

# Atomic wavefunction initialization in *ab initio* molecular dynamics using distributed Lanczos

C. Bekas, A. Curioni and W. Andreoni  
IBM Research, Zurich Research Laboratory  
CH-8803 Rüschlikon, Switzerland  
{bek, cur, and}@zurich.ibm.com

## Abstract

We present a distributed scheme for initialization from atomic wavefunctions in *ab initio* molecular dynamics simulations. Good initial guesses for approximate wavefunctions are very important in order to enable practical simulations with thousands of atoms. The new scheme is based on a distributed implementation of the Lanczos algorithm for very large dense eigenproblems. We show that the massively parallel BG/L supercomputer with its very fast separate network for collective communications is an ideal platform for the parallel Lanczos algorithm. We have implemented the new scheme in the popular plane-wave code CPMD. We showcase the applicability of the distributed initialization by a series of examples on a family of Silicon super cells ranging from 512 to 2048 atoms.

## 1 Introduction and motivation

The core problem in *ab initio* electronic structure calculations in the Density Functional Theory (DFT) framework consists of solving a nonlinear eigenvalue problem

$$H_\rho u_\rho = \lambda_\rho u_\rho, \quad (1)$$

where  $H_\rho$  is the DFT Hamiltonian operator and  $(\lambda_\rho, u_\rho)$  are the corresponding eigenvalue-eigenfunction pairs. Let the system under study have  $M$  valence electrons. Then,  $\rho$  is the charge density, originated by these valence electrons, and the nonlinearity stems from the fact that the Hamiltonian operator is a functional of the charge density  $\rho$  which is in turn defined by the eigenfunctions  $u_\rho$  that correspond to the  $M$  algebraically smallest eigenvalues  $\lambda_\rho$ . Density Functional Theory, which is founded on a series of seminal papers by Kohn, Hohenberg and Sham [7, 9] in the early 1960's, establishes that basically all interesting ground state quantum-mechanical properties of a molecular system can be derived from the charge density of the ground state.

To enable numerical simulations, the Hamiltonian operator  $H_\rho$  and the eigenfunctions  $u_\rho$  are discretized using a suitable basis and then (1) becomes a non-linear algebraic eigenvalue problem. Most *ab initio* simulation codes use a plane-wave basis and thus the eigenfunctions  $u_\rho$  are represented in Fourier space. The charge density  $\rho(j)$  at the  $j$ -th point in real (Euclidian) discretized 3D-space is given by

$$\rho(j) = \sum_{i=1}^M |u_i(j)|^2, \quad (2)$$

where for the sake of economy in notation we have substituted the  $\rho$  subscript in the eigenfunction with its index  $i$ . Thus,  $u_i(j)$  is the  $j$ -th entry of the  $i$ -th eigenvector of the discretized Hamiltonian  $H_\rho$ .

In order to solve equation (1) two main approaches have been heavily used:

**Self Consistent Field (SCF) iteration:** Relies on a linearization of (1). We start with an educated guess for the charge density  $\rho^{(0)}$  which thus defines an initial Hamiltonian  $H_\rho^{(0)}$ . Then, we calculate the  $M$  algebraically smallest eigenvalues of  $H_\rho^{(0)}$ , using the eigensolver of our choice. Thus, we are ready to define the next charge density  $\rho^{(1)}$ , using (2) and a possible mixing with the previous charge density. SCF can be viewed as a form of inexact or quasi Newton approach. The process is repeated until the change in the charge density falls below a predetermined tolerance. Then, the final Hamiltonian as well as the eigenvectors  $u_i$  are declared to be self consistent.

**Direct Minimization:** We seek to directly minimize the quadratic forms

$$u_i^* H u_i, \quad i = 1, \dots, M, \quad (3)$$

where  $u_i^*$  denotes the conjugate transpose of the eigenvector  $u_i$ . This amounts to non-linear optimization and very successful methods such as the Car-Parrinello method [2] and Conjugate Gradient [12] have revolutionized the field of electronic structure calculations, enabling simulations with thousands of atoms. In contrast to the SCF methodology, here we do not require a good initial charge density  $\rho^{(0)}$  but rather good initial guesses for the eigenvectors  $u_i^{(0)}$ . It is common experience that for systems with an increasing number of atoms good guesses are absolutely crucial for rendering the simulations tractable.

In this paper we concentrate on the initialization problem of Direct Minimization techniques. In particular, our main objective was to come up with a scalable solution suitable to utilize hundreds of processors, in view of massively parallel computing platforms such as the Blue Gene/L supercomputer. We have implemented our method within the plane-wave code CPMD [3]. Thus, to some extent some of our design decisions respect the general layout and data structures inherent in plane-wave codes. However, we stress that the methodology could be easily adapted to other codes. BG/L is a distributed memory

supercomputer. The parallel programming environment is the Message Passing Interface (MPI)<sup>1</sup>.

The main ingredient of the proposed approach is a Lanczos iteration with standard Gram-Schmidt reorthogonalization. A large enough Krylov subspace is built in parallel on which a suitably restricted Hamiltonian is projected and thus initial guesses for the eigenfunctions are calculated. The accuracy of the initial guesses can be easily controlled by monitoring the convergence of approximate eigenvalues. The latter can be a very useful feature, especially when very large systems are involved, since we are interested in approximate initial guesses for the eigenfunctions in the first place.

The host application CPMD, is a plane-wave code for *ab initio* molecular dynamics simulations. CPMD has been demonstrated to achieve exceptional scalability on massively parallel systems. In particular, 110.4 Tflops were achieved on the BG/L supercomputer<sup>2</sup> [8]. However, before this work the initialization phase was serial, essentially replicated on every processor, which restricted the number of atoms that could be simulated. Thus, the distributed initialization is, among others, a crucial ingredient in order to facilitate next generation simulations that involve tens of thousands of atoms. If we were to start the simulation from random wave-functions, it would require a very long number of iterations for the initial direct minimization, thus limiting the practical size of the systems for study. We have recently been able to calculate the ground state of 8000 Silicon atoms, which resulted to a restricted dense Hamiltonian of dimension 32000. Using 350 cpus of a BG/L system the initialization phase required less than 10 minutes.

## 2 Initialization from atomic orbitals

Consider a molecular system with  $N$  atoms and  $M$  valence electrons. Obviously, it is far easier to solve equation (1) for each atom separately, i.e. to define a separate Hamiltonian for each atom of the system and thus producing a set of “atomic” wavefunctions for each atom. Then, we can approximate the solution to the complete problem by superimposing these single atom wavefunctions. Indeed, electronic structure codes make use of precalculated “atomic” wavefunctions for different types of atoms. Thus, from a linear algebra point of view, atomic wavefunctions initialization consists of restricting the full system Hamiltonian operator on a large enough basis of atomic wavefunctions of dimension  $k > M$  and then solving for the  $M$  smallest eigenvectors of the restricted Hamiltonian.

Formally put, let matrix  $W_k \in \mathbb{C}^{n \times k}$  be the expansion of the atomic wave functions on the basis of  $n$  plane-waves, where each column of  $W_k$  corresponds to a single atom wavefunction. The standard methodology for initialization from atomic wavefunctions, which we improve in this paper, proceeds as follows:

---

<sup>1</sup><http://www-unix.mcs.anl.gov/mpi/>

<sup>2</sup>See [www.cpmc.org](http://www.cpmc.org)

1. Compute the restricted Hamiltonian:  $\tilde{H}_k = W_k^* H W_k$  and the overlap matrix  $O_k = W_k^* W_k$ . Matrices  $\tilde{H}_k \in \mathbb{C}^{k \times k}$  and  $O_k \in \mathbb{C}^{k \times k}$  are both Hermitian.
2. Calculate the eigen-decomposition of the restricted generalized Hermitian eigenproblem

$$\tilde{H}_k x = \lambda O_k x. \quad (4)$$

3. Approximate the  $M < k$  desired initial wavefunctions as  $U_m = W_k X_m$ , where the columns of  $X_m$  hold eigenvectors that correspond to the  $m$  smallest eigenvalues of the restricted generalized eigenproblem (4).

The calculation of the restricted Hamiltonian  $\tilde{H}_k$  and of the overlap matrix  $O_k$  is performed in parallel since plane-waves are distributed among processors:

1. Each processor: Calculate the application of the Hamiltonian  $H$  to its set of plane-waves:  $H W_k$ .
2. Each processor: Calculate the overlap  $W_k^* (H W_k)$ .
3. All processors: Calculate matrix  $\tilde{H}_k$  using global summation of  $W^* (H W_k)$  among all processors.

In CPMD the solution of the generalized eigenproblem (4) was not distributed across available processors, but rather solved exclusively on a single processor. We next illustrate that this practice, although perfectly adequate for conventional simulations, is absolutely impractical for next generation target simulations that will involve tens of thousands of atoms. Instead, we propose a fully parallel initialization from atomic wavefunctions that is based on the parallel Lanczos algorithm.

## 2.1 Parallel initialization using Lanczos

The dimension  $k$  of the restricted Hamiltonian  $\tilde{H}_k$  is immediately linked to the number of valence electrons  $M$ , and thus with the total number of atoms  $N$ . Thus, in the context of large simulations that involve several thousands of atoms, the dimension of the restricted Hamiltonian  $\tilde{H}_k$ , which is a dense matrix, will be in the order of tens of thousands. Clearly, i) storage requirements, in the order of  $\mathcal{O}(k^2)$  ii) as well as computational complexity of the generalized eigenproblem (4), in the order of  $\mathcal{O}(k^3)$ , render the calculation intractable on a single processor. The following observations are key in the design characteristics of a fully parallel approach:

- Matrices  $\tilde{H}_k, O_k$  are dense. Solution of the eigenproblem (4) will require the transformation of these matrices to simpler form. Namely, since this is a Hermitian generalized eigenproblem, it can be transformed to a simple eigenproblem, by means of a Cholesky factorization of the overlap matrix  $O_k$ , and then a reduction to tridiagonal form of matrix  $O_k^\dagger \tilde{H}_k$  is needed, where  $O_k^\dagger$  is the pseudoinverse of  $O_k$ , using the Cholesky factors.

- The calculation of the eigendecomposition of the resulting tridiagonal matrix will require  $\mathcal{O}(k^2)$  storage. Thus, it must be done in parallel.
- The new approach should exploit the current distribution of all involved matrices in terms of the distribution of plane-waves across processors.

In light of the above we propose to use the Lanczos algorithm for iterative partial tridiagonalization of a modified restricted Hamiltonian.

### 2.1.1 The Lanczos algorithm

Consider a symmetric matrix  $A$  and a starting vector  $v_1$  such that  $\|v_1\|_2 = 1$ , where  $\|\cdot\|_2$  denotes the standard Euclidian norm. The Lanczos algorithm computes an orthonormal basis for the Krylov subspace

$$\mathcal{K}_l(A, v_1) = \text{span}\{v_1, Av_1, A^2v_1, \dots, A^{l-1}v_1\}. \quad (5)$$

In particular, after  $l$  steps of the Lanczos algorithm for matrix  $A$  and starting vector  $v_1$ , the following Lanczos factorization holds:

$$AV_l = V_lT_l + \beta_{l+1}v_{l+1}e_l^*, \quad (6)$$

where  $V_l$  is the orthonormal basis for  $\mathcal{K}_l(A, v_1)$  and  $T_l$  is a symmetric tridiagonal matrix with structure

$$T_l = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \beta_{l-1} & \alpha_{l-1} & \beta_l & \\ & & & \beta_l & \alpha_l & \end{bmatrix}. \quad (7)$$

When  $l$  is taken to be equal to the dimension  $n$  of matrix  $A$ , then matrix  $T_n$  will have the same eigenvalues as  $A$ . Thus, for  $l < n$ , the Lanczos algorithm can be viewed as a means for partial reduction to tridiagonal form for matrix  $A$ .

### 2.1.2 A preprocessing step

A preprocessing step that will significantly simplify the whole process is orthogonalization of matrix  $W_k$ . Then, the overlap matrix  $O_k$  reduces to the identity matrix  $O_k = W_k^*W_k = I_k$  and the new restricted eigenproblem becomes a standard one:  $H_kx = \lambda x$ .

The cost of orthogonalizing  $W_k$  is in the order of  $\mathcal{O}(nk^2)$ . We have adopted the approach used in CPMD. Let  $Y_k$  be a matrix the columns of which are the orthonormalized columns of  $W_k$ . Then, we seek a matrix  $R$  such that

$$Y_k = W_kR, \quad \text{such that } Y_k^*Y_k = I_k.$$

Substituting the first equation above into the second one we find that  $R$  can be taken to be the inverse of the Cholesky factor of the overlap matrix  $S_k = W_k^\top W_k$ .

Observe that calculating matrix  $S_k$  is a BLAS 3 operation which is also straightforward to parallelize since  $W_k$  is row-wise distributed across the processors through its plane-wave expansion. The cost of computing the Cholesky decomposition of  $S_k$  is  $\mathcal{O}(k^3)$ , for which we utilize a column-wise parallel implementation, already available in CPMD. Finally, solving with the inverse of the Cholesky factor, which is already parallelized in CPMD as well, requires an additional  $\mathcal{O}(k^3)$ . For what follows we consider  $W_k$  to have orthonormal columns.

### 2.1.3 The distributed initialization

The working matrix is the new restricted Hamiltonian  $W_k^* \tilde{H}_k W_k$  (see the previous section). Then, the proposed fully distributed initialization method proceeds as follows:

1. Calculate a Lanczos factorization for matrix  $H_k = W_k^* \tilde{H}_k W_k$  (see Fig. 1)

$$(W_k^* \tilde{H}_k W_k) V_l = V_l T_l + \beta_{l+1} v_{l+1} e_l^*, \quad (8)$$

where  $M < l \leq k$ , matrix  $T_l \in \mathbb{R}^{l \times l}$  is symmetric tridiagonal and matrix  $V_l \in \mathbb{C}^{k \times l}$  has orthonormal columns. The eigenvalues of the restricted Hamiltonian  $H_k = W_k^* \tilde{H}_k W_k$  are approximated by the eigenvalues of matrix  $T_l$ . Factorization (8) is calculated by means of the Lanczos algorithm (see Fig. 1), which requires only a matrix-vector product with matrix  $(W_k^* H W_k)$ . It is clear that this matrix need not be formed, but rather the product  $(W_k^* H W_k) y$  with a vector  $y$  can be calculated as a series of matrix-vector products (with matrices that are already distributed across processors).

2. Notice however, that if we let the length  $l$  of the Lanczos basis  $V_l$  approach the size  $k$  of the restricted Hamiltonian  $H_k$ , then it is preferable to form the restricted Hamiltonian  $H_k$  explicitly and distribute it row-wise across the involved processors. This choice greatly simplifies the implementation of the parallel Lanczos algorithm, since the formation of  $H_k$  can already be done in parallel in CPMD. Furthermore, note that in CPMD the application of the Hamiltonian on a vector is fully distributed among all available processors. In this case, the restricted Hamiltonian  $H_k$  is distributed row-wise.
3. The basis  $V_l$  can be easily distributed row-wise across the available processors, as is the standard approach followed in parallel implementations of the Lanczos algorithm. Note then that, the `xAXPY` operations at lines 3 and 5 of the Lanczos algorithm (see Fig. 1) can be accomplished with no communication whatsoever. The only synchronization points are in line 4 and in line 6, which require global reduction operations.
4. Monitoring the convergence of eigenvalues can be cheaply calculated at every step of Lanczos. The Lanczos iteration is a variational process: approximations to eigenvalues at step  $i+1$  will always be better than those

of the previous step. Extremal eigenvalues tend to converge first: thus, one can monitor convergence of the smallest eigenvalue of the tridiagonal matrix  $T_l$  (at every step  $l$ ) and when this has converged move to monitoring convergence of the next one. Cheap algorithms for the calculation of a few selected eigenvalues of symmetric tridiagonal matrices are available in LAPACK. The cost of computing only the eigenvalues (not the eigenvectors) will be at most a modest  $\mathcal{O}(l^2)$  with storage requirements  $\mathcal{O}(l)$ , since  $T_l$  is tridiagonal. Thus, it can be easily achieved on a single processor.

5. When convergence of all  $M$  desired (leftmost) eigenvalues has been achieved, these are distributed to the available processors. Then, each processor will do a small number of inverse iteration steps with the exact eigenvalue as shift:  $(T_l - \lambda_i I)^{-1} q_i$  on a random starting vector  $q_i^{(0)}$ . Two to three iterations should be enough to achieve very good convergence to targeted eigenvectors  $q_i$  of  $T_l$ . Observe that  $q_i \in \mathbb{R}^l, l \ll n$ , thus storage requirements per processor are kept very small. However, a potential problem with this approach can arise when the converged eigenvalues are closely clustered. Alternatively, we can utilize “divide and conquer” techniques also available through LAPACK expert drivers such as `xSYEVX`, in which the user can choose to compute particular consecutive eigenvalues/eigenvectors. Thus, we can easily distribute the calculation of wanted eigenpairs on the involved processors. Each processor calculates only a number of consecutive eigenpairs by suitably calling routine `xSYEVX`. We adopted this latter approach in our current implementation the method in CPMD.
6. The approximate eigenvectors for the modified restricted Hamiltonian are computed as:  $\tilde{q}_i = V_l q_i$ . Since the basis  $V_l$  is distributed row-wise this calculation is performed in parallel. Note that, because a processor holds complete eigenvectors  $q_i$ , the calculation of  $\tilde{q}_i$  will require a loop of broadcast collectives. Each processor will again end up with complete (consecutive) eigenvectors  $\tilde{q}_i$ .
7. Finally, approximations to wave functions are similarly (see above) computed in parallel as  $x_i = W_k \tilde{q}_i$ . Note that matrix  $W_k$  is distributed row-wise while processors hold complete consecutive eigenvectors  $\tilde{q}_i$ .

In Fig. 1 we give an algorithmic outline of the Lanczos method.

## 2.2 Practical application of the parallel Lanczos algorithm on BG/L

The plane-wave code CPMD, with single processor initialization, has been demonstrated to achieve excellent scalability on the full scale BG/L system, consisting of 64K computing nodes<sup>3</sup> [8]. However, it is straightforward to see that in order for the new parallel initialization to scale analogously, a huge number atomic

---

<sup>3</sup>See [www.cpmid.org](http://www.cpmid.org)

<p><b>Lanczos</b></p> <p>(*Input*) Hamiltonian <math>\tilde{H}</math>, orthonormal matrix <math>W_k</math>,  starting vector <math>v_1</math>, <math>\ v_1\ _2 = 1</math>, scalar <math>l \leq k</math></p> <p>(*Output*) Orthogonal basis <math>V_l \in \mathbb{R}^{k \times l}</math>  unit norm vector <math>v_{l+1}</math> such that <math>V_l^\top v_{l+1} = 0</math></p> <ol style="list-style-type: none"> <li>1. Set <math>\beta_1 = 0</math>, <math>v_0 = 0</math></li> <li>2. <b>for</b> <math>i = 1, \dots, l</math></li> <li>3. <math>r_i = W_k^*(\tilde{H}(W_k v_i)) - \beta_i v_{i-1}</math></li> <li>4. <math>\alpha_i = \langle r_i, v_i \rangle</math></li> <li>5. <math>r_i = r_i - \alpha_i v_i</math></li> <li>6. <math>\beta_{i+1} = \ r_i\ _2</math></li> <li>7. <b>if</b> (<math>\beta_{i+1} == 0</math>) <b>then stop</b></li> <li>8. <math>v_{i+1} = r_i / \beta_{i+1}</math></li> <li>9. <b>end</b></li> </ol>
--

Figure 1: The Lanczos algorithm for matrix  $W_k^* \tilde{H} W_k$ . The inner product for vectors is denoted by  $\langle \cdot, \cdot \rangle$ .

wavefunctions  $k$  would be required (i.e. 1 million), which is far beyond our target. Thus, the implementation has to be able to utilize only a subset of the available processors. For instance, while all processors contribute in the calculation of the restricted Hamiltonian  $H_k$ , only a subset of them will actually be employed in the Lanczos iteration. To this end, matrix  $H_k$  is distributed row-wise to these processors. This is facilitated by means of a new MPI communicator for these processors. An additional benefit is that the collective communications will be restricted to only a subset of the machine, thus reducing the overall communication latency.

The matrix-vector operation (line 3) as well as the DAXPY operations (line 3, 5) require no communication. On the other hand, the calculation of scalars  $\alpha_i, \beta_i$  require global reductions (MPI\_ALLREDUCE). For such collective communications we utilize the tree network of the BG/L system, which sustains 2.8 Gb/s of bandwidth per link with a latency of tree traversal no more than 2.5  $\mu$ s.

**Reorthogonalization of Lanczos vectors** It is well known that although the Lanczos iteration theoretically ensures orthogonality among the basis vectors, in practice the basis vectors quickly lose orthogonality due to roundoff. To remedy this we employ reorthogonalization at each step. We stress though that future versions of CPMD will utilize techniques for partial reorthogonalization [1, 10, 11], that perform orthogonalization only when it is deemed necessary. Reorthogonalization is performed by means of standard Gram-Schmidt [5]. At each step  $i$ , and before calculation of scalar  $\beta_{i+1}$ :

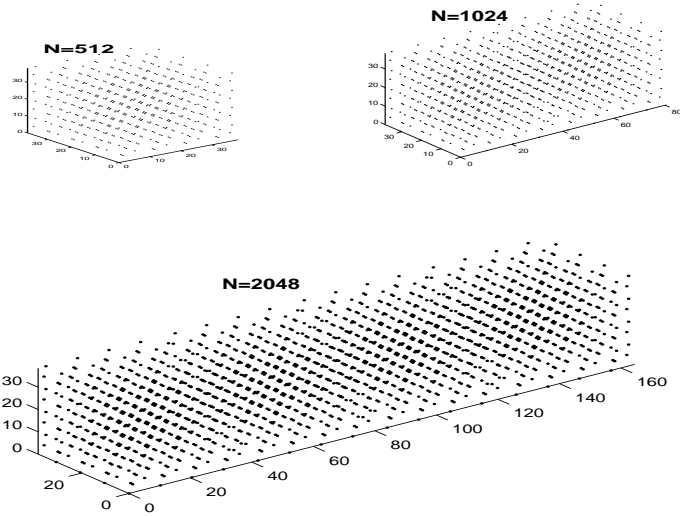


Figure 2: Silicon clusters with  $N=512$ , 1024 and 2048 atoms. The last two clusters were generated by replication of the smallest cluster along the  $X$  axis.

- Compute the local projection coefficient  $w_i^l = V_{i-1}^* r_i$ . No communication required.
- Compute the global projection coefficients  $w_i^g$  by global reduction on the local projection coefficients  $w_i^l$  (`MPI_ALLREDUCE`).
- Compute the reorthogonalized  $r_i^r = r_i - \sum_{j=1}^{i-1} w_j^g V_j$ . No communication required.

Observe that standard Gram-Schmidt (GS) reorthogonalization induces only an additional collective operation per Lanczos step. We note that although modified GS is known to be more stable than standard GS (see [5]), it requires  $i-1$  additional collectives at each step  $i$ , which includes  $O(l^2)$  additional collectives for a total of  $l$  Lanczos steps. Since we are only interested in initial guesses for the wavefunctions we opt to use standard GS. In practical applications so far we have not encountered a problem of unrecoverable severe loss of orthogonality. However, if this happens, we can always temporarily switch to modified GS.

### 3 Numerical Examples

We now illustrate the scalability performance of the parallel Lanczos algorithm for distributed atomic wavefunctions initialization. We experimented with a

family of super cells of silicon bulk, ranging  $N = 512, 1024$  and  $2048$  atoms (see Fig. 2). For the first case ( $N=512$ ) we used a cutoff energy of 20 Rydbergs while for the larger cases of  $N = 1024, 2048$  atoms the cutoff energy was set to 12 and 8 Rydbergs respectively. The cutoff energy controls the dimension of the plane-wave basis on which the Hamiltonian and the wavefunctions are expanded (a larger value for the cutoff energy results into more plane-waves).

The smallest case ( $N=512$ ) is a cubic mesh (with cube edge equal to 41.0449) and the larger two cells were generated by replication of the smallest cell along the  $X$  axis. The sizes of the restricted Hamiltonians were  $4 \times N = 2048, 4096, 8192$ . CPMD utilized in all runs 128 compute nodes of a BG/L system, while for the atomic wavefunctions initialization we utilized a subset of  $2^k, k = 1, \dots, 7$  nodes. We stress that the ability to utilize only a subset of the available compute nodes is crucial in achieving good scaling for the initialization, while other parts of CPMD can take advantage of the full set of available processors. We point out that although CPMD utilizes dual-core, dual FPU features available on the BG/L nodes by means of a highly optimized DGEMM matrix multiplication library [6], distributed Lanczos initialization utilizes a single core of the BG/L node. The other node handles communication tasks (runs known as co-processor mode).

Table 1 illustrates run times for the distributed initialization. The dashes in the columns of Table 1 indicate that no run was possible for the corresponding number of CPUs because the BG/L node memory (512 Mbytes) was not enough to hold the data. Remember that we have implemented the parallel Lanczos algorithm in CPMD in such a way that only a subset of the available processors are actually employed for the initialization. This versatility is crucial when we utilize thousands of processors with CPMD, since the minimization of the quadratic forms (3) has been demonstrated to sustain exceptional scalability on thousands of BG/L nodes [8]. It is clear that as the dimension of the restricted Hamiltonian increases, the distributed Lanczos algorithm scales better. For example, while for  $N = 512$  scaling stops at 32 processors, for  $N = 1024$  it stops at 64 and for  $N = 2048$  scaling continues up to 128 processors. We note that typically the run time for distributed Lanczos, using the best scaling, is in the order of one step of the minimization process for the quadratic forms (3).

In Table 2 we provide detailed run times (in seconds) for the various computational stages of the distributed initialization. MATVEC is the time spent multiplying the restricted Hamiltonian by the current Lanczos vector (line 3). REORTH is the time spent for reorthogonalizing the new Lanczos vector. COLL is the time for collective communications in the Lanczos loop (excluding the REORT). DAXPY is the time spent in BLAS daxpy operations within the Lanczos loop. L. EIGS is the time spent to calculate the Lanczos eigenvectors and eigenvalues (immediately after the Lanczos loop has run). VECS is the time spent to multiply the Lanczos eigenvalues with the atomic wavefunctions  $W_k$  to get the final estimation of the initial wavefunctions. Remember that matrix  $W_k$  is row-wise distributed across all available processors, while the eigenvectors of the restricted Hamiltonian are distributed column-wise across the group of processors that participate in the Lanczos run.

Clearly, the matrix-vector operation (MATVEC) scales very well in all cases and

N=512		N=1024	N=2048
#CPUs	time (secs)	time (secs)	time (secs)
2	90	-	-
4	46	381	-
8	23	191	-
16	13	99	738
32	9	54	382
64	8.6	36	211
128	8.7	30	114

Table 1: Run times for distributed initialization using the parallel Lanczos algorithm.

the DAXPY operations contribute only minimally to the overall cost. The computation of the eigenvalues and eigenvectors of the Lanczos matrix (L. EIGS) also scales every well, especially for the larger problems. Reorthogonalization exhibits satisfactory scaling which is attributed to the modest additional communication cost of standard GS and to the very fast collective communication tree network of the BG/L. Finally, we see that the part that does not scale is the final calculation of the approximate initial wavefunctions. This is expected, since the total number of messages in this part increases as we increase the number of processors in the Lanczos group. To see this, remember that the Lanczos eigenvectors are distributed column-wise to the group of processors that takes part in the Lanczos loop, i.e. each processor in this group holds a number of consecutive Lanczos eigenvectors. On the other hand, remember that matrix  $W_k$  is distributed row-wise across all of the available processors. Thus, this part of the calculation involves collectives that span the full breadth of the machine. Indeed, the number of these broadcast collective operations increases as the number of processors in the Lanczos group increases. However, we also note that for the larger case (N=2048) the VECS part is only a fraction of the total cost (roughly %20), which explains the very good scaling of the scheme for this case.

## 4 Discussion

Initialization from atomic wavefunctions in *ab initio* molecular dynamics codes is crucial in order to facilitate large scale next generation simulations with thousands of atoms. This initialization leads to very large dense eigenproblems that are impossible to solve on a single processor, both in terms of computational complexity as well as of memory requirements. In this paper we are reporting a new scheme for distributed initialization that is based on a distributed version of the Lanczos algorithm. Our decision to use Lanczos instead of parallel dense methods such as the ones in SCALAPACK [4] is based on the following key observations:

N=512						
#CPUs	MATVEC	REORT	COLL	DAXPY	L. EIGS	VECS
2	36	49.2	1.2	0.07	4.9	1.0
4	18	24.3	1.4	0.05	2.6	1.1
8	7.2	11.9	1.3	0.03	1.4	1.3
16	3	6.2	1.6	0.03	0.87	1.7
32	1.5	3.5	1.8	0.02	0.51	2.2
64	0.8	2.2	1.7	0.02	0.41	3.7
128	0.4	1.3	0.3	0.03	0.5	6.3

N=1024						
#CPUs	MATVEC	REORT	COLL	DAXPY	L. EIGS	VECS
4	141	201	20	0.12	15.1	4
8	7.0	97	9	0.09	9.7	4.4
16	35	48	6.5	0.07	4.8	5.1
32	14	24	5	0.06	2.7	6.9
64	6.5	13	4.8	0.06	1.7	9.6
128	3.2	7.7	1	0.05	1.2	16.8

N=2048						
#CPUs	MATVEC	REORT	COLL	DAXPY	L. EIGS	VECS
16	277	384	33	0.16	35	11.1
32	137	191	25	0.15	16	12.3
64	69	97	19	0.12	9	17.4
128	29	53	5	0.12	5	21.6

Table 2: Detailed run times, in seconds, for the various stages of distributed initialization using the parallel Lanczos algorithm.

- We needed to respect as much as possible the existing data structures in CPMD, which is the host molecular dynamics code. The distribution of the matrices involved, in terms of plane-waves, significantly favors matrix vector operations, rather than matrix transformations-factorizations that are inherit in parallel dense linear algebra.
- We can safely use in practice standard Gram-Schmidt reorthogonalization, instead of modified Gram-Schmidt, which induces only one additional collective operation at each step of the parallel Lanczos algorithm.
- Our target computational platform is the BG/L supercomputer which is equipped with an excellent separate network for collective communications.

We point out that although good scalability of the new scheme is of course a desired property it is not of crucial importance. This is due to the fact that the total run time of large simulations is by far dominated by the minimization

of the quadratic forms (3) (after atomic wavefunction initialization has run) and the subsequent molecular dynamics simulation thereafter. For example, for the case of 1024 Silicon atoms (see previous section) the new distributed initialization scheme on 8 BG/L nodes required 193 seconds while the total run time for minimization was 2400 seconds. Then, each step, out of the thousands, of the molecular dynamics run costs itself roughly the same as the distributed initialization. However, if it was not for this successful initialization from atomic wavefunctions, the first minimization would require an enormous number of iterations in order to converge.

In anticipation of very large simulations with tens of thousands of atoms for which we will be using tens or hundreds of thousands of processors, we strongly believe that the parallel Lanczos algorithm scheme we describe in this paper is very well suited for distributed atomic wavefunction initialization. We stress here that recently we were able to compute the ground state of 10000 Silicon atoms, on 4096 BG/L nodes, using the new distributed initialization scheme which is a problem size one order of magnitude larger than current typical simulations.

## References

- [1] C. Bekas, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Computing charge densities with partially reorthogonalized Lanczos. *Comp. Ph. Com.*, 171(3):175–186, 2005.
- [2] R. Car and M. Parrinello. Unified approach for molecular dynamics and density functional theory. *Phys. Rev. Lett.*, 55(22):2471–2474, 1985.
- [3] CPMD. Copyright IBM corporation 1990-2006, copyright MPI für Festkörperforschung Stuttgart 1997-2001.
- [4] J. Demmel, J. Dongarra, R. van der Geijn, and D. Walker. SCALAPACK: Linear algebra software for distributed memory architectures. In T.L. Casavant, P. Tvrđík, and F. Plášil, editors, *Parallel Computers: Theory and practice*, pages 267–282, Los Alamitos, CA, 1995. IEEE Computer Society Press.
- [5] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3d edition, 1996.
- [6] François Gygi, Robert K. Yates, Juergen Lorenz, Erik W. Draeger, Franz Franchetti, Christoph W. Ueberhuber, James C. Sexton, Bronis R. de Supinski, Stefan Kral, and John A. Gunnels. Large-Scale First-Principles Molecular Dynamics simulations on the BlueGene/L Platform using the Qbox code. In *Proceedings of Supercomputing 05*, pages 24–31, 2005.
- [7] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136(3B):B864–B871, 1965.

- [8] J. Hutter and A. Curioni. Car-parinello molecular dynamics on massively parallel computers. *Chem. Phys. Chem.*, 6:1788–1793, 2005.
- [9] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140(4A):A1133–A1138, 1965.
- [10] R. M. Larsen. PROPACK: A software package for the symmetric eigenvalue problem and singular value problems on Lanczos and Lanczos bidiagonalization with partial reorthogonalization, SCCM, Stanford University  
URL: <http://sun.stanford.edu/~rmunk/PROPACK/>.
- [11] H. D. Simon. Analysis of the symmetric lanczos algorithm with reorthogonalization methods. *Linear Algebra Appl.*, 61:101–132, 1984.
- [12] M. P. Teter, M. C. Payne, and D. C. Allan. Solution of Schrödingers equation for large systems. *Phys. Rev. B*, 40:12255–12263, 1989.