Performance of Parallel TURBOMOLE for Density Functional Calculations

MALTE VON ARNIM, REINHART AHLRICHS

Lehrstuhl für Theoretische Chemie, Institut für Physikalische Chemie, Universität Karlsruhe, D-76128 Karlsruhe, Germany

Received 28 February 1998; accepted 11 July 1998

ABSTRACT: The parallelization of density functional treatments of molecular electronic energy and first-order gradients is described, and the performance is documented. The quadrature required for exchange correlation terms and the treatment of exact Coulomb interaction scales virtually linearly up to 100 nodes. The RI-J technique to approximate Coulomb interactions (by means of an auxiliary basis set approximation for the electron density) even shows superlinear speedup on distributed memory architectures. The bottleneck is then linear algebra. Demonstrative application examples include molecules with up to 300 atoms and 3000 basis functions that can now be treated in a few hours per geometry optimization cycle in C_1 symmetry. © 1998 John Wiley & Sons, Inc. J Comput Chem 19: 1746–1757, 1998

Keywords: *ab initio*; density functional; parallelization; quantum chemistry; message passing

Introduction

Parallel computers are a mixed blessing. Their main advantage lies in the reduction of turnaround times; their disadvantages are in the usually increased price/performance ratio (as compared to single processor machines) and the difficulties encountered in the development of efficient parallel codes necessary to exploit the potential of parallel machines. The latter problem is especially serious in quantum chemistry where a variety of methods and algorithms are required that in turn are in a permanent state of flux to improve efficiency. Further, it is annoying that the existence of different machine architectures may require different strategies for parallelization.

A variety of parallel *ab initio* (self-consistent field SCF) codes is described in the literature.[1–5] All of these codes are designed for large systems on massive parallel architectures because all large data structures are distributed among the nodes. To our knowledge, up to now none of these meth-

Correspondence to: R. Ahlrichs
Contract/grant sponsor: Fonds der Chemischen Industrie

CCC 0192-8651 / 98 / 151746-12

ods have been applied routinely to the large systems (a few hundred atoms and a few thousand basis functions) for which they are intended.

Closely related to our work is the parallel density functional code of Guerra et al.[6] (ADF, Amsterdam density functional). A single point calculation of $Pt(PPh_3)CO$ with 105 atoms is reported in ref. 6. Another parallel density functional program is DMol[7]; it uses numeric basis functions defined on a grid rather than analytical basis functions as in TURBOMOLE; the largest benchmark calculation reported in ref. 7 is C_{20} with 280 basis functions.

In this article we describe the essential parallelization features of TURBOMOLE modules for SCF and density functional theory (DFT) calculations of molecular electronic energies and their gradients with respect to nuclear coordinates. We extend and improve previous work described elsewhere[8, 9] that concerned SCF treatments only.

We briefly specify the methodology of the TURBOMOLE SCF and DFT implementation as far as this is necessary for the subsequent presentations. Then we characterize the parallelization strategy and specify details of test molecules treated and computers used. The performance of the programs is presented for various molecular calculations that include up to 2754 basis functions and 294 atoms in C_1 symmetry. Special emphasis is given on the scaling behavior of computation times as a function of the number of processors and the molecular symmetry.

Program Structure for SCF and DFT Calculations and Their Gradients

DIRECT SCF (DSCF) AND GRAD MODULES

The DSCF module was originally developed for semidirect SCF calculations in which part of the two-electron integrals are stored on disk and the remaining ones are computed on the fly if necessary.[10] The first parallelization[8] concerned only the Fock matrix builder: the direct computation of two-electron integrals and their processing was distributed over available nodes or workstations. This parallelization was based on message passing (e.g., PVM or MPI) and data replication; the density matrix and the Fock matrix were kept at each node. Later refinements mainly concerned the parallelization of matrix algebra.[11] This part of the programs is described elsewhere; we only have improved performance by increasing the efficiency of the remaining sequential code. Because various tests showed only marginally (about 10% for 1000 basis functions) better performance of the semidirect mode over a fully direct code (all two-electron integrals treated directly), we support only the fully direct version.

For the DFT calculations one has to replace part of the exchange contributions to the Fock matrix and the total energy by the corresponding exchange-correlation contribution. This is achieved by a numerical integration (quadrature). For the purpose of the present article it is sufficient to write the two equations of interest. Let μ or ν denote basis functions (here always of the CGTO type, contracted Gaussian type orbitals) and $D_{\mu\nu}$ the density matrix in the closed shell case. (Open shell cases are a trivial extension from the technical point of view.) The one-particle density is then given as

$$\rho(\vec{r}) = \sum_{\mu\nu} D_{\mu\nu}\,\mu(\vec{r})\,\nu(\vec{r}). \qquad (1)$$

Letting r_i and w_i denote grid points and weights of the quadrature, the crucial step of a DFT treatment is then

$$E_{\mathrm{xc}} = \sum w_i f\big(\rho(\vec{r}_i)\big), \qquad (2)$$

which gives the exchange correlation contributions to the energy, E_{xc}, where $f(\rho(\vec{r}))$ specifies the DFT functional that may include generalized gradient terms, of course. A term similar to the E_{xc}, in eq. (2) gives the corresponding contribution to the Fock operator; because its treatment is completely analogous to that of eq. (2) it does not need to be specified in detail. The quadrature methods implemented in TURBOMOLE are described in ref. 12.

The evaluation of first-order gradients of the electronic energy with respect to nuclear coordinates is carried out in the molecule GRAD for SCF and DFT. The evaluation of the gradient is dominated by the computation and processing of derivative two-electron integrals (SCF) and the derivative of E_{xc}, eq. (2). For later purposes it is sufficient to consider eq. (2) as a typical representative of the computational tasks to be carried out for a treatment of E_{xc}.

On this level one treats all common DFT variants on the same footing (e.g., BP86,[13, 14] S-VWN,[15] B-LYP[13, 16] and hybrid functionals such as B3LYP[17]) are implemented in TURBOMOLE. We use the terms DSCF/DFT or GRAD/DFT in later context for a DSCF or GRAD calculation employing DFT

with a conventional treatment of Coulomb (and exchange) terms based on two-electron integrals.

RIDFT AND RDGRAD MODULES

In DFT one achieves, as a by-product, a separation of the treatment of Coulomb and exchange contributions to the electronic energy. This opens the way to more efficient evaluations of the Coulomb terms (than the exact conventional procedure implemented, e.g., in SCF programs) that have been exploited in many DFT implementations; very recent developments in this context are described in refs. 18–20.

In TURBOMOLE we take advantage of this feature by approximating the exact density as a linear combination of atom-centered auxiliary (fitting) basis functions, here denoted as α. Because this technique[21] bears similarities to the resolution of the identity we call it RI-J and the corresponding DFT program RIDFT, the gradient module RDGRAD. Details of our implementation, as well as the proper choice of the auxiliary basis sets, have been reported elsewhere.[22,23] The crucial equations of the RI-J approximations are as follows where approximations are indicated by a tilde:

$$\rho(\vec{r}) \approx \tilde{\rho}(\vec{r}) = \sum_{\alpha} \alpha(\vec{r}) C_{\alpha}, \tag{3}$$

$$\sum_{\beta} (\alpha|\beta) C_{\beta} = \gamma_{\alpha}, \tag{4}$$

$$\gamma_{\alpha} = \sum_{\mu\nu} (\mu\nu|\alpha) D_{\mu\nu}, \tag{5}$$

$$\mathbf{J}_{\mu\nu} \approx \tilde{\mathbf{J}}_{\mu\nu} = \sum_{\alpha} (\mu\nu|\alpha) C_{\alpha}, \tag{6}$$

where

$$(x|y) = \int x(1) r_{12}^{-1} y(2)\, d\tau. \tag{7}$$

In each SCF iteration the three-center integrals $(\mu\nu|\alpha)$ are needed twice [eqs. (5), (6)]. To increase efficiency we keep as many $(\mu\nu|\alpha)$ in memory as possible (those most expensive to evaluate) and recompute the remaining ones (twice) in each iteration.[22] The coefficient vector C_{α} (specifying the approximation $\tilde{\rho}$ of ρ and $\tilde{J}$ of J) is determined as a solution of the linear system of eq. (4). We do, of course, exploit symmetry and neglect sufficiently small integrals.[22]

The system of linear equations, eq. (4), is most conveniently solved by means of a Cholesky decomposition of the symmetric and positive definite matrix $(\alpha|\beta)$

$$(\alpha|\beta) = \sum_{\gamma} U_{\gamma\alpha} U_{\gamma\beta},$$
$$U_{\alpha\beta} = 0, \quad \text{if } \alpha > \beta. \tag{8}$$

The solution of eq. (4) then reduces to two simple substitution (forward and backward) procedures. Because the number of auxiliary basis functions, N_{aux}, is typically 2–3 times N_{BF} (the number of basis functions μ), here we must deal with large matrices; examples presented below include cases up to $N_{\mathrm{aux}} \approx 6000$. The computation of $U_{\alpha\beta}$ [eq. (8)] is an N^3 step, but it is the fastest linear algebra procedure: at most 50% of a corresponding matrix multiply or 5% of a diagonalization. The Cholesky decomposition has to be carried out only once and can be done ''in space'' (i.e., no scratch memory is required).

The evaluation of first-order gradients within the RIDFT approach is carried out in the module RDGRAD. The differentiation of RI-J terms is straightforward[22] and mainly involves the processing of integral derivatives $(\mu\nu|\alpha)'$. The exchange correlation contributions are treated exactly as in GRAD.

CHARACTERIZATION OF COMPUTATIONAL LOAD

For later discussions and a better understanding of the problems faced in the parallelization, we group the computational tasks in the following way.

Electron Repulsion Integrals (ERI)

This is the computation and processing of two-electron integrals in DSCF or DFT.

Quadrature

These are all steps executed in the numerical integration required for the treatment of E_{xc}, eq. (2), in DFT.

RI-J

These are the computations to execute eqs. (5) and (6).

Linear Algebra (LA)

This is the matrix diagonalization, acceleration of SCF convergence, etc.

Remainder (Rem)

Rem is mainly steps outside SCF iterations [e.g., evaluation of one-electron integrals, input and output of molecular orbitals (MOs), orthonormalization of start MOs]. Some steps inside the SCF iterations are also included (e.g., construction of density matrix, first-order orthonormalization of MOs, etc.).

In Table I we have collected single processor CPU times of an energy calculation for three representative cases in C_1 symmetry. Wall clock times are only a few percent larger. Gradient evaluations with TURBOMOLE are always faster than the corresponding SCF run; the load distribution is very similar, however. The ERI load depends somewhat ($\pm 10\%$) on the case (inclusion of exchange as in SCF or DFT hybrid functionals, or Coulomb only), but this is immaterial for the present considerations.

From the timings of Table I the following general picture emerges. ERI work dominates DSCF treatments with about 90% of the CPU time. The quadrature in DFT requires about 10% of the total or ERI times. RI-J requires only 10% of the ERI times and is comparable to quadrature. This holds for a direct treatment (all $(\mu\nu|\alpha)$ computed on the fly as needed); if all the integrals could be kept in memory, the RI-J timings would be further reduced by a factor[22,23] of 10 (see also below).

The load distribution can be rationalized by the scaling behavior of different steps, which is the increase of CPU times with increasing molecular size, N, for fixed atomic basis sets. If we distinguish between formal, asymptotic (large N), and real scaling (for $N_{BF} \approx 1000$), we get in this order: ERI: N^4, N^2, $\approx N^2$; quadrature: N^3, N, $\approx N^{1.5}$; RI-J: N^3, N^2, $\approx N^2$; and LA: N^3, N^3, N^3.

TABLE I.
Contributions of Different Steps to Single Node CPU Time (min) for Typical Cases; Calculations in C_1 Symmetry

	SiAl	Cd10	Mg16
ERI	4771	3963[a]	—
RI-J	445	382	27.6
Quadrature	632	241	27.9
LA	46	53	9.2
Remainder	8	11	2.2
t (DFT)[b]	5457	4268	—
t (RIDFT)[c]	1131	687	66.9

The abbreviations are explained in the text.
[a] Estimated from 16 nodes.
[b] Sum of ERI, quadrature, LA, and remainder.
[c] Sum of RI-J, quadrature, LA, and remainder.

The scaling behavior of RI-J applies for memory and direct mode, but a more complicated behavior results if one switches between modes if N is increased.

As a consequence of the load distribution one has to efficiently parallelize the work connected with ERI, quadrature, and RI-J. This is relatively easy to achieve. The problem is then the LA that already constitutes 5% of the CPU time for about 1000 basis functions in RIDFT treatments. Because LA is the only step with N^3 scaling, it will require an even larger share for cases with a few thousand basis functions. The problem here is that LA is more difficult to parallelize on distributed memory machines then RI-J or quadrature. An alternative approach to avoid diagonalization are the linear scaling methods (see refs. 24 and 25 and references therein).

As a consequence of the relative increase of CPU load connected with LA, we made an effort to optimize the corresponding routines (e.g., by loop unrolling and minimizing cache misses). We believe the code used to procure the data in Table I to be efficient. We also implemented the new effective core potential (ECP) routines including f functions from Breidung and Thiel.[26] Another concern was the determination of weights, w_i [eq. (2)], according to the Becke partition function technique to reduce the molecular to atomic quadratures.[27] Becke's prescription is a formal N^3 step, and the weight construction as originally implemented[12] exceeded the quadrature load for larger molecules (more than 2000 basis functions). It is possible to screen the weight determination (to avoid the computation and processing of near zeros), which results in an N^2 procedure yielding exactly the same weights on a finite accuracy computer.[28] Weight construction times are now always around a few percent of quadrature times.

We finally note that molecular symmetry is exploited in TURBOMOLE in a way that reduces CPU times by roughly the order of the molecular symmetry group.

Parallelization Strategies

GENERAL

The structure of parallel TURBOMOLE is basically as described in previous publications[8,9] as a master slave concept with message passing and

data replication. This is clearly not ideal for shared memory machines, but it has the advantage of portability to shared and distributed memory architectures. One of the slaves has a special function and is therefore denoted as the ''diagonalizer'': the diagonalizer node keeps all data and the complete code; the slaves keep only the necessary data and the code required for their tasks. The density matrix (required for most tasks) is broadcast by the diagonalizer to all slaves in every SCF cycle. The nodes compute their contribution to the Fock matrix that is then collected and accumulated by the diagonalizer. Serial tasks (not parallelized) are carried out only by the diagonalizer. The master organizes dynamical task distribution to the diagonalizer and slaves; the master process can run on a slave node if two UNIX activities on one node are allowed by the system. This is currently not possible on our IBM SP2 installation because of technical reasons. The master node is not counted in the speedup data given in a later section.

In the task definition we distinguish between static and dynamic tasks. The static tasks are rigidly associated with one of the nodes to reduce data communication. For good load balancing a certain percentage* of tasks is distributed dynamically to the nodes. The task definition and distribution is based on an estimation of the corresponding CPU load. For an optimal total load balancing it is important that the tasks are of different size and are sorted according to decreasing size so that the most time consuming ones are executed first. The dynamical task distribution has to be done by an additional process (master; see above), because this process has to wait for requests for tasks from the diagonalizer and the slaves and deal with them immediately.

The LA has been parallelized,[11] but one obtains only moderate speedup factors and the parallel LA runs only on a small subset of the nodes. The reason is that parallel LA has much more communication than the other steps, but it needs far less CPU time. The one-electron integrals are parallelized also; this is especially important for calculations using ECPs. The current version is slightly more efficient (and it now includes DFT and RI-J) than the one described previously,[9] mainly as a result of code optimizations of sequential tasks mentioned earlier.

*For ERI computation and processing all tasks are distributed dynamically in the first SCF iteration and a user defined fraction (default 0.3) is distributed dynamically in the subsequent iterations (see ref. 8).

QUADRATURE

The quadrature is parallelized by distributing grid points: each node evaluates the density $\rho(\vec{r})$, eq. (1), and performs the summation, eq. (2), for the partial grid assigned to the node. Because the density and Fock matrix were replicated anyway, this procedure causes negligible additional data transfer and message passing. The actual grid construction was also parallelized to achieve the best efficiency. The procedure implemented was chosen according to the structure of the molecular grid that is composed of atomic grids corresponding to radial and spherical quadratures.

We first define tasks (batches of grid points) that consist of identical types (same atom and number of points) of spherical shells of points. A task contains a few hundred grid points. In the first SCF iteration using a new grid, the tasks are distributed dynamically to the nodes. Then the actual grid points and their weights are constructed. After this the partial quadrature is carried out. In later SCF iterations each node works on the same grid points as in the first iteration (i.e., we have a static distribution). In the later SCF iterations only the quadrature has to be done, because coordinates and weights are known from the first iteration.

Because the number of grid points on a given sphere around an atom depends on the type of atom, the distance from the nucleus, and molecular symmetry, we necessarily get some variation in the number of grid points in a task. We order the tasks according to the number of grid points included. The large tasks are executed first and the small ones last; this minimizes message passing and guarantees good load balancing.

RI-J

For the construction of the approximated Coulomb term RI-J the three-center repulsion integrals have to be computed. The most ''expensive'' of these integrals are calculated and stored in memory before beginning the SCF iterations.

These expensive (in-core) integrals are distributed statically: they are divided into fixed packages, each node receiving one package. The definition of tasks is done by a statistics subroutine on the basis of a crude integral cost estimate. In spite of the approximate nature of this estimation, the law of statistics leads to good load balancing, because 100,000 and more integral batches are distributed among the nodes for large molecules.

All integrals that cannot be stored in memory have to be recomputed on the fly twice in each SCF iteration. These integral batches are divided into parallel tasks; each node gets 20 tasks. The tasks are distributed dynamically in the first SCF iteration, and each node recomputes its share of the tasks in the subsequent iterations.

On distributed memory parallel computers (e.g., IBM-SP2) the available memory increases with the number of nodes, so that a growing fraction of the integrals can be stored in memory. This implies an overproportional ''superlinear'' acceleration with an increasing number of nodes for RI-J.

The user specifies the amount of memory, RI-memory, assigned to storage of repulsion integrals on each of the nodes. These numbers are given later and in Table II.[†]

Technical Details

PARALLEL MACHINE

The IBM-SP2 (located at RZ-Karlsruhe) has a total of 256 RS6000 nodes, 168 nodes have 512-MB core memory, and it has a 120-MHz clock cycle. The nodes are connected with a fast switch.

TEST MOLECULES

It is convenient to use a shorthand notation for the test molecules used by specifying molecular symmetry, the number of basis functions N_{BF}, auxiliary basis functions N_{aux}, and the number of atoms. The calculations were done for the RIDFT structures. Extended Hückel MO were used as start vectors in all calculations. The gradient corrected BP-86 functional was used throughout for DFT. The specifications made below are valid unless stated otherwise. The grid specifications are described in previous publications[12,13]: m3 denotes a ''multigrid'' in which a coarse grid is employed during SCF iterations and the finer grid only in the evaluation of the final energy and the gradient; m4 is finer than m3.

[†]The ''diagonalizer node'' uses less memory for repulsion integrals, because some memory is needed for sequential only purposes.

SiAl

This is $SiAl_{14}[C_5(CH_3)_5]_6$; it has 165 atoms; symmetry D_{3h} (Fig. 1); and an SV(P) basis[29]: $N_{BF} = 1365$, $N_{aux} = 3585$. Grid m4[23] was used for the DFT. Thirteen SCF iterations were needed for convergence in the DSCF/RIDFT runs, and 200 MB was used for RI-J repulsion integrals on each of the nodes if not stated otherwise.

Cd10

The formula is $Cd_{10}Se_4(SeCH_3)_{12}[P(CH_3)_3]_4$; it has 126 atoms; symmetry T (Fig. 2)[30,31]; and an SV(P) basis: $N_{BF} = 1400$, $N_{aux} = 3386$. For Cd an ECP replacing 28 core electrons up to $3d$ was used, ''ecp-28-mwb.''[32] The grid was m3; it has 15 SCF iterations and 200 MB was used for RI-J.

Cd10Ph

This is $Cd_{10}Se_4(SeC_6H_5)_{12}[P(C_6H_5)_3]_4$; it has 294 atoms and symmetry T.[30,31] Cd10Ph has the same structure as Cd10 (Fig. 2), and the H atoms were replaced by phenyl groups. The ECP for Cd and basis for Cd, Se, and P are as in Cd10. The basis was SV for bridge C atoms of the phenyl groups. The SZ benzene basis[33] was for other atoms of the

TABLE II.
Dependence of CPU Times (min) on Symmetry.

	SiAl 16 Nodes			Cd32 32 Nodes			Cd10Ph 16 Nodes		
Symmetry	C_1	C_2	D_3	C_1	C_2	D_2	C_1	C_2	T
RI memory (MB)	200	300	300	200	200	200	200	200	200
RIDFT	71	32	13.2	221	97	42	234	97	12.5
RI-J	12	2.8	0.4	45	18	5.9	108	40	0.8
Quadrature	39	19.9	6.4	45	24	12.2	52	27	5.2
Linear algebra	14	6.8	1.8	72	37	15.4	40	20	3.5
Remainder	7	2.4	1.3	58	17	7.3	33	10	2.8
RDGRAD	24	11.9	4.3	39	19	10.0	65	29	6.0

The abbreviations are explained in the text.

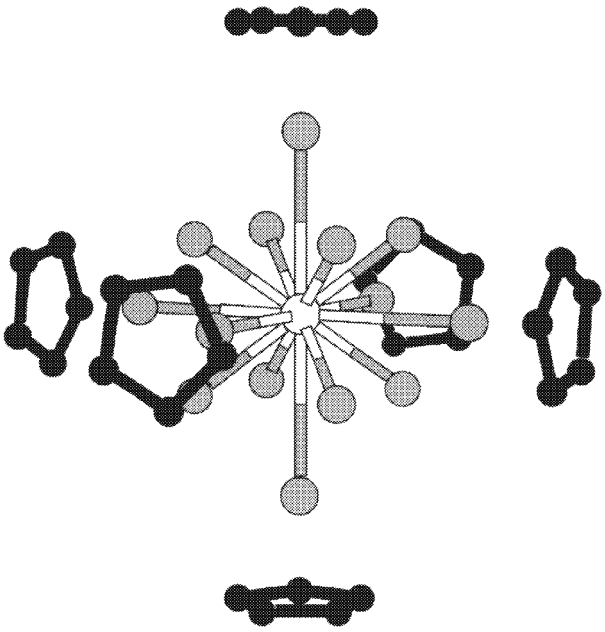

FIGURE 1. The $SiAl_{14}[C_5(CH_3)_5]_6$ structure: central white atom, Si; grey atoms, Al; black five membered rings; $[C_5(CH_3)_5]$ groups.

phenyl groups; $N_{BF} = 1952$, $N_{aux} = 7226$. The grid was m3; there were 18 SCF iterations, and 200 MB was used for RI-J.

Cd32

This is $Cd_{32}Se_{14}(SeH)_{36}(PH_3)_4$; it has 134 atoms; symmetry T (Fig. 3)[30,31]; and an SV(P) basis: $N_{BF} = 2754$, $N_{aux} = 6106$. The ECP ecp-28-mwb was used for Cd. The grid was m4; it had 16 SCF iterations; and 200 MB was used for RI-J.

Mg16

This is $Mg_{16}Cl_{32}$; it has 48 atoms; symmetry D_{8d} (Fig. 4)[34]; and an SVP basis: $N_{BF} = 768$, $N_{aux} = 2160$. It was grid m3; it had 13 SCF iterations; and 300 MB was used for RI-J.

Results and Discussion

SCALING BEHAVIOR OF IMPORTANT INDIVIDUAL STEPS

Timings for a few selected molecules will now be discussed in some detail, mainly to assess the

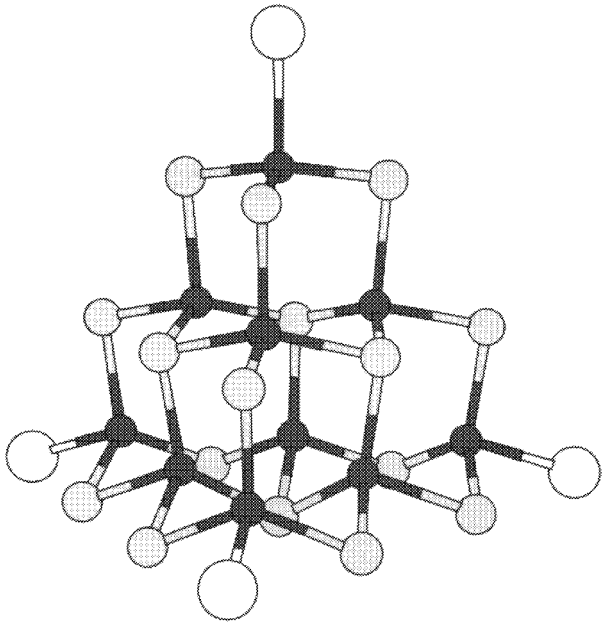

FIGURE 2. The $Cd_{10}Se_4(SeR)_{12}(PR_3)_4$ structure: dark grey atoms, Cd; light grey atoms, Se; large white balls, $P(R_3)_3$ groups; R = CH_3 or R = C_6H_5; R groups on two-coordinate Se atoms omitted.

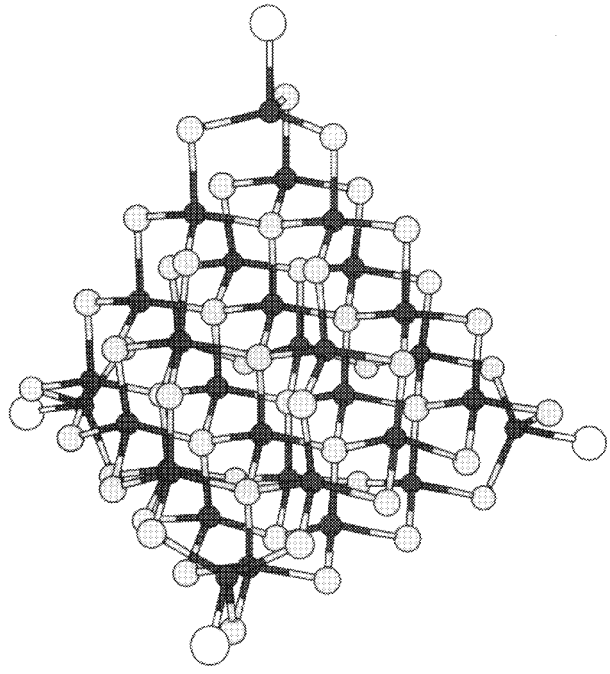

FIGURE 3. The $Cd_{32}Se_{14}(SeH)_{36}(PH_3)_4$ structure: dark grey atoms, Cd; light grey atoms, Se; large white balls, $P(H_3)_3$ groups; H atoms on two-coordinate Se atoms are omitted.

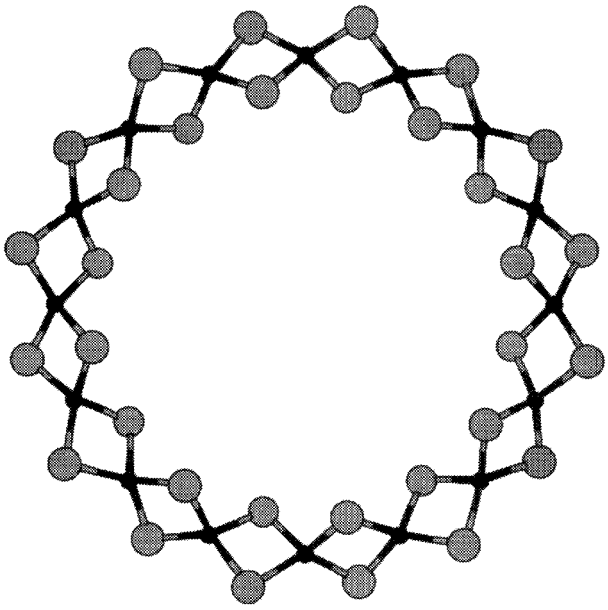

FIGURE 4. The $Mg_{16}Cl_{32}$ structure: black atoms, Mg; grey atoms, Cl.

parallelization efficiency for the tasks defined earlier. One encounters a problem here because cases typically treated on a parallel machine are too large to carry out a single processor run for comparison. In the present context this mostly matters for the ERI work in SCF or conventional DFT calculations. This is not serious, however, because the scaling behavior can be safely established by comparison with smaller cases and from runs with 16 and more nodes by extrapolation.

In Table III we present a detailed breakdown of timings and their dependence on the number of nodes for a calculation of SiAl in C_1 symmetry; the same data is plotted in Figure 5. The ERI work scales well up to 127 nodes; the speed up of about 103 implies a parallelization efficiency of better than 80%. The corresponding percentages are 95% for 32 nodes and 93% for 64 nodes.

The quadrature scales even slightly better than ERI with a speedup of 107.8 for 127 nodes and a parallelization efficiency of 96% (for 32 nodes), 92.5% for 64, and 85% for 127. For ERI and quadrature we thus achieved a satisfactory load balancing and sufficiently small overhead due to communication and data replication.

A markedly better picture results for the parallelization of RI-J where we achieve a pronounced superlinear speedup for a small number of nodes. The reason for this was already explained. The in-core mode is roughly 10 times faster than the direct mode; the more nodes used, the more memory becomes available with a corresponding reduction in necessary computational work. This is nicely reflected by the data presented in Table III. The speedup always exceeds the number of nodes and amounts to 114 for 32 nodes. Here about 80% of the three-center integrals can be kept in memory as compared to a virtually direct run on a single node. Because the CPU time for RI-J is only 3.3 min on 32 nodes, we did not even consider more nodes for this case.

In Table IV we consider a smaller case, Mg16. We used only up to eight nodes because LA and the ''remainder'' are dominating for eight nodes and more. The LA has a large contribution for this example even on a single node (see Table I) because Mg16 behaves like a linear molecule (see Fig. 4). In linear systems the dominating steps RI-J and quadrature are relatively fast due to the exploitation of near zeros in TURBOMOLE. The timings presented for Mg16 confirm the pattern found for the preceding case. The quadrature scales al-

TABLE III.
Scaling of CPU Times with Number of Nodes for SiAl.

Nodes	ERI		RI-J		Quadrature		LA		Rem	
1	—	—	444.8	(1.0)	631.8	(1.0)	46	(1.0)	8	(1.0)
4	—	—	92.4	(4.8)	153.8	(4.0)	20	(2.3)	7	(1.1)
8	—	—	37.9	(11.3)	77.3	(7.9)	14	(3.4)	7	(1.1)
16	307.8	(15.5)[a]	17.2	(25.9)	39.4	(15.6)	14	(3.4)	7	(1.1)
32	156.3	(30.5)	3.9	(114.0)	19.9	(30.8)	14	(3.4)	7	(1.1)
64	80.2	(59.5)	—	—	10.4	(59.2)	14	(3.4)	7	(1.1)
127	46.1	(103.5)	—	—	5.9	(107.8)	14	(3.4)	7	(1.1)

The abbreviations are explained in the text. CPU times are in minutes; the relative speedup is in parentheses.
[a]Estimated as 15.5.

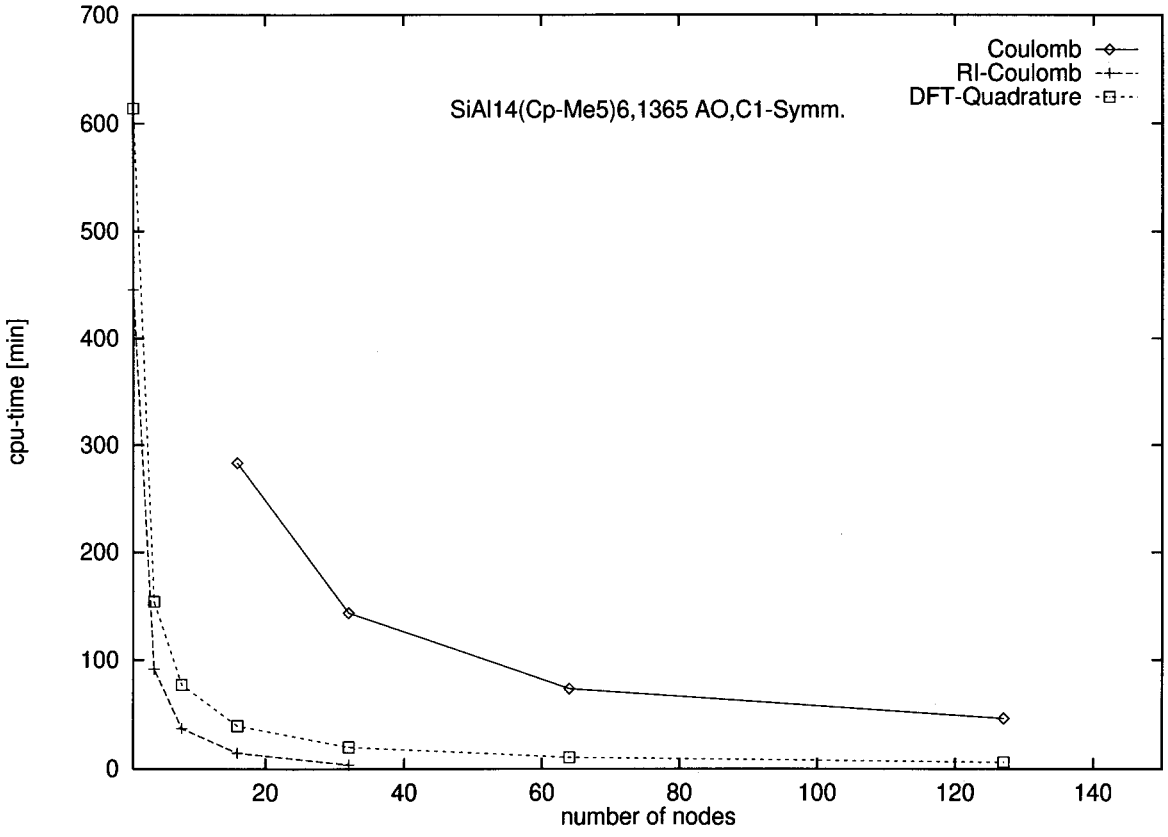

FIGURE 5. Scaling of dominant contributions to CPU times.

most linearly with the number of nodes: a speedup of (7.8) for eight nodes. For the RI-J step one can now for four or more nodes keep all $(\mu\nu|\alpha)$ in memory; this results in a speedup of 55.2 for eight nodes.

In Table V we report the results for a large molecule, Cd32, with 2754 basis functions and 6106 auxiliary basis functions on 134 atoms. Because of the reasonable limitations of the maximally allowed run time on the SP2 (8 h), we could only measure timings for four or more nodes in C_2 symmetry and total times for C_1; other symmetries will be given below. Although we are not on very solid ground, the following conclusions can be drawn. The quadrature still scales linearly with the number of nodes. Our implementation of RI-J shows a slight superlinear speedup, a factor of 10.6 from four to 32 nodes. The number of $(\mu\nu|\alpha)$ is now so large that even for 32 nodes only a small fraction of them can be kept in memory.

In Tables III–V we also report the timings for the LA package parallelized by Brakhagen and

TABLE IV.
Scaling of CPU Times with Number of Nodes for Mg16.

Nodes	RI-J		Quadrature		LA		Rem	
1	27.6	(1.0)	27.9	(1.0)	9.2	(1.0)	2.2	(1.0)
4	0.8	(34.5)	7.0	(4.0)	5.1	(2.0)	2.2	(1.0)
8	0.5	(55.2)	3.6	(7.8)	5.1	(2.0)	2.2	(1.0)

The abbreviations are explained in the text. CPU times are in minutes; relative speedup is in parentheses.

TABLE V.
Scaling of CPU Times with Number of Nodes for Cd32 in C_2 Symmetry.

Nodes	RI-J		Quadrature		LA		Rem	
1	1030[a]	(1.0)	697[a]	(1.0)	101[a]	(1.0)	26[a]	(1.0)
4	216	(4.8)	176	(4.0)	44	(2.3)	21	(1.2)
8	97	(10.6)	89	(7.8)	36	(2.8)	19	(1.4)
16	45	(22.9)	46	(15.2)	37	(2.7)	18	(1.4)
32	18	(57.2)	24	(29.0)	37	(2.7)	17	(1.5)

The abbreviations are explained in the text. CPU times are in minutes, relative speedup is in parentheses.
[a]Extrapolated values.

Lauwers[11]; the best measured speedup is 3.4 (Table III). The parallelization effort for the LA[11] had very limited success.

Of the tasks summarized as remainders, only the evaluation of one-electron nuclear attraction integrals was parallelized. This is a formal N^3 step with an asymptotic N^2 dependence. The parallelization appeared necessary, especially if ECPs are employed, which are more costly to evaluate than genuine three-center nuclear attraction integrals. It is straightforward to distribute these tasks over various nodes and the moderate speedup of the remainder is solely due to the parallelization of one-electron nuclear attraction integrals.

TOTAL PARALLELIZATION EFFICIENCY AND EFFECT OF SYMMETRY

In the preceding subsection we showed that the parallelization of the dominant computational steps of SCF, DFT, and RIDFT calculations (ERI, quadrature, and RI-J) was achieved in a satisfactory way. The measured speedup is almost proportional to the number of nodes used for ERI and quadrature and is superlinear for RI-J.

The total scaling of a complete run is limited by the remaining tasks: LA and remainder. This is of minor importance for the relatively CPU intensive conventional DFT treatments as can be seen from Table VI. The measured speedup (number of nodes) amounts to 30 (32), 52 (64), and 75 (127). From these results it is certainly justifiable to use 64 nodes for a conventional DFT run. The modest parallelization efficiency of LA and remainder limits the total efficiency, but this is not serious for up to 64 nodes.

A different picture emerges for RIDFT calculations in which ERI is replaced by the much more efficient RI-J, which in addition always shows superlinear speedup. The latter results in a total superlinear speed up for four nodes in some cases (e.g., Tables III, IV; Fig. 6), but LA and remainder limit the efficiency for more nodes in a more pronounced way then for DFT. For good parallelization efficiency one should never improve dominant steps (RI-J to handle the Coulomb term) if they are well parallelized!

One may look at our results from different angle: in all cases documented in Table VI we reduced the total time for a single point RIDFT calculation to 11 min for a small case (MgCl in C_1 on 16 nodes) and to less than 4 h for a large case (Cd32 with 2754 basis functions on 32 nodes in C_1).

TABLE VI.
Scaling of Total CPU Times (min); Calculations in C_1 Symmetry.

	Nodes				
	1	4	8	16	32
RIDFT					
SiAl	1131	272	135	71	43
Cd32[a]		458	242	146	97
Cd10	687	170	87	49	33
Mg16	66.9	15.1	11.4		
RDGRAD					
SiAl	331	86	45	24	14
Cd32[a]		102	55	32	19
Cd10	164	44	24	14	9
Mg16	21.2	6.3	3.8		
	8	16	32	64	127
DSCF					
SiAl		342	183	104	72
GRAD					
SiAl	109	56	29	16	9

The abbreviations are explained in the text.
[a]Calculated in C_2 symmetry.

Even more gratifying is the exploitation of symmetry documented in Table II. An increase in symmetry always leads to a reduction of CPU time by a factor that is larger than the order of the corresponding symmetry group $|G|$. This results from three facts: first of all TURBOMOLE exploits symmetry efficiently; the corresponding organizational overhead is very small (except for I_h). Next we take advantage of the fact that the diagonalization work is reduced roughly as $|G|^{-2}$. Finally, the number of three-center integrals $(\mu\nu|\alpha)$ to be considered varies proportional to $|G|^{-1}$; this makes it easier to store these integrals in memory.

Let us finally comment on timings for the evaluation of first-order gradients of electronic energies with respect to nuclear coordinates. As is obvious from Table VI, this always requires less effort then the computation of the energy. Although the parallelization efficiency is typically somewhat smaller for GRAD or RDGRAD than for DSCF/DFT or RDGRAD, the total times are only a fraction of the latter: 20–30%. Speedups for RDGRAD are very similar to those of RIDFT. In

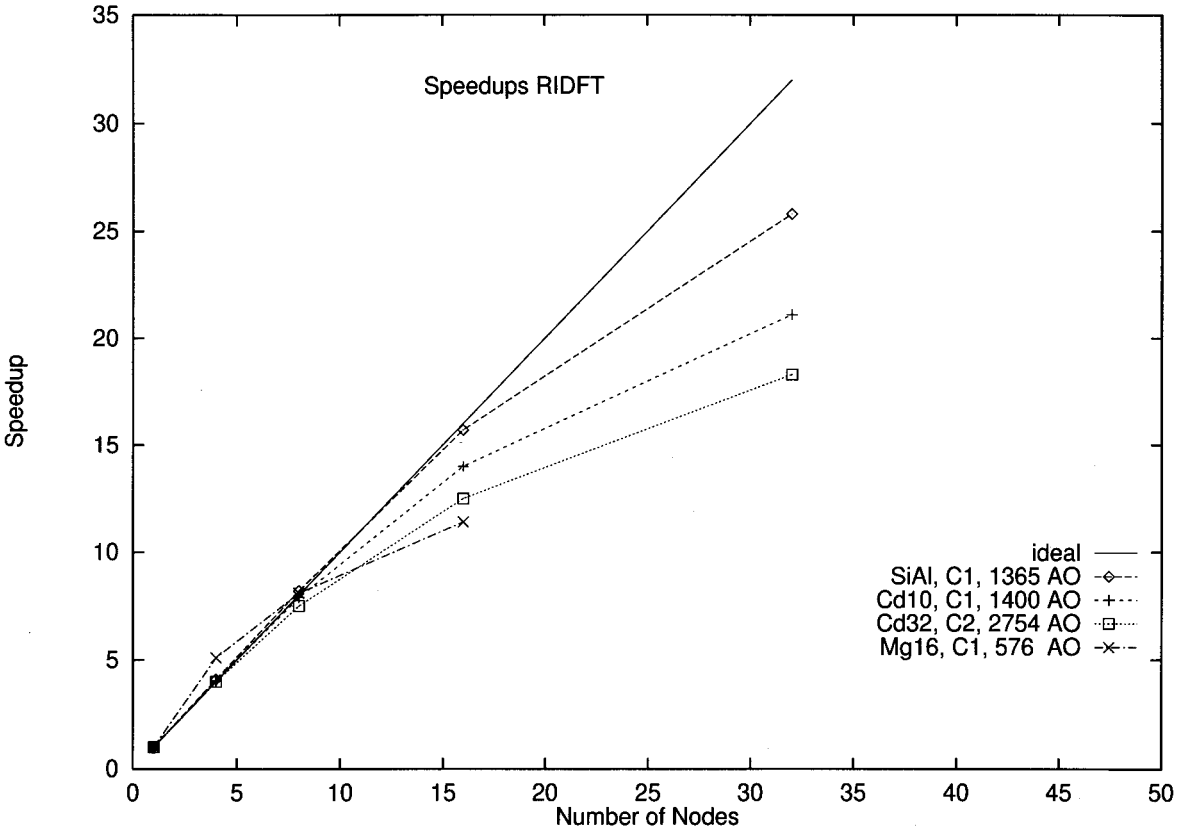

FIGURE 6. Speedup factors for RIDFT.

other words, the bottleneck is always the determination of energies and MO vectors.

Conclusions

We demonstrated an efficient parallelization of the dominant tasks of SCF or DFT calculations for 64 or more nodes. As a consequence, small cases (up to 600 basis functions) can be treated in a few minutes, molecules with up to 1500 basis functions and/or 150 atoms in less than 1 h, and large cases with up to about 3000 basis functions and/or 300 atoms in a few hours per geometry optimization cycle (energy plus gradient evaluation). For even larger cases the present approach encounters problems with LA, especially the diagonalization, which is not well parallelized. Linear scaling methods[24, 25] to optimize the energy directly with respect to the density matrix offer an attractive alternative in this respect.

The efficient RI-J approximation scales well on machines with up to 32 nodes that will become accessible in the near future for scientific and industrial users.

Although we only carried out extensive tests on an IBM SP2, similar results were obtained on HP-Convex and SGI parallel machines.

Acknowledgments

We thank the IBM, Silicon Graphics, and HP-Convex companies for their very helpful collaboration and for giving us access to parallel hardware.

References

1. A. T. Wong and R. J. Harrison, *J. Comput. Chem.*, **16,** 1291 (1995).
2. T. R. Furlani and H. F. King, *J. Comput. Chem.*, **16,** 91 (1995).
3. T. R. Furlani and H. F. King, In *Proceedings of Quantum Mechanical Simulation Methods for Studying Biological Systems*, Les Houches, France, May 2–7, 1995.
4. R. J. Harrison, M. F. Guest, R. A. Kendall, D. E. Bernholdt, A. T. Wong, M. Stave, J. L. Anchell, A. C. Hess, R. J.

Littlefield, G. L. Fann, J. L. Nieplocha, G. S. Thomas, D. Elwood, J. L. Tilson, R. L. Shepard, A. F. Wagner, I. T. Foster, E. Lusk, and R. Stevens, *J. Comput. Chem.*, **17,** 124 (1996).

5. I. T. Foster, J. L. Tilson, A. F. Wagner, R. L. Shepard, R. J. Harrison, R. A. Kendall, and R. J. Littlefield, *J. Comput. Chem.*, **17,** 109 (1996).
6. C. Fonseca Guerra, O. Visser, J. G. Snijders, G. te Velde, and E. J. Baerends, In *Methods and Techniques in Computational Chemistry* (*Metecc*-95), E. Clementi and G. Corongiu, Eds., Université L. Pasteur, Strasbourg, France, 1995.
7. Y. S. Li, M. C. Wrinn, J. M. Newsam, and M. P. Sears, *J. Comput. Chem.*, **16,** 226 (1995).
8. S. Brode, H. Horn, M. Ehrig, D. Moldrup, J. E. Rice, and R. Ahlrichs, *J. Comput. Chem.*, **14,** 1142 (1993).
9. R. Ahlrichs and M. v. Arnim, In *Methods and Techniques in Computational Chemistry* (*Meteccc*-95), Eds., E. Clementi and G. Corongiu, Eds., Université L. Pasteur, Strasbourg, France, 1995.
10. M. Häser and R. Ahlrichs, *J. Comput. Chem.*, **10,** 104 (1989).
11. F. Brakhagen and P. G. Lauwers, In *Methods and Techniques in Computational Chemistry* (*Metecc*-95), E. Clementi and G. Corongiu, Eds., Université L. Pasteur, Strasbourg, France, 1995.
12. O. Treutler and R. Ahlrichs, *J. Chem. Phys.*, **102,** 346 (1995).
13. A. D. Becke, *Phys. Rev. B*, **38,** 3098 (1988).
14. J. P. Perdew, *Phys. Rev. B*, **33,** 8822 (1986).
15. S. H. Vosko, L. Wilk, and M. Nussair, *Can. J. Phys.*, **58,** 1200 (1980).
16. C. Lee, W. Yang, and R. G. Parr, *Phys. Rev. B*, **37,** 785 (1988).
17. A. D. Becke, *J. Chem. Phys.*, **98,** 5648 (1993).
18. C. A. White, B. G. Johnson, P. M. W. Gill, and M. Head–Gordon, *Chem. Phys. Lett.*, **253,** 268 (1996).
19. M. C. Strain, G. E. Scuseria, and M. J. Frisch, *Science*, **271,** 51 (1996).
20. P. M. W. Gill and R. D. Adamson, *Chem. Phys. Lett.*, **261,** 105 (1996).
21. B. I. Dunlap and N. Rösch, *J. Chim. Phys.*, **86,** 671 (1989).
22. K. Eichkorn, O. Treutler, H. Öhm, M. Häser, and R. Ahlrichs, *Chem. Phys. Lett.*, **240,** 283 (1995).
23. K. Eichkorn, F. Weigend, O. Treutler, and R. Ahlrichs, *Theor. Chem. Acc.*, **97,** 119 (1997).
24. J. M. Millam and G. E. Scuseria, *J. Chem. Phys.*, **106,** 5569 (1997).
25. R. Baer and M. Head–Gordon, *Phys. Rev. Lett.*, **79,** 3962 (1997).
26. J. Breidung and W. Thiel, *Chem. Phys. Lett.*, **153,** 76 (1988).
27. A. D. Becke, *J. Chem. Phys.*, **88,** 2547 (1988).
28. R. Ahlrichs, unpublished results.
29. A. Schäfer, H. Horn, and R. Ahlrichs, *J. Chem. Phys.*, **97,** 2571 (1992).
30. K. Eichkorn and R. Ahlrichs, *Chem. Phys. Lett.*, **288**, 235 (1998).
31. S. Behrens, M. Bettenhausen, A. C. Deveson, A. Eichhöfer, D. Fenske, A. Lohde, and U. Woggon, *Angew. Chem.*, **108,** 2360 (1996).
32. D. Andrae, U. Häussermann, M. Dolg, H. Stoll, and H. Preuss, *Theor. Chim. Acta*, **77,** 123 (1990).
33. All basis sets mentioned in this article are contained in the TURBOMOLE basis set catalogue that is available via ftp, internet address tchibm16.chemie.uni-karhsruhe.de (129.13.106.9), user name ''anonymous,'' directory/pub/ri-j.
34. K. Eichkorn, U. Schneider, and R. Ahlrichs, *J. Chem. Phys.*, **102,** 7557 (1995).