In our closed shell algorithm, the calculation of **R** (see Eq. (*33*)) is actually carried out as a factorization of

$$R_{a^*b^*} = \sum_{ij} \sum_{cd} [\langle ij|a^*c\rangle (2\langle ij|b^*d\rangle - \langle ji|b^*d\rangle) P(c,d)] / (D_{ij}^{a^*c} D_{ij}^{b^*d}) \,. \qquad (35)$$

The scheme of the parallel algorithm for evaluation of **R** is depicted in Table VI. Algorithm starts with a preparation of blocks of biexcitation amplitudes due to the selected segmentation on all nodes. Then, a list of virtual segments (label as d', see Table VI) is assigned to each node. From there on, certain parts of the algorithm, like contraction of the amplitudes with matrix **P**, creation of $2\langle ij|b^*d\rangle - \langle ji|b^*d\rangle$, etc. are performed for one segmented virtual index locked to particular node. Finally, the contributions to the matrix **R** are evaluated as a contraction of two $O^2V'V$ matrixes (see Step 2d of alg. in Table VI) and summed up and gathered from all the nodes. Computationally insignificant steps, like the diagonalization of **R** are then performed on each node in a replicant mode.

Numerical Tests and Performance

OVOS

For a large molecule, the time required for obtaining optimized VOs is expected to be almost negligible with respect to the total cost of the OVOS-CCSD(T) calculation. Some examples illustrating scaling of the OVOS method with the number of correlated occupied and virtual orbitals and

with the number of parallel computational nodes are shown in the Table VII. Timings of the preparation step (O^2V^2 integrals precalculation from the MO ChVs, etc.) significantly depend on the number of ChVs (as discussed in Section Two Alternative Computational CCSD Schemes in algorithm "V"). In calculations presented in Table VII, the simplest parallel algorithm for OVOS was used, where the calculation on all nodes starts from the MO transformed complete set of ChVs (VO|M). With fast enough internode connection (≥1 Gb/s) more efficient algorithm is available which generates the O^2V^2 integrals from ChVs with segmented auxiliary index in the first

TABLE VI
Scheme of the parallelized evaluation of **R** matrix – the core of the OVOS algorithm

Step 1:	do a′=1,N′
Step 1a:	do c′=1,N′
	read (i,a′|j,c′) and divide by denominator
	complete (i,j,a′,c) ← (i,a′,j,c′)
	end do c′
Step 1b:	for d′ on this node:
	mult (i,j,a′,d′) ← (i,j,a′,c) . $\mathbf{P}_{cd'}$
	save (i,j,a′,d′) as record "a′d′_rec"
	next d′
	end do a′
Step 2:	for d′ on this node:
Step 2a:	do b′=1,N′
	read (i,b′,j,d′) from "b′d′_rec"
	complete (i,j,b,d′) ← (i,b′,j,d′)
	end do
Step 2b:	make (i,j,d′,b) ← 2(i,j,d′,b) – (j,i,b,d′)
Step 2c:	do a′=1,N′
	read (i,j,a′,d′) and divide by denominator
	complete (i,j,d′,a) ← (i,j,a′,d′)
	end do
Step 2d:	mult $\mathbf{R}_{ab}$ ← (i,j,d′,a) . (i,j,d′,b)
	next d′
	gather **R** matrix from nodes and redistribute it

TABLE VII
Illustrative timings for the processes involved in the OVOS procedure. O, V and ChV represent numbers of occupied and virtual orbitals and the number of Cholesky vectors, respectively

O	V	ChV	Nodes	Cores/CPUs per node	MP2	preparation min	iteration	total	scaling
30	786	3833	1[a]	4	8.0	6.8	4.1	48.0	
30	1419	6968	1	4	35.6	32.8	19.2	282.4	2.6[b]
42	786	3833	1	4	9.7	5.7	8.8	66.0	2.3[c]
42	862	3484	8[d]	2	22.6[e]	31.0	3.3	67.7	
42	1549	6203	8	2	119.8	167.5	17.5	443.0	2.85[b]
55	1203	6147	1[f]	8	49.4	24.5	34.9	333.2	
55	1203	6147	2	8		24.3	22.6	225.2	1.54[g]
55	1203	6147	4	8		23.6	16.3	172.2	2.14

[a] Intel Core2 Quad (4 Cores), 2.4 GHz, 8 GB RAM. [b] Assumed scaling formula kO^2V^x. [c] Assumed scaling formula kO^xV^3. [d] 2× Itanium-2 (Madison), 1.5 GHz, 10 GB RAM. [e] Timing on 1 node (2 CPUs). [f] 2× Intel Xeon E5345 (4 Cores), 2.33 GHz, 8 GB RAM. [g] Parallel scaling, speedup against the run on 1 node.

step similarly to Eq. (*18*), and then gathers the integrals from all the nodes. Results also clearly show that the timing of a single OVOS iteration is comparable with the timing for the MP2 calculation. Since only a minimum internode transfer ($\propto V^2$) is required after each OVOS iteration, almost linear scaling with the number of nodes can be expected. However, from the timings presented in Table VII it is clear that only 65% speedup is gained from going from one to two nodes, and 47% in going from one to four nodes. This is a consequence of several cumulative factors, like domination of I/O over CPU timing spent in the matrix–matrix multiplication very efficiently parallelized on 8 cores via multi-thread BLAS routines and presence of some parts that are either not fully parallelized or are executed in a replicant mode (diagonalization of **R**, etc.). As aforementioned, further optimization of the OVOS algorithm would not lead to any significant speedup in the total OVOS-CCSD(T) procedure, because the consequent steps scale with the size of the system by one (CCSD) or two (CCSD(T)) orders of magnitude steeper.

CCSD – Parallelization Timings

Speedup of a single CCSD iteration with increasing number of parallel nodes is shown in Fig. 1. Two sets of data representing the wall-clock speedup with respect to the calculation on 16 nodes are presented. One set

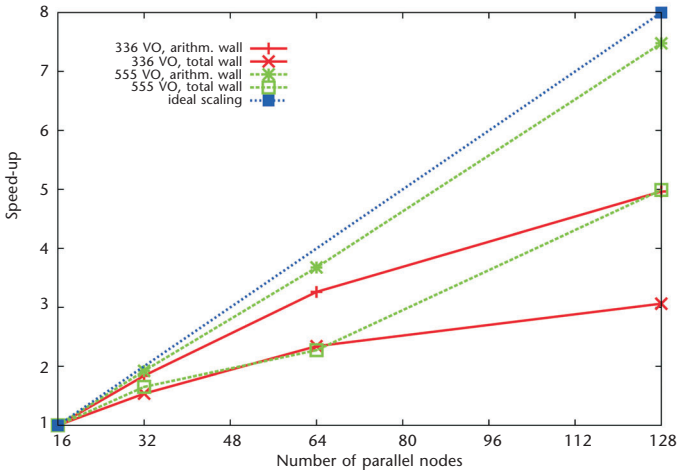

FIG. 1
Speedup of a single CCSD iteration calculation (algorithm "V") with increasing number of parallel computer nodes. Speedup in arithmetic (i.e. CPU) and total wall-clock timings is shown

is for a rather small calculation of the system with 30 correlated OOs and 336 VOs (benzene dimer, aug-cc-pVDZ basis set[31]), the other is for roughly by 50% larger system of the same number of correlated OOs and 555 VOs (benzene dimer, aug-cc-pVTZ basis set truncated by OVOS to ~70%). For each system two curves are showed. One represents the speedup in the "arithmetic" part of the CCSD iteration (including I/O overhead resulting from VO segmentation), which is basically the calculation of all terms (O^3V^3 and O^2V^4) without the network traffic due to summing up and redistributing of the new excitation amplitudes over all the nodes. By showing both the "arithmetic" and total wall-clock speedups we want to stress the fact that by going to larger systems "arithmetic" part scales as $\propto N^6$ while the network traffic only as $\propto N^4$. This is clearly demonstrated in comparison of the speedups for smaller and larger systems. Scaling of the "arithmetic" part for the larger system is almost linear (despite the I/O overhead resulting from larger VO segmentation) while the total speedup is improved only from 38 to 63%. We can conclude, that our implementation is thus not suitable for massively parallel calculation of "small" systems, but has the capacity to achieve satisfactory efficiency for calculation of large systems on a reasonable number of nodes, thanks to almost linear scaling of the computationally more demanding (O^3V^3/O^2V^4) part.

As aforementioned, our implementation takes advantage of a composite shared- and distributed-memory parallelization via multi-thread BLAS

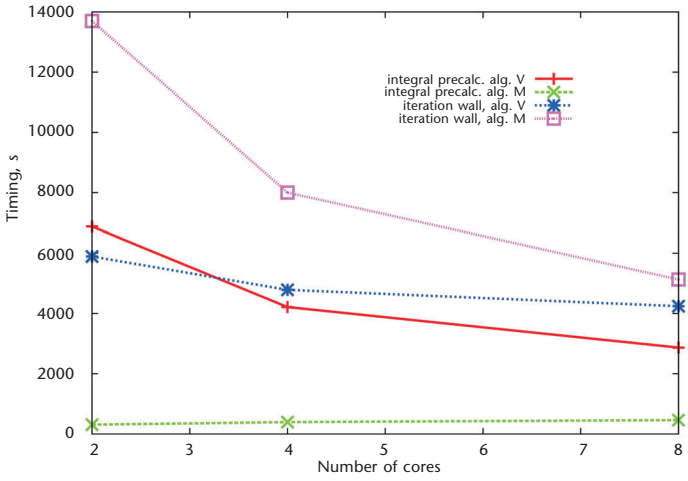

FIG. 2
Timings of the preparation step (prior to the CCSD iterations) and the CCSD iteration for algorithms "V" and "M" for increasing number of CPU cores per parallel process

matrix–matrix multiplication routines (dgemm). Figure 2 shows the scaling of the wall-clock time with increasing number of active CPU cores the integral precalculation step and for total CCSD iteration in "V" and "M" algorithms. The integral precalculation step is insignificant in the algorithm "M", since all the V^4 integrals are reconstructed "on-the-fly" within the CCSD iterations. In algorithm "V", 83% efficiency is achieved going from 2 to 4 cores and about 58% going to 8 cores, as measured with respect to the theoretical speedup. Decreasing of the efficiency is related to the I/O overhead, since writing of the produced portion of V^4 integrals on the node becomes limiting. More interesting is to look at the timings of the CCSD iteration. Efficiency of the algorithm "V" increases only slightly going from 2 to 4 and remains almost constant going to 8 cores. This has a lot of reasons, the most important being probably the ratio of the time spent in "arithmetics" versus I/O. Number of arithmetic operations is smaller compared to the integral precalculation step and, on the other hand, the amount of I/O is increased to 1/2 V^4. More favourable algorithm for the multi core/processor nodes is clearly the algorithm "M". 88% efficiency in going from 2 to 4 cores and 70% on 8 cores demonstrates the domination of the "arithmetics" in the CCSD iteration. Overall timing of the iteration on the 8 core nodes is now only slightly larger than the timing for the algorithm "V". The efficiency of the algorithm "V" on the multi-core/processor nodes has a potential to improve with increasing of the calculated system size due to the same interplay between the $\propto N^6$ "arithmetics" and $\propto N^4$ I/O. Algorithm "M" is certainly promising for SMP or hybrid-SMP architectures. Moreover, since algorithm "M" is purely matrix-matrix multiplication based, constantly evolving multi-thread BLAS routines efficiency could challenge the integral-direct AO based algorithms for medium sized systems.

To further validate the assumptions stated above, another example is shown in Fig. 3 and Table VIII. In Fig. 3 two curves are displayed representing the relative speedup and timings of the "arithmetic" and the "parallel overhead" part of the CCSD iteration. Calculation was done on 128 two-processor nodes (dual Intel 1.5 GHz Itanium-2 Madison), $QSNet^{II}$/Elan-4 ~7 Gb/s network) for benzene dimer (30 OOs) and different number of VOs versus calculation with 336 VOs. The speedup of the arithmetic part by increasing the system size from 336 to 976 VOs, converges to a value close to the number of processors/cores per node. The second curve corresponds to the speedup and wall-clock timings of the network transfer. Compared to the first curve, slowdown of the "overhead" part of the CCSD iteration is much less pronounced – on this particular network – when compared to the "arithmetic" part. Going from calculation with 762 to 976 VOs slow-

down of the "overhead" is practically constant, except for the "noise" caused probably by an interference with other calculations simultaneously running on the computer cluster. This supports our statement on increasing the efficiency of the algorithm on SMP (or hybrid-SMP) architectures with increasing the system size as a consequence of the increasing "arithmetics" versus "overhead" ratio.

TABLE VIII
The wall-clock timings (in min) of the arithmetic part, overhead ("non-arithmetic" part) and overall CCSD iteration for benzene dimer in OVOS-truncated aug-cc-pVXZ basis sets for several parallel calculation setups

Nodes	VOs	N'[a]	Wall clock arithm.	Wall clock overhead[a]	Wall clock total
32	336	16	2.7	0.8	3.6
128	555	16	3.7	2.4	6.1
128	976	16	24.2	7.2	31.3
242	762	22	6.6	13.3	19.9
242	1421	22	50.5	16.4	66.9

[a] Number of VO segments.

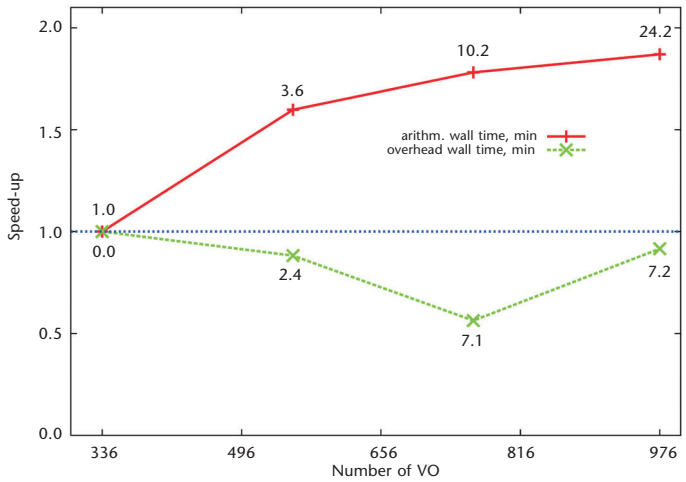

FIG. 3
Speedup and wall-clock timings of the arithmetic part and complete CCSD iteration

Table VIII presents a similar comparisons, but with different number of the parallel nodes of the same architecture. Comparison for benzene dimer calculation on 128 nodes of with 555 and 976 VOs (70% OVOS aug-cc-pVTZ and cc-pVQZ basis set) shows that in the first case, the "overhead" part of the CCSD iteration took almost 40% of the overall wall-clock time, while in the second case (976 VOs) it was only 23%. Keeping in mind the previous conclusions on close-to-linear scaling for the "arithmetic" part of the CCSD iteration, this means a significant improvement in overall scaling. For larger number of nodes, i.e. 242, and somewhat larger systems, 762 and 1421 (benzene dimer, aug-cc-pVTZ and aug-cc-pVQZ basis sets), the improvement on scaling is even more apparent. The "overhead" on calculation with 762 VOs represented almost 70% of the total wall-clock time, while for 1421 VOs it was only 25%.

Applications

The robustness and efficiency of parallel scaling of our (CD-OVOS)-CCSD(T) implementation can be demonstrated on several successful applications done throughout the past few years[20–23]. Most of the applications address the field of noncovalent interactions that are, for several reasons, "ideal" candidates for large-scale CCSD(T) applications. Among other reasons, noncovalent interactions are extremely challenging for accurate and reliable description at a lower theoretical level then CCSD(T), investigated systems are typically closed-shells of single-reference character and still not sufficiently well treatable by local-correlation based linear-scaling approaches. At the same time, noncovalent interactions are particularly important in typical bonding situations in biomolecules[53–55].

To such application belongs the benchmark calculations of the relative stabilities of several conformations of benzene dimer complex[20]. Calculations up to aug-cc-pVQZ basis set level, with more than 1400 VOs, were possible thanks to both the OVOS technique (truncation to about 850 VOs) as well as the tunable memory/disk requirements facilitated by our code. Interaction energy calculations of the uracil dimer in stacked and hydrogen-bonded conformation[21] (aug-cc-pVTZ basis set, 84 correlated electrons, 920 VOs truncated to ~570 VOs using OVOS) and stacked complex of methyladenine methylthymine[22] (aug-cc-pVTZ basis set, 110 correlated electrons, 1203 VOs truncated to ~750 VOs using OVOS) followed.

So far, the largest application carried out using our code was the CCSD(T) analysis of the many-body effects in the stacked dimer of guanine...cytosine (GCGC) hydrogen-bonded pairs[23] in 6-31G**(0.25, 0.15) basis set (modified

6-31G** basis set, with increased exponents of the d-function on "heavy" atoms, 0.25, and p-function on hydrogen, 0.15). To evaluate pair interaction energies and the post two-body nonadditivity terms, calculation of six dimers (all in supermolecular basis sets, 82–112 correlated electrons, 554–574 VOs), four trimers (140–154 correlated electrons, 525–535 VOs) and a single tetramer calculation (196 correlated electrons, 496 VOs) had to be done. The most demanding of these calculations, the GCGC tetramer, was still feasible on available parallel supercomputer architecture within reasonable time. A single CCSD iteration for the GCGC tetramer took about 1.17 h on 16 nodes (executed as 32 parallel processes), each constituted of two quad-core CPUs (Quad-Core AMD Opteron 2354, 2.20 GHz), and the overall wall-clock time for the CCSD step was less then one day (22.23 h). Computational time for the triples step was roughly by an order of magnitude higher, 286 h, but executing the (T) step as nine independent parallel jobs (each utilizing the same 32 parallel processes as in CCSD), made the (T) step done within 30 h.

CONCLUSIONS

In this paper we discussed the ideas of implementation of new-generation CCSD(T) code in MOLCAS program package. The driving force behind this effort was to provide a computer program which significantly lowers the demands on the computational resources (mostly the memory and disk storage) as well as being capable of efficiently utilize the multi-core/processor parallel architectures routinely available nowadays.

According to our preferences, we decided to stay as much as possible on the "safe ground" of MO-based approaches, without introducing any approximations leading to loss of the control of accuracy. Essentially, the only (controllable) approximation considered is the Cholesky-decomposition of the two-electron integrals, which was shown to be extremely beneficial in elimination of the storage bottleneck. Segmentation of the VO index in all important objects/intermediates is only a "technical" issue allowing us to break down the algorithm to block operations. This way we can tune the algorithm to fit basically in any (reasonably large) available computer memory for merely small I/O and CPU overhead.

Block design of the algorithm is tightly bound to its efficient parallelization. A particular segmentation of the VO index generates well defined number of independent parallel tasks that can be distributed among the computations nodes or parallel processes withing each CCSD iteration. The network traffic inevitable for the exchange of information between the

parallel processes after each CCSD iteration was shown to scale two orders of magnitude less steep than the CPU (or the "arithmetic") portion. This means that a proper balance between the number of parallel processes and the size of the calculated systems can ensure efficient utilization of a computer system with tens to hundreds of parallel computational nodes.

The OVOS technique is already known for its usefulness in many applications involving large VO spaces. From the point of view of the computational efficiency, use of OVOS leads to negligible computational overhead in combination with post-MP2 correlated methods. Its novel, massively parallel, Cholesky-decomposed two-electron based implementation makes it applicable to systems with more then 2000 basis functions.

This work was supported by the Slovak Grant Agency VEGA under the contracts No. 1/0428/09 and No. 2/0079/09. Support from EURATOM, contract No. FU07-CT-2006-00441 is also gratefully acknowledged. This work was also a part of research project No. Z40550506 of the Institute of Organic Chemistry and Biochemistry, Academy of Sciences of the Czech Republic and was funded by Grants No. LC512 and No. MSM6198959216 from the Ministry of Education, Youth and Sports of the Czech Republic. The support of Praemium Academiae, Academy of Sciences of the Czech Republic, awarded to P.H. in 2007 is also acknowledged.

REFERENCES

1. Raghavachari K., Trucks G. W., Pople J. A., Head-Gordon M.: *Chem. Phys. Lett.* **1989**, *157*, 479.
2. Neogrády P., Szalay P. G., Kraemer W. P., Urban M.: *Collect. Czech. Chem. Commun.* **2005**, *70*, 951.
3. Schütz M., Hetzer G., Werner H.-J.: *J. Chem. Phys.* **1999**, *111*, 5691.
4. Schütz M.: *J. Chem. Phys.* **2000**, *113*, 9986.
5. Schütz M., Werner H.-J.: *J. Chem. Phys.* **2001**, *114*, 661.
6. Subotnik J. E., Head-Gordon M.: *J. Chem. Phys.* **2005**, *123*, 064108.
7. Auer A. A., Nooijen M.: *J. Chem. Phys.* **2006**, *125*, 024104.
8. Christiansen O., Manninen P., Jørgensen P., Olsen J.: *J. Chem. Phys.* **2006**, *124*, 084103.
9. Flocke N., Bartlett R. J.: *J. Chem. Phys.* **2004**, *121*, 10935.
10. Noga J., Kutzelnigg W., Klopper W.: *Chem. Phys. Lett.* **1992**, *199*, 597.
11. Noga J., Kutzelnigg W.: *J. Chem. Phys.* **1994**, *101*, 7738.
12. Ten-no S.: *Chem. Phys. Lett.* **2004**, *398*, 56.
13. Tew D. P., Hättig C., Bachorz R. A., Klopper W. in: *Recent Progress in Coupled Cluster Methods* (P. Čársky, J. Paldus and J. Pittner, Eds), p. 535. Springer, New York 2010.
14. Werner H.-J., Adler T. B., Knizia G., Manby F. R. in: *Recent Progress in Coupled Cluster Methods* (P. Čársky, J. Paldus and J. Pittner, Eds), p. 573. Springer, New York 2010.
15. Noga J., Kedžuch S., Šimunek J., Ten-no S.: *J. Chem. Phys.* **2008**, *128*, 174103.
16. Janowski T., Ford A. R., Pulay P.: *J. Chem. Theory Comput.* **2007**, *3*, 1368.
17. Olson R. M., Bentz J. L., Kendall R. A., Schmidt M. W., Gordon M. S.: *J. Chem. Theory Comput.* **2007**, *3*, 1312.

18. Valiev M., Bylaska E. J., Govind N., Kowalski K., Straatsma T. P., van Dam H. J. J., Wang D., Nieplocha J., Apra E., Windus T. L., de Jong W. A.: *Comput. Phys. Commun.* **2010**, *181*, 1477.
19. Lotrich V., Flocke N., Ponton M., Yau A. D., Perera A., Deumens E., Bartlett R. J.: *J. Chem. Phys.* **2008**, *128*, 194104.
20. Pitoňák M., Neogrády P., Řezáč J., Jurečka P., Urban M., Hobza P.: *J. Chem. Theory Comput.* **2008**, *4*, 1829.
21. Pitoňák M., Riley K. E., Neogrády P., Hobza P.: *Chem. Phys. Chem.* **2008**, *9*, 1636.
22. Pitoňák M., Janowski T., Neogrády P., Pulay P., Hobza P.: *J. Chem. Theory Comput.* **2009**, *5*, 1761.
23. Pitoňák M., Neogrády P., Hobza P.: *Phys. Chem. Chem. Phys.* **2010**, *12*, 1369.
24. Aquilante F., Vico L. D., Ferre N., Malmqvist P.-Å., Neogrády P., Pedersen T., Pitoňák M., Reiner M., Roos B. O., Serrano-Andrés L., Urban M., Veryazov V., Lindh R.: *J. Comput. Chem.* **2010**, *31*, 224.
25. Rendell A. P., Lee T. J.: *J. Chem. Phys.* **1994**, *101*, 400.
26. Neogrády P., Pitoňák M., Urban M.: *Mol. Phys.* **2005**, *103*, 2141.
27. Taube A. G., Bartlett R. J.: *Collect. Czech. Chem. Commun.* **2005**, *70*, 837.
28. Taube A. G., Bartlett R. J.: *J. Chem. Phys.* **2008**, *128*, 164101.
29. Pitoňák M., Neogrády P., Kellö V., Urban M.: *Mol. Phys.* **2006**, *104*, 2277.
30. Pitoňák M., Holka F., Neogrády P., Urban M.: *J. Mol. Struct.* **2006**, *768*, 79.
31. Dunning T. H., Peterson K. A.: *J. Chem. Phys.* **2000**, *113*, 7799.
32. Scuseria G. E., Janssen C. L., Schaefer III H. F.: *J. Chem. Phys.* **1998**, *89*, 7382.
33. Hampel C., Peterson K. A., Werner H.-J.: *Chem. Phys. Lett.* **1992**, *190*, 1.
34. Koch H., Christiansen O., Kobayashi R., Jørgensen P., Helgaker T.: *Chem. Phys. Lett.* **1994**, *228*, 233.
35. Klopper W., Noga J.: *J. Chem. Phys.* **1995**, *103*, 6127.
36. Stanton J. F., et al.: *ACES II Program, A Product of the Quantum Theory Project*. University of Florida, 1993.
37. Werner H.-J., et al.: *MOLPRO, a Package of ab initio programs*, version 2009.2. 2009. http://www.molpro.net
38. Pedersen T. B., Aquilante F., Lindh R.: *Theor. Chem. Acc.* **2009**, *124*, 1.
39. Feyereisen M., Fitzgerald G., Komornicki A.: *Chem. Phys. Lett.* **1993**, *208*, 359.
40. Beebe N. H. F., Linderberg J.: *Int. J. Quantum Chem.* **1977**, *7*, 683.
41. Koch H., de Meras A. S., Pedersen T. B.: *J. Chem. Phys.* **2003**, *118*, 9481.
42. Boström J., Delcey M. G., Aquilante F., Serrano-Andres L., Pedersen T. B., Lindh R.: *J. Chem. Theory Comput.* **2010**, *6*, 747.
43. Aquilante F., Lindh R., Pedersen T. B.: *J. Chem. Phys.* **2007**, *127*, 114107.
44. Aquilante F., Lindh R., Pedersen T. B.: *J. Chem. Phys.* **2008**, *129*.
45. Aquilante F.: *Ph.D. Thesis*, Chap. 5. Lund University, Lund 2007. ISBN 978-91-7422-169-5.
46. Nieplocha J., Palmer B., Tipparaju V., Krishnan M., Trease H., Apra E.: *Int. J. High Perfom. Comp. Appl.* **2006**, *20*, 203.
47. Noga J., Valiron P.: *Mol. Phys.* **2005**, *103*, 2123.
48. Urban M., Noga J., Cole S. J., Bartlett R. J.: *J. Chem. Phys.* **1985**, *83*, 404.
49. Watts J. D., Gauss J., Bartlett R. J.: *J. Chem. Phys.* **1993**, *98*, 8718.
50. Noga J., Klopper W., Helgaker T., Valiron P.: 2003. A web repository is accessible on http://www-laog.obs.ujf-grenoble.fr/~valiron/ccr12/
51. Adamowicz L., Bartlett R. J.: *J. Chem. Phys.* **1987**, *86*, 6314.

52. Adamowicz L., Bartlett R. J., Sadlej A. J.: *J. Chem. Phys.* **1988**, *88*, 5749.
53. Hobza P., Havlas Z.: *Chem. Rev.* **2000**, *100*, 4253.
54. Rulíšek L., Havlas Z.: *J. Phys. Chem.* **1999**, *103*, 1634.
55. Fanfrlík J., Lepšík M., Horinek D., Havlas Z., Hobza P.: *ChemPhysChem* **2006**, *7*, 1100.