PARALLEL FP-LAPW FOR DISTRIBUTED-MEMORY MACHINES

To meet the high computing-power and memory requirements of applying the full-potential linearized augmented plane wave method to complex problems, the authors have parallelized the FP-LAPW code WIEN97 for distributed-memory machines. They then implemented the parallel code on a Cray T3E.

Today's problems in surface diffusion, crystal growth, and heterogeneous catalysis involve large numbers of atoms and electrons. If you are considering 3D transition metals or first-row elements, or phenomena involving core electrons or electronic wave function near the nucleus, you will need an all-electron approach and "chemically best" basis functions. In this case, the *full-potential linearized augmented plane wave* method[1] is appropriate.

The development of the formalism and techniques for FP-LAPW appears in several references, including those by Henri J.F. Jansen and Arthur J. Freeman[2] and by Peter Blaha, Karlheinz Schwarz, and Peter Herzig.[3] In recent years, FP-LAPW has experienced significant progress. For example, researchers routinely calculate magnetism and nuclear quantities (for example, isomer shifts, hyperfine fields, electric field gradients, and core level shifts).[4–7] Also, forces and molecular dynamics have been implemented,[8] and recent optimizations have reduced the CPU time of FP-LAPW calculations considerably.[9] Nevertheless, because the computational expense and memory requirements are still quite high, FP-LAPW implementations are applicable only to moderately complex systems.

The best way to overcome this drawback is to parallelize the numerical algorithm for execution on high-performance parallel computer architectures. We have parallelized WIEN97,[10] a sequential FP-LAPW program for message-passing systems, and implemented the parallel program on a Cray T3E. Performance measurements prove the T3E version's efficiency.

1521-9615/01/$10.00 © 2001 IEEE

RENATE DOHMEN
Rechenzentrum Garching der Max-Planck-Gesellschaft
JAKOB PICHLMEIER
Silicon Graphics GmbH
MAX PETERSEN, FRANK WAGNER, AND MATTHIAS SCHEFFLER
Fritz-Haber-Institut der Max-Planck-Gesellschaft

How FP-LAPW works

Basically, FP-LAPW divides the system's volume into two regions and adopts different definitions of the basis functions in these regions. First, as Figure 1a shows, the method places *muffin tin spheres* around the atomic positions. The remaining area is the *interstitial region*. The basis functions for expanding the electronic wave functions are

$$\Phi_{\mathbf{G}}^{\mathrm{LAPW}}(\mathbf{k},\mathbf{r}) = \begin{cases} e^{i(\mathbf{G}-\mathbf{k})\cdot\mathbf{r}} & : \mathbf{r} \in \mathrm{I} \\ \sum_{lm} R_{lm}(r,\epsilon_l) Y_{lm}(\hat{r}) & : \mathbf{r} \in \mathrm{MT} \end{cases} \quad (1)$$

with

$$R_{lm}(r,\epsilon_l) = a_{\mathbf{G}lm}(\mathbf{k})u_l(r,\epsilon_l) + b_{\mathbf{G}lm}(\mathbf{k})\frac{\partial u_l(r,\epsilon)}{\partial \epsilon}\Big|_{\epsilon=\epsilon_l}. \quad (2)$$

Constructing the basis function inside the MT spheres requires the radial functions $R_{lm}(r)$. To obtain these functions, LAPW employs a linear combination of

- the solutions $u_l(r, \epsilon_l)$ of the radial Schrödinger equation for the spherical component $V_{00}(r)$ of the potential, at a reference energy ϵ_l, and
- each solution's energy derivative, $\frac{\partial u_l(r,\epsilon)}{\partial \epsilon}\Big|_{\epsilon=\epsilon_l}$.

The continuity in value and slope of the basis function at the MT boundary determines the coefficients $a_{\mathbf{G}lm}$ and $b_{\mathbf{G}lm}$. Figure 1b shows such an LAPW basis function.

In the interstitial region, the basis functions are plane waves with reciprocal lattice vector **G** and point **k** of the first Brillouin zone. The expansion in **G** must be truncated; an input parameter controls the number of considered plane waves.

In the two regions, MT and I, the potential and electron density are represented in a different way. In this context, the MT concept is employed only to construct high-quality basis functions, not to approximate the Kohn-Sham potential or the electron density. The potential is written as

$$V(\mathbf{r}) = \begin{cases} V_{\mathbf{G}}e^{i\mathbf{G}\cdot\mathbf{r}} & : \mathbf{r} \in \mathrm{I} \\ \sum_{lm} V_{lm}(r)Y_{lm}(\hat{r}) & : \mathbf{r} \in \mathrm{MT} \end{cases}. \quad (3)$$

Figure 1c illustrates such a potential. We express the electron density similarly. Here, **G** denotes the Fourier components that must be truncated at a certain cutoff value. This truncation occurs at a value that is twice as high as the one in Equation 1. The $V_{lm}(r)$ are the potential's expansion coefficients in terms of the spherical harmonics, $Y_{lm}(\hat{r})$. Also the expansion in l must be truncated. We represent the $V_{lm}(r)$ numerically on a radial mesh.

The basis functions inside the MT spheres are nonorthogonal, which implies that we must solve a generalized, but linear, eigenvalue problem:

$$(H - \epsilon_i S)|c_i\rangle = 0 \quad (4)$$

where H is the Hamiltonian matrix,

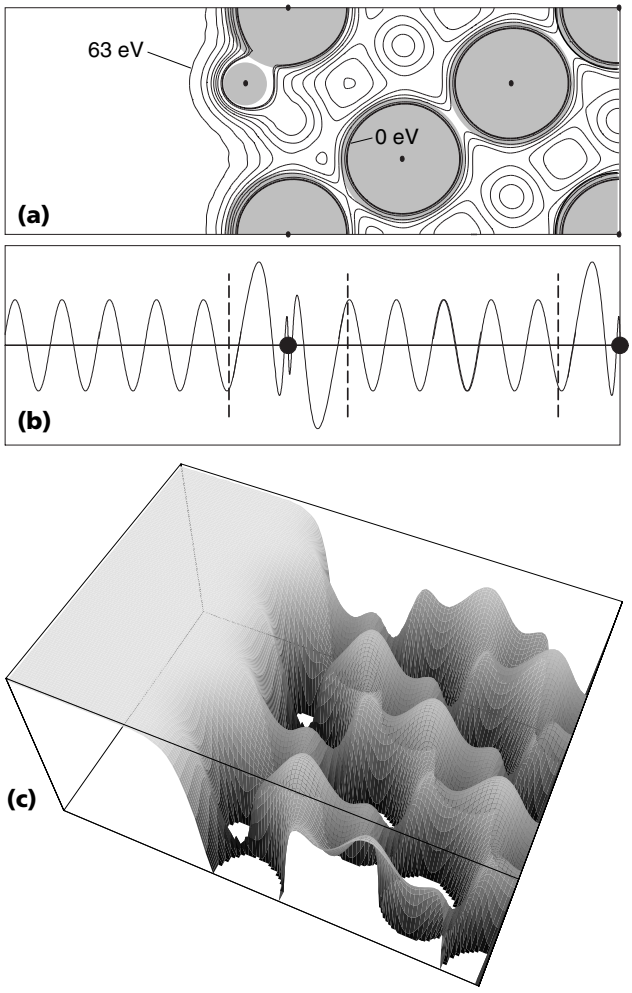


Figure 1. (a) A contour plot of the effective Kohn-Sham potential for H:Pt (111). The contour spacing is 9 eV. The gray regions (*muffin tin spheres*) indicate the unit cell's partitioning; the white area is the *interstitial region*. (b) The LAPW basis function: Inside the MT spheres, LAPW employs solutions of an atomic-like potential that are matched at the MT boundaries (the dashed lines) to plane waves (compare with Equation 1). In Figures 1a and 1b, full circles indicate the atomic positions. (c) The effective Kohn-Sham potential for H:Pt (111), calculated by the FP-LAPW method as a height profile.

The Cray T3E

Each processing element of the T3E we used has a 300-MHz DEC Alpha 21164 processor and 128 Mbytes of main memory. The PEs form a 3D torus and are connected through a fast low-latency communication network whose peak performance is 600 Mbytes per second. Through this network, the processors can synchronize and exchange messages and—what is unusual for distributed-memory systems—directly access the memories of remote PEs without any agency of the remote PEs. The T3E's hardware explicitly supports this *one-sided communication*.

Cray offers three communication libraries: the native Cray Shmem library and the Parallel Virtual Machine and Message Passing Interface portable libraries. Shmem performs best and fully supports asynchronous one-sided communication. Furthermore, several parallel numerical libraries are available for the T3E—particularly Cray's general-purpose parallel scientific library Scilib, and the ScaLapack library for linear algebra problems, including eigenvalue problems.

$$H = \frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r})$$

and S is the overlap matrix.

We can solve the eigenvalue problem numerically in two ways. The first consists of a full diagonalization of the matrix and delivers an exact solution. The second exploits the requirement that the electron density be calculated self-consistently; it uses a previous calculation's eigenvectors to determine an approximate solution of the generalized eigenvalue problem. Several researchers have exploited this method.[11–13]

In the code underlying our parallel implementation, we have chosen a blocked Davidson method.[14] We want to find a vector increment

$$\left|\delta A_j\right\rangle$$

such that a new eigenvector

$$\left|c_j^{i+1}\right\rangle$$

can be expressed by the old one,

$$\left|c_j^i\right\rangle, \left|c_j^{i+1}\right\rangle = \left|c_j^i\right\rangle + \left|\delta A_j\right\rangle . \quad (5)$$

Combining Equations 4 and 5 and retaining only the diagonal part of the Hamiltonian and overlap matrices, we reach the approximated expression

$$\left|\delta A_j\right\rangle = \sum_k \frac{\left\langle c_k^i \middle| H - \epsilon_j S \middle| c_k^i \right\rangle}{H_{kk} - \epsilon_j S_{kk}} \left| e_k \right\rangle \quad (6)$$

for the vector increment. We can then use the set of new and old eigenvectors as a unitary transformation to reduce the size of the problem from the order of the number of plane waves to the order of twice the number of eigenvalues to be calculated. This reduces the number of numerical calculations by a factor of five to 10.

WIEN97

This sequential program consists of three subprograms:

- LAPW0 generates the potential from a given charge density.
- LAPW1 builds H and S within the LAPW basis and computes the eigenvalues and eigenvectors of

  $$\left(H - \epsilon_i S\right)\left|c_i\right\rangle = 0$$

  with H and S being symmetric.
- LAPW2 computes the valence charge density from the eigenvectors and occupation numbers.

The three subprograms execute iteratively until they reach a self-consistent state. The most time-consuming part is LAPW1. To solve the eigenvalue problem, the code provides a *direct solver* based on full diagonalization (`FDIAG`) and an *iterative solver* based on the Davidson algorithm (`DAVID`). The code implements both methods through numerical library routines. The ratio of the execution times of LAPW0, LAPW1, and LAPW2 is 1:150:10; LAPW1's time is distributed 40:60 between setup and solution of the eigenvalue problem. The sequential code runs on several computer systems, including the IBM RS6000, SGI Power Challenge, and NEC SX4 and SX5.

Starting from the sequential IBM version, we've parallelized all three subprograms for distributed-memory machines. However, we've emphasized LAPW1 as the dominating part of the code, both during the parallelization and in this article. The Cray T3E, the first target machine for our parallel version, seemed particularly suitable for our purposes (see the sidebar for more on the Cray T3E).

Parallelizing WIEN97

We wrote our parallel WIEN97 implementation in Fortran 90; we based the communication on Cray's Shmem library for performance reasons. Furthermore, we use the ScaLapack and Scilib libraries. (For more on libraries for the Cray, see the sidebar.) Our message-passing ver-

sion is in principle not special to the Cray T3E. However, we did optimize the first implementation somewhat for the T3E. The implementation is, of course, affected by the T3E's architectural features—for example, the computer's fast communication and the limited memory per processing element (PE). Furthermore, it uses Cray's native libraries, which are not necessarily available on other parallel machines.

We parallelized WIEN97 on two levels. First, because LAPW1 must execute repeatedly for the different **k**-points, we could perform a very coarse-grained parallelization at script level. The number of **k**-points is typically between one and 20, depending on the physical problem, and all **k**-points are independent. However, because the number of **k**-points is not very large, implementing WIEN97 for a massively parallel system with several hundred processors will require additional parallelization of source code.

To implement such fine-grained parallelism, we used the message-passing paradigm; to implement the parallelization related to the **k**-points, we used a task group concept. So, for LAPW1, we split the processors into moderate-size groups, each group treating its share of the **k**-points with a message-passing version of LAPW1. For LAPW0 and LAPW2, all processors form one group. While the parallelization with respect to the **k**-points is trivial, the source-code parallelization is not that obvious, as we show in the next section. We present performance measurements for *rh2x2.31*, a small case with a matrix size of 3,000, and *cu4x2.9L14*, a large case with a matrix size of 7,000.

LAPW1

As we mentioned before, LAPW1 has two parts: the setup of H and S and the solution of

$$(H - \epsilon_i S)|c_i\rangle = 0 .$$

H and S are dense matrices with a size of approximately 10,000; the task is typically to calculate the lowest 10 percent of the eigenvalues and corresponding eigenvectors.

Parallel solvers for the generalized eigenvalue problem. Both the direct and the iterative eigenvalue problem solvers of the sequential IBM code employ Eispack, Lapack, and ESSL (Engineering and Scientific Subroutine Library) library routines as building blocks. Our parallelization strategy was to replace these library calls by corresponding routines from the ScaLapack library as much as possible. This strategy let us retain large parts of the code and especially the program's structure.

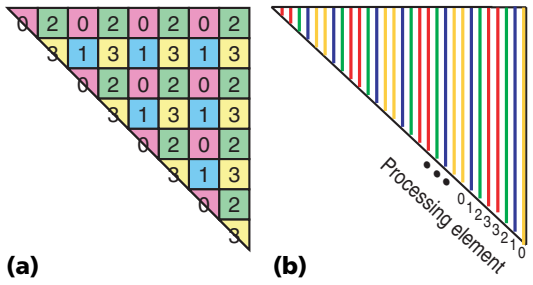


Figure 2. Parallel data structures of the symmetric matrices *H* and *S* for four processing elements: (a) block-cyclic distribution for the eigenvalue solvers; (b) column-oriented distribution for matrix setup.

ScaLapack uses Lapack for local computations and the Basic Linear Algebra Communication Subprograms (Blacs) for communication between processors. Lapack is optimized to efficiently use the hierarchical memory structure (cache) of RISC microprocessors. The Blacs implementation on the Cray T3E directly accesses the machine's fast communication hardware.

Using ScaLapack restricts data distribution. Matrices require a 2D block-cyclic distribution. Figure 2a shows this for H and S. For symmetric (and thus quasitriangular) matrices, the block-cyclic distribution has two slight disadvantages.

First, because ScaLapack does not support a packed storage format, memory space for the complete 2D array must be allocated—distributed, of course, among the PEs. (However, allocating space for only one-half of the array to store all the data of the matrices, as the sequential program does, would suffice.) Consequently, the minimum number of processors necessary to solve the eigenvalue problem increases owing to memory limitations. This increase, in turn, might lower the parallel program's efficiency.

Second, the block-cyclic distribution applied to triangular matrices usually results in a slight load imbalance. However, we can control this somewhat by choosing appropriate block sizes. A block size of 32 is the default. It gives both good cache utilization and reasonable load balance and keeps within limits the communication overhead due to boundary exchanges.

The ScaLapack routine `PSSYGVX` corresponds to the full diagonalization method. It is suited for solving the given eigenvalve problem. For example, it lets you choose the number of eigenvalues to calculate. The working storage for this rou-

Table 1. The performance of two parallel eigenvalue problem solvers on the basis of ScaLapack routines corresponding to full diagonalization and the Davidson algorithm, for matrix sizes 2,500 and 7,000. The number of determined eigenvalues was 250 and 700, respectively.

	Full diagonalization				Davidson algorithm (two iterations)			
	$n = 2{,}500$		$n = 7{,}000$		$n = 2{,}500$		$n = 7{,}000$	
No. of Processors	Time (seconds)	Mflops/PE	Time (seconds)	Mflops/PE	Time (seconds)	Mflops/PE	Time (seconds)	Mflops/PE
1	—	—	—	—	113	216	—	—
2	—	—	—	—	51	245	—	—
4	110	128	—	—	28	226	—	—
8	70	108	—	—	17	200	—	—
16	39	97	—	—	11	161	—	—
32	30	65	377	111	8	117	85	212
64	20	52	206	103	6	83	51	182
128	19	35	163	70	5	57	35	140
256	15	23			4	34	23	105

tine is higher than that for the sequential routine.

The solver performs quite well. Table 1 shows the solver's execution times and performance for quadratic matrices of sizes 2,500 and 7,000. The larger matrix requires at least 32 processors to solve the eigenvalue problem. However, the good scaling behavior from four to 16 PEs for the smaller problem moves to higher processor numbers for the larger problem. So, we can use 128 PEs efficiently for the larger matrix.

Parallelization of the iterative Davidson algorithm also requires a 2D block-cyclic distribution of the matrices, but now we use a whole series of ScaLapack routines:

- `PSSYMM` and `PSGEMM` perform the residual calculation and the projection into the reduced space.
- `PSPOTRF` performs Cholesky factorization (in the reduced space).
- `PSSYGST` reduces the generalized eigenvalue problem to standard form.
- `PSSYEVX` solves the standard symmetric eigenvalue problem.
- `PSTRSM` carries out the backsubstitution of the eigenvectors.

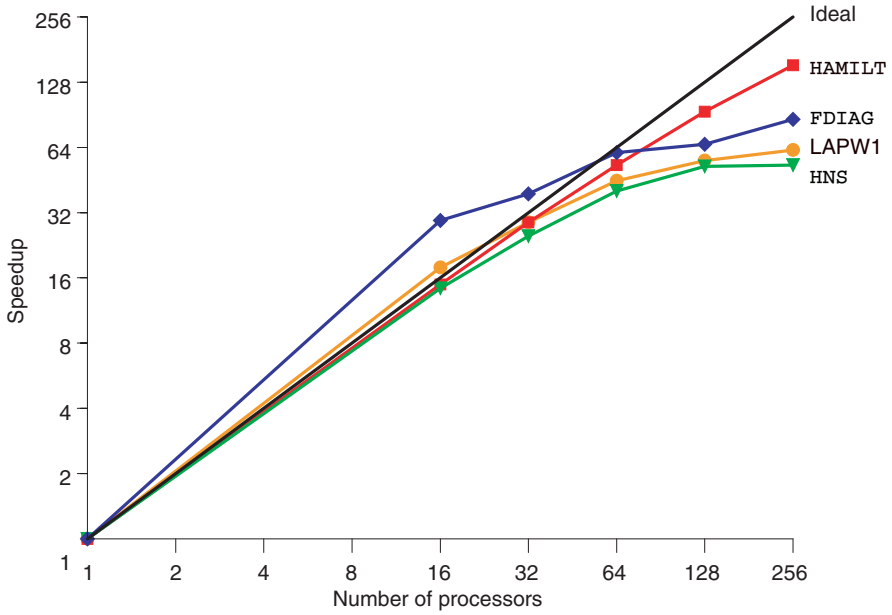


Figure 3. Parallel speedup for the different parts of LAPW1 for the small data set rh2x2.31 with matrix size 3,000 (no parallelization for k-points).

The performance measurements in Table 1 show that this parallel solver scales similarly to the direct solver, while its absolute performance is higher. As in the sequential case, the parallel version of the Davidson algorithm is approximately four times faster than the eigenvalue computation with full diagonalization.

Parallel setup of H and S. Because the corresponding sequential algorithm treats the matrices columnwise, we choose a column-oriented parallel data structure and algorithm instead of the block-cyclic distribution used in the solvers. To achieve a good load balance, we distribute columns to the *np* processors according to a special scheme (see Figure 2b). Back to front, we assign the columns of the matrices one after the other to processor 0, 1, …, $np - 1$, $np - 1$, $np - 2$, …, 0, and repeat this until no columns are left. The effect is that the lengths of the first two columns on all processors sum to the same value, the next two again, and so on. Any remaining columns are small, as is the load imbalance.

To build the columns, we use the `HAMILT` and `HNS` subroutines, which correspond to the spherical and the nonspherical parts of the Hamiltonian matrix. In `HAMILT` the columns can be treated nearly independently. `HAMILT` requires only a small amount of communication arising from the fact that memory constraints necessitate distributing the large arrays that are needed in full for all columns. So, the subprogram scales well, as its speedup curve in Figure 3 indicates.

The `HNS` subroutine updates *H* only; this requires several calculations of the kind depicted in Figure 4. The subroutine updates a specific column *i* with the *i*th element of vector **b** and the elements 1:*i* of vector **a**. Owing to the distribution scheme of *H* in the parallel version, each PE needs special elements of vector **b**, but all elements of vector **a**, to calculate its part of the columns. The arrays **a** and **b**—and several analogous pairs of vectors—are calculated in parallel and must be collected before the parallel update of *H* as we described previously.

On the Cray T3E, the most efficient way is to treat all 12 vectors in a blockwise fashion and collect them with a highly efficient Shmem routine. This is primarily because memory constraints require distributing the data necessary for calculating the vectors. On architectures where memory space is no problem but where communication is the bottleneck, we might instead calculate the special elements of **b** and the total vector **a** on the different PEs to avoid communication. The speedup curves for the different parts of LAPW1 in Figure 3 show that data collection—which must be performed often—limits the speedup for `HNS`, especially for higher processor numbers. However, `HAMILT` and `FDIAG` scale reasonably well.

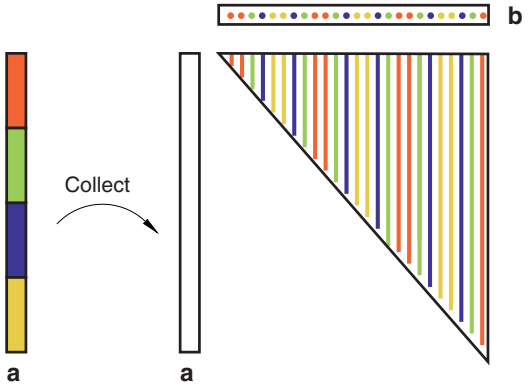


Figure 4. A schematical presentation of the calculations in subroutine `HNS`.

Performance. We parallelized the LAPW1 parts (matrix setup and eigenvalue problem solution) with quite different means and even different data distribution schemes for the main variables, to get optimal performance for each part. The communication step necessary to copy the matrices from one data format into the other takes negligible time and does not noticeably affect the parallel program's performance. Owing to dynamic allocation and reuse of memory space, using two different data representations for the matrices leads to additional memory requirements only for the short moment of copying the data.

This parallelization strategy has led to an efficient parallel message-passing version of LAPW1. The measurements in Figure 3 show that the strategy achieves satisfactory performance and speedup even for a relatively small data set with matrices of size 3,000. This is particularly true in the moderately parallel range of up to 32 or 64 processors, as needed for the task group concept we mentioned earlier. The performance is even better for larger problems, as we show later.

LAPW0

This subprogram computes the total potential as the sum of the Coulomb and the exchange correlation potential from the total electron density. According to the APW method, LAPW0 divides the space into the MT spheres and interstitial region. A multipolar Fourier expansion

calculates the Coulomb potential. The subprogram computes the exchange correlation potential numerically on a grid. Inside the atomic spheres, LAPW0 employs a least-squares procedure to reproduce the potential, using a lattice harmonics representation. In the interstitial region, the subprogram uses a 3D fast Fourier transform.

The parallel version partitions the plane waves, and all calculations in the plane wave space, blockwise over the processors. This version requires only a small amount of communication (global sums). It distributes the FFT grid over the last dimension and uses the Cray Scilib routine `PCCFFT3D` for the distributed parallel 3D FFT. The FFT and the transition between coordinate and Fourier space requires intense global communication (gather, scatter), which LAPW0 performs using Shmem communication routines. The subprogram distributes the calculations in the spheres over the radial mesh, even though the mesh data must be replicated on all the PEs. Again, this requires global gather and scatter operations.

The parallel program scales well up to high processor numbers.

LAPW0 performs all I/O operations through PE0. Unformatted I/O uses the asynchronous, buffered I/O layer from Cray's Flexible-File-IO package, which reduces the required I/O time significantly. The parallel program scales well up to high processor numbers. To port the T3E implementation to other parallel machines requires a substitution for the parallel 3D FFT routine.

LAPW2

To parallelize LAPW2, we had to solve two main problems. First, we had to redesign the I/O concept. Second, computing the charge density inside the MT spheres and in the interstitial region requires a sequence of linear algebra calculations, including actions on triangular matrices. So, parallelization included finding a parallel data format that holds for the whole sequence of linear algebra tasks yet nevertheless yields good load balancing and scaling.

Redesigning I/O. LAPW2 processes the large eigenvector matrices that LAPW1 produces for the different **k**-points. But the amount of data is usually too large to fit into a single processor's memory. The sequential version settles this problem by reading the vector file anew for each atom and extracting the data referring to that atom. In the test case rh2x2.31, with 40 atoms and a matrix size of 3,000, the time for reading the vector file totals approximately 400 seconds, compared to a total execution time of 3,600 s. This might be tolerable for a sequential code, but compared to a much smaller parallel execution time of, say, 30 to 40 s with approximately 100 PEs, it is too much.

So, we redesigned the I/O concept by exploiting the scalable memory that comes with parallel processing. The subprogram reads the vector file only once and distributes the data among the processors. For this purpose, we've augmented the corresponding array-valued variables with a further dimension counting the atoms, while partitioning these values with respect to one of the former dimensions. These measures reduce the I/O time roughly by a factor that is equal to the number of atoms. For rh2x2.31, for example, reading and distributing data take approximately 10 to 15 s.

Computing the valence charge density. The subroutines `L2MAIN` and `FOURIR` compute the valence charge density inside the MT spheres and in the interstitial region, respectively. The bases for both subroutines are the large eigenvector matrices for the different **k**-points. Matrix–matrix products are the core of the computations in both subroutines. The sequential version performs these products by first cutting the matrices into blocks of a certain size along the larger dimension. It then uses the fast BLAS3 library routine `SGEMM` to perform a sequence of matrix–matrix operations of the form $X = X + Y^* Z$ on the blocks.

In the parallel version, each subroutine employs a different kind of matrix–matrix product, and the eigenvector matrices must be distributed skillfully to obtain good load balancing in both subroutines. The eigenvector matrices have size $Nmat \times Nume$. *Nmat* and *Nume* are the number and length of the calculated eigenvectors and depend on the test case and the respective **k**-point. Because *Nmat* typically is approximately 10 times larger than *Nume*, partitioning the eigenvector matrix with respect to this dimension is advisable. To provide optimal overall load balancing, we first partition the eigenvector matrices into $2 \cdot np$ blocks, with *np* being the number of used processors. We then assign the blocks one by one to the processors 0, 1, …, $np - 1$, $np - 1$, $np - 2$, …, 0, as Figure 5 shows for the eigenvector matrix *A*.

L2MAIN multiplies the eigenvector matrix with other matrices that are distributed in the same way. By using SGEMM, each processor computes with its share of blocks a partial sum of the complete result matrix (see Figure 5a). The only difference from the sequential version is that the block size is not a constant but depends on the number of used processors. Finally, L2MAIN sums the contributions of all processors fully in parallel. To do this, it first partitions the result matrix blockwise. Each processor then fetches the blocks needed for its part of the result matrix from the other PEs and sums them up. This block partitioning is an appropriate distribution scheme for the subsequent calculations of L2MAIN; no further communication is necessary at this point.

By symmetry arguments, we can reduce the matrix–matrix products in FOURIR to calculating the upper triangular matrix and copying each element of that matrix to the corresponding element in the lower triangular matrix. We multiply eigenvector matrix A of size $Nmat \times Nume$ and matrix A_h of size $Nume \times Nmat$, which is the transposition of A multiplied by a weight function w.

In the sequential version, the matrix–matrix product for the upper triangular matrix is based on SGEMM in combination with block subdivision. This version divides A, A_h, and the resulting matrix of size $Nmat \times Nmat$ along dimension $Nmat$ into blocks of a certain length. This results in a set of quadratic and triangular blocks covering the triangular result matrix, similar to the matrix shown in Figure 5b for the parallel version. The sequential version uses SGEMM to calculate the quadratic blocks in the upper triangular matrix and uses a series of dot products to compute the triangles along the diagonal.

The parallel version partitions matrices A and A_h blockwise into $2 \cdot np$ blocks and assigns the blocks to the processors as we described previously. Each processor calculates those blocks of the result matrix for which it possesses the necessary block of A. So, processor 0 computes the first and the last line of blocks, processor 1 the second and last but one, and so on. This results in an ideal load balance, because all processors are responsible for calculating exactly the same number of blocks—namely, $2 \cdot np - 1$ quadratic and two triangular blocks. However, each processor possesses only two of the blocks of A_h and must fetch the other blocks from remote PEs to calculate its two lines of blocks.

A closer look at Figure 5b shows a slight imbalance in the communication: PE 0 must fetch $2 \cdot (np - 1)$ blocks, while all other PEs can reuse at least one of the blocks stemming from remote PEs. Because communication is very fast, this imbalance is inconsequential. Nevertheless, communication limits the parallel speedup to some degree. This, however, is unavoidable when the subroutine needs all pairs of elements of a distributed vector or matrix.

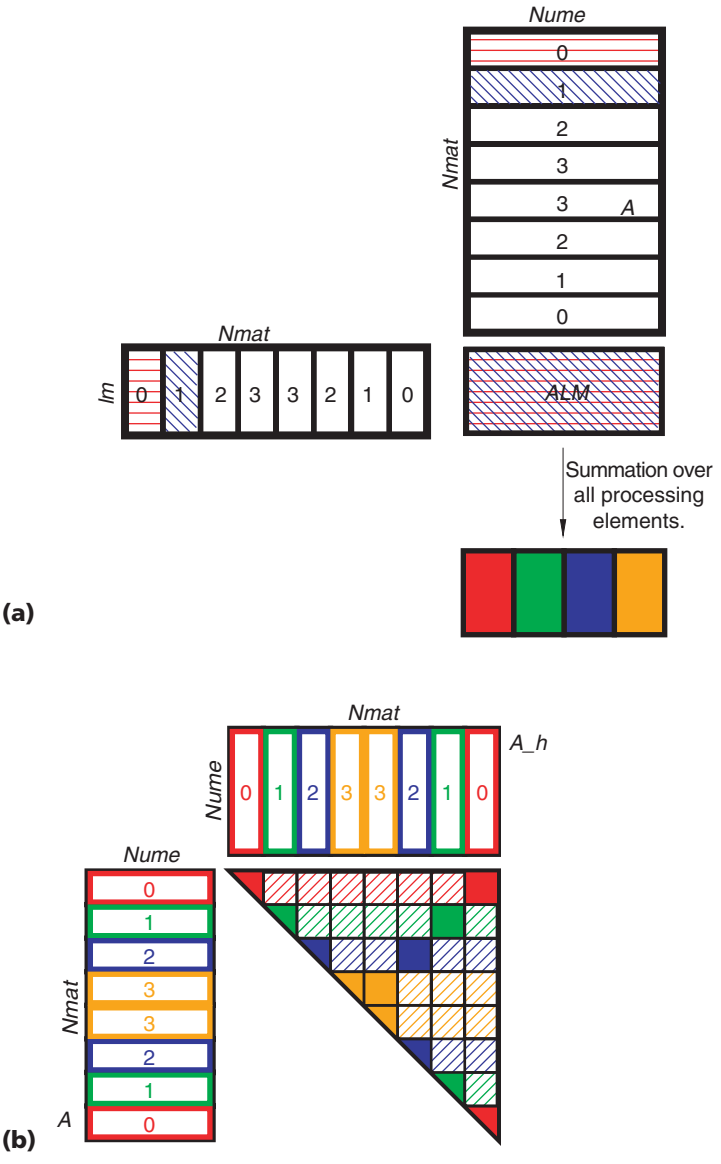


Figure 5. Parallel computation of the central matrix–matrix products for the LAPW2 subroutines, for four processors: (a) In L2MAIN, each processing element calculates a partial sum of the complete result matrix *ALM*. (b) In FOURIR, each PE has the data to calculate the filled blocks of its associated triangular matrix, but calculating the hatched blocks requires data from remote PEs.

Examples and performance measurements

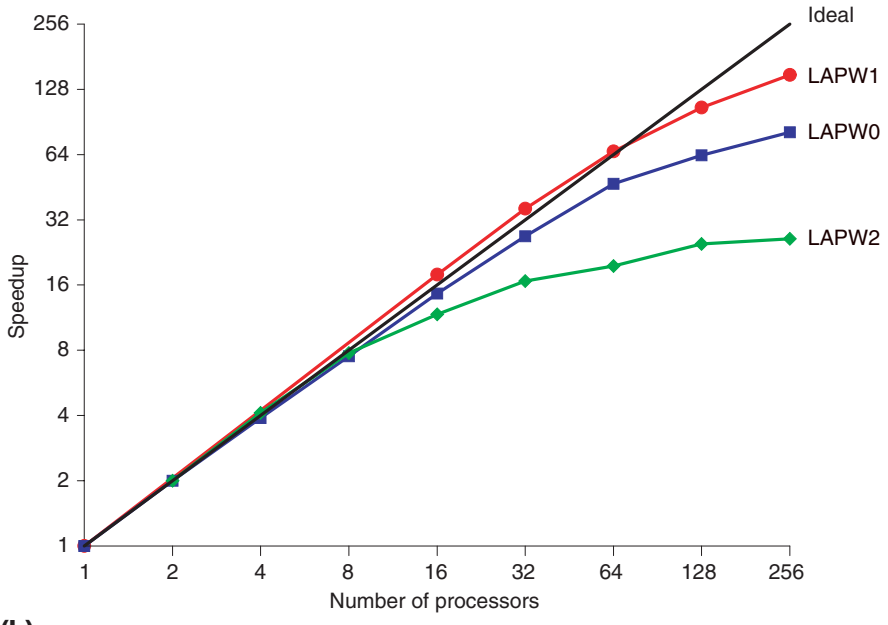
We've checked the parallel program's correctness with several typical test cases covering different application areas and both methods of eigenmatrix solution. A direct comparison with the sequential version running on an IBM RS6000 confirmed that both versions produce identical results within the bounds of the numeric precision of the two machines.

To figure out the parallel program's speedup on the T3E, we chose the rh2x2.31 data set, which just fits into a single T3E processor's memory. This case refers to a 31-layer slab of the (2 × 2) rhodium (110) surface cell (125 atoms, 2,108 valence electrons) and eight layers of vacuum, which is repeated periodically in all three dimensions. For the measurements in this section, we calculated six **k**-points; LAPW1 calculated 1,200 eigenvalues per **k**-point. Figure 6a shows the execution times for all three subprograms and the total execution time for one iteration, with LAPW1 referring to full diagonalization. We did not use any **k**-point parallelism, so the performance measurements in Figure 6b reflect the pure speedup yielded by the message-passing program. LAPW0 and LAPW1 (the dominant part) scale very well. So, the somewhat lower efficiency of LAPW2, resulting from the relatively large, unavoidable amount of communication time, is of no great consequence, and the total code's speedup is quite satisfactory. The superlinear speedup of LAPW1 for small processor numbers is due to the fact that the eigenvalue problem solver in the sequential program that we used as reference for the speedup performs somewhat more poorly than the ScaLapack solver that the parallel version uses.

The actual target problem size is, however, a matrix of approximately 10,000. As a typical example of this problem class, we have chosen test case cu4x2.9L14, a nine-layer slab of (4 × 2)-Cu (110) (that is, 72 atoms and 792 valence electrons). This slab repeats periodically in all three

	Execution time (seconds)					
	1 PE	16 PEs	32 PEs	64 PEs	128 PEs	256 PEs
LAPW0	890	61	33	19	14	11
LAPW1	30,387	1,695	847	457	288	203
LAPW2	2,212	189	132	113	89	84
Total	33,489	1,944	1,013	589	391	306

(a)

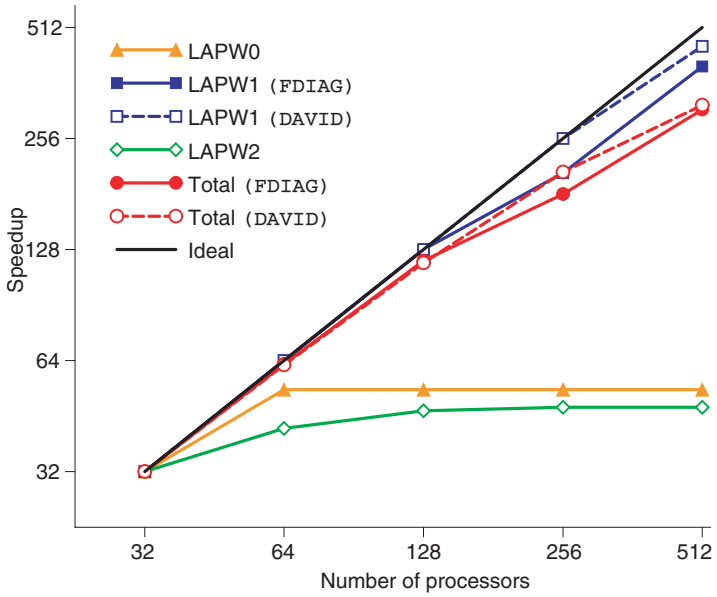


(b)

Figure 6. (a) Execution times and (b) parallel speedup of the pure message-passing versions of LAPW0, LAPW1, and LAPW2, and of the total code without k-point parallelism for the data set rh2x2.31 (matrix size 3,000, 1,208 eigenvalues, and six k-points) on the Cray T3E.

	Execution time (seconds)					
	IBM SP2	Cray T3E				
	1 PE	32 PEs	64 PEs	128 PEs	256 PEs	512 PEs
LAPW0	345	20	12	12	12	12
LAPW1 (FDIAG)	66,840	5,076	2,538	1,269	788	405
LAPW1 (DAVID)	41,344	4,008	2,004	1,002	501	282
LAPW2	3,597	203	155	139	136	136
Total time (FDIAG)	70,782	5,299	2,705	1,420	936	552
Total time (DAVID)	45,286	4,231	2,171	1,152	649	430

(a)



(b)

Figure 7. (a) Execution times and (b) parallel speedup of LAPW0, LAPW1, LAPW2, and the total code (including k-point parallelism) for the cu4x2.9L14 data set (matrix size 7,000, 700 eigenvalues, and eight k-points) on the Cray T3E. For comparison, Figure 7a includes the execution time on one IBM SP2 processor (thin node, 120 MHz). The speedup is relative to the execution time on 32 PEs; we defined the speedup value for 32 PEs as 32.

dimensions; between each slab is a vacuum zone equivalent to five substrate layers. We use a lattice constant of 6.64 bohr, which corresponds to the theoretical local-density approximation bulk value. Because both methods scale almost linearly with the number of **k**-points, we've used only one point in the surface Brillouin zone for these benchmarks. The MT sphere radii are 2.20 bohr. The kinetic-energy cutoff for the plane wave basis needed for the interstitial region is 13.22 Ry, which leads to an approximately 7,000 × 7,000 Hamiltonian matrix. The partial wave (l, m) representation (inside the MT spheres) is taken up to $l_{max} = 10$. We use a plane wave cutoff energy of 81 Ry for the Fourier representation of the potential. The maximum angular momentum in the (l, m) expansion of the potential inside the atomic spheres is set to $l_{max} = 4$.

Figure 7a shows the total program's execution time on the T3E for cu4x2.9L14, with matrix size 7,000, a calculation of 700 eigenvalues, and eight **k**-points. This example exploited all kinds of parallelism, including that with respect to the **k**-points. For LAPW1, Figure 7a gives the execution time for full diagonalization and for the iterative Davidson algorithm. The minimum number of processors necessary to run this case on the T3E is 32 owing to memory constraints. For comparison, the figure gives the execution time on one IBM SP2 processor (thin node, 120 MHz, 1 Gbyte of main memory).

The parallel LAPW1 especially profits from the increased problem size and, of course, from **k**-point parallelism. For this larger data set, a remarkable performance gain compared to one IBM SP2 processor is obtained. The total performance is 9 Gflops, resulting in an execution time of approximately 25 minutes on 128 T3E processors, compared to approximately 20 hours on one IBM SP2 processor. On 512 processors, the execution time reduces further to approximately 10 minutes.

Our parallel implementation is successful for several reasons. We have found efficient parallel solvers appropriate for the eigenvalue problems of this code. Also, we have been able to define distributed data structures that hold for a balanced parallel execution of large parts of the code, before reorganization and subsequent communication become necessary. Besides the inherent unavoidable communication—for example, in LAPW2, where all pairs of elements of distributed arrays are needed—the implemented version on the T3E contains some "artificial" communication. This communication is due to the restricted memory resources on the T3E and is necessary because we have been forced to distribute data that is needed on each PE in full. This slight disadvantage is, however, compensated by the T3E's very fast communication network and the capability of one-sided communication. These properties let us optimize load balancing in the different parts of the code by means of appropriate data structures, for the resulting reorganization and consequent communication required when going from one part to another does not decrease the efficiency significantly.

A planned conversion to MPI will let the program work on other platforms.

We're planning a conversion to the Message Passing Interface portable communication library in the near future. This conversion will let the parallel message-passing program work on parallel computers other than the T3E. We need to exchange the program's Shmem routines with the corresponding MPI routines. Also, we need to replace the Scilib routines, in the worst case with explicitly coded subroutines. Because the code's parallelization has drastically reduced execution time, compared to conventional scalar computers, we will be able to tackle a new class of larger problems. CiSE

Acknowledgments

We thank Peter Blaha of the Technical University of Vienna for advice and helpful discussion concerning all aspects of the WIEN97 implementation of FP-LAPW.

References

1. D.J. Singh, *Planewaves, Pseudopotentials, and the LAPW Method*, Kluwer Academic, Boston, 1994.
2. H.J.F. Jansen and A.J. Freeman, "Total-Energy Full-Potential Linearized Augmented Plane-Wave Method for Bulk Solids: Electronic and Structural Properties of Tungsten," *Physical Rev. B*, vol. 30, no. 2, July 1984, pp. 561–569.
3. P. Blaha, K. Schwarz, and P. Herzig, "First-Principles Calculation of the Electric Field Gradient of Li_3N," *Physical Rev. Letters*, vol. 54, no. 11, Mar. 1985, pp. 1192–1195.
4. D.J. Singh et al., "Electronic Structure and Heavy-Fermion Behavior in LiV_2O_4," *Physical Rev. B*, vol. 60, no. 24, 15 Dec. 1999, pp. 16359–16363.
5. K. Schwarz, H. Ripplinger, and P. Blaha, "Electric Field Gradient Calculations of Various Borides," *Zeitschrift für Naturforschung* (*Magazine for Natural Science*), vol. 51a, nos. 5–6, May/June 1996, pp. 527–533.
6. O. Radar et al., "Electronic Structure of Two-Dimensional Magnetic Alloys: c(2 x 2) Mn on Cu(100) and Ni(100)," *Physical Rev. B*, vol. 55, no. 8, Feb. 1997, pp. 5404–5415.
7. X.-G. Wang, A. Chaka, and M. Scheffler, "Effect of the Environment on α-Al_2O_3 (0001) Surface Structures," *Physical Rev. Letters*, vol. 84, no. 16, 17 Apr. 2000, pp. 3650–3653.
8. B. Kohler et al., "Force Calculation and Atomic-Structure Optimization for the Full-Potential Linearized Augmented Plane-Wave Code WIEN," *Computer Physics Comm.*, vol. 94, no. 1, Mar. 1996, pp. 31–48.
9. M. Petersen et al., "Improving the Efficiency of FP-LAPW Calculations," *Computer Physics Comm.*, vol. 126, no. 3, 11 Apr. 2000, pp. 294–309.
10. P. Blaha et al., "Full-Potential Linearized Augmented Plane Wave Programs for Crystalline Systems," *Computer Physics Comm.*, vol. 59, 1990, pp. 399–415.
11. D.M. Wood and A. Zunger, "A New Method for Diagonalising Large Matrices," *J. Physics A*, vol. 18, no. 9, 21 June 1985, pp. 1343–1359.
12. G. Kresse and J. Furthmüller, "Efficiency of Ab-Initio Total Energy Calculations for Metals and Semiconductors Using a Plane-Wave Basis Set," *Computational Materials Science*, vol. 6, no. 1, July 1996, pp. 15–50.
13. G. Kresse and J. Furthmüller, "Efficient Iterative Schemes for Ab Initio Total-Energy Calculations Using a Plane-Wave Basis Set," *Physical Rev. B*, vol. 54, no. 16, Oct. 1996, pp. 11169–11186.
14. E.R. Davidson, "The Iterative Calculation of a Few of the Lowest Eigenvalues and Corresponding Eigenvectors of Large Real-Symmetric Matrices," *J. Computational Physics*, vol. 17, no. 1, Jan. 1975, p. 87.

Renate Dohmen is a computational physicist at the computing center of the Max-Planck-Gesellschaft at the Max-Planck-Institute for Plasma Physics. Her research interest is parallel scientific computing. She received her PhD in experimental physics from the University of Münster; her thesis dealt with pattern formation in electrical systems such as semiconductor and gas discharge systems. Contact her at Rechenzentrum Garching, Boltzmannstr. 2, D-85748 Garching, Germany; dohmen@rzg.mpg.de.

Jakob Pichlmeier is a high-performance computing consultant with the Hitachi Europe HPC Group. He was previously a benchmarking and application consultant at Cray Research and a system analyst at

Control Data GmbH. He received his Diplom Informatiker from the Technische Universität München. Contact him at Hitachi Europe GmbH, Tecnopark IV, Lohstr. 28, D-85445 Schwaig-Oberding, Germany; jakob@hpcc.hitachi-eu.co.uk.

Max Petersen is a postdoctoral student in a joint project between UCLA's Department of Mathematics and Georgia Tech's School of Physics. He is developing models to simulate epitaxial growth using the level-set method. His research focuses on effects of reversibility and strain. He obtained his PhD in theoretical physics at the Fritz-Haber-Institut of the Max-Planck-Gesellschaft in Berlin. In his thesis he used density functional theory combined with the full-potential linear augmented plane wave method to study interactions of metal surfaces with atomic systems such as hydrogen, helium, and neon. Contact him at the Dept. of Mathematics, UCLA, Box 951555, Los Angeles, CA 90095-1555; max@math.ucla.edu.

Frank Wagner works in the high-performance computing group at the Leibniz Computer Center in Munich, Germany, supporting scientific users in running and optimizing their codes on Germany's "Federal Supercomputer in Bavaria." He previously worked at the Fritz-Haber-Institut of the Max-Planck-Gesellschaft in Berlin, studying transition metal surfaces and their interactions with molecules (especially carbon monoxide), using density functional theory combined with the full-potential linear augmented plane wave method. He obtained his master's in physics from the Technical University of Berlin. Contact him at Leibniz-Rechenzentrum, Gruppe Hochleistungsrechnen, Barer Str. 21, D-80333 München, Germany; wagner@lrz.de.

Matthias Scheffler is the director of the Theory Department of the Fritz-Haber-Institut der Max-Planck-Gesellschaft and a professor at the Technical University Berlin. He is also a researcher working in condensed-matter theory, materials, and the chemical physics of surfaces. At present, his main interest is to develop first-principles methods (using density functional theory) for molecular simulations that bridge the time and length scales from those of the atomistic processes to those that determine the properties of realistic systems. Contact him at Fritz-Haber-Institut der Max-Planck-Gesellschaft, Faradayweg 4-6, D-14195 Berlin-Dahlem, Germany; scheffler@fhi-berlin.mpg.de; www.fhiberlin.mpg.de/th/th.html.