Fast matrix-vector multiplication for tight-binding hamiltonial matrices in the Lanczos method

D. Santo-Orcero[a], C. R. Zacharias[b], A. Falcó[c]

[a] *Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Complejo Tecnológico, Campus de Teatinos 29071 Málaga. Spain.*
[b] *Departamento de Fisica-Quimica, Universidade Estadual Paulista - UNESP, Av. Dr. Ariberto Pereira da Cunha, 333, 12.516-410- Guaratinguetá, Brazil*
[c] *Departamento de Ciencias, Físicas, Matemáticas y de la Computación, Universidad CEU Cardenal Herrera San Bartolomé 55 46115 Alfara del Patriarca (Valencia), Spain.*

Abstract

In this paper we introduce an fast matrix-vector multiplication for tight-binding hamiltonial matrices in the Lanczos method, to obtain a $O(N)$-matrix-vector multiplication algorithm. We will show that we are to be able to reduce the memory size used in the storage, from $O(N^2)$ to $O(N)$, and also to avoid the use of swap on disk. Moreover, we obtain an efficient use of the system cache, due to an improvement on the locality of the numerical operations. Some performances based on Silicon clusters are given.

Keywords: Matrix–vector multiplication , Tight-Binding Hamiltonian matrices, Lanczos Algorithm.

Email addresses: dsanto@lcc.uma.es (D. Santo-Orcero), zacha@feg.unesp.br (C. R. Zacharias), afalco@uch.ceu.es (A. Falcó)

1. Introduction

The numerical tools used to study larger systems, like pair-potential methods, usually fail on accuracy even in the case when few atoms are involved. Moreover, it is well-known that they cannot correctly evaluate the energy from larger molecules to crystalline solids.

The semi-empirical methods are a good compromise solution between the accuracy of "ab-initio" and the performance of pair-potential methods. One of the most common is the tight–binding method [20]. It has a good accuracy in order to compute the energy levels for elements, like Silicon [29, 19, 2, 22, 23, 20, 21, 27]. The eigenvalues of the the tight–binding Hamiltonian can also be used for molecular dynamics simulations when a suitable repulsive potential is added in the final energy expression [15, 11, 15, 24, 25]. In particular, for several atoms with directional bonds and with different possible hybridization, the pair-potential methods are useless [8]; and the tight–binding framework provides a good approximation of the energy levels.

The dimension N of a Hamiltonian matrix arising in the tight–binding method, can be written as $N = n_{orb}N_{at}$, where N_{at} is the number of atoms and n_{orb} is the number of Löwdin orbitals per atom. Then, in order to calculate the total energy, this matrix must be tridiagonalized. From a computational point of view, this fact has been recognized as the more important critical one. Several techniques based either on the concept of localized orbitals [9, 18, 26, 13, 14], or in calculation of the density matrix [17, 7, 1, 33, 3] have been proposed. There are a lot of methods that can be used to tridiagonalize Hamiltonian matrices, they have in common the same drawback: their cubical scaling on N.

One of these numerical methods is Lanczos algorithm [10] which has two interesting properties. The first is that it does not degrade the sparsity of the matrix. The second one, is that its bottleneck is produced by a vector-matrix multiplication producing a cubical scaling in the execution of the algorithm.

In this paper, we substitute the standard matrix–vector multiplication scaling quadratically, by one that scales linearly on N. Moreover, due to the improvement of the locality of the operations, it allows to a better use of the cache of the system and it also reduces the memory storage avoiding the swap on disk. We point out that this procedure strongly depends on the structure of the tight–binding Hamiltonian matrices, and that it can be applied to any variant of the Lanczos method, even if it includes any kind of re-orthogonalization process.

2. Computing the energy levels of atom clusters

2.1. The tight–binding Hamiltonian matrices

The Hamiltonian matrix $\mathcal{H}$ can be viewed as a partitioned matrix $N \times N$, which is formed by $n_{orb} \times n_{orb}$–matrices $\mathcal{H}_{i,j}$, for $1 \leq i, j \leq n_{orb}$. Each of these blocks is generated by the relationship between the i–th and the j–th cluster atoms. Then, $N_{at} := N/n_{orb}$ is the total number of atoms in the cluster. The composition of each block $\mathcal{H}_{i,j}$ is also intuitive: it models the interaction of the orbitals of i–th atom against the orbitals of j–th atom. This implies that the

$l \times m$–element of the $\mathcal{H}_{i,j}$–block is computed by using the relationships between the l–th orbital from i–th atom and the m–th orbital from the j–th atom. The blocks placed in the diagonal model the energy of the atom against itself and, in consequence, are diagonal matrices. The tight–binding Hamiltonian matrix $\mathcal{H}$ for large clusters of atoms is heavily sparse, due to the existence of a cut distance as we explain below. Some examples are given in Figures 1-3.

2.2. Obtaining the eigenvalues of the tight–binding Hamiltonian

The mathematical problem of tight–binding energy computations can be reduced to the diagonalization of the Hamiltonian matrix. Usually its eigenvalues and its eigenvectors are computed reducing this matrix to its tridiagonal form.

There are several algorithms in order to compute the tridiagonal form of a symmetric matrix. However, the most part of the computational time is used on the process of tridiagonalization of the matrix. In the better algorithms, the complexity of each one reduction scales cubically on N.

Moreover, some optimizations that can be used on heavily-sparse, banded matrices allow us to break the limit of the $O(N^3)$ complexity on a standard tridiagonalization. For example, some matrices can be internally reordered, keeping the values of the eigenvalues invariant. However, this is not the case for tight–binding due to the fact that the zero blocks are caused by the distance between a pair of atoms. Thus the atoms have a spacial relationship with physical sense in a 3D environment. We point out that the Hamiltonian of 1D atoms can be easily banded, and maybe 2D atoms clusters can also be banded with some restrictions, but those methods are outside the scope of this paper. Finally, there are algorithms such as Cuthill and McKee [6] that allow reordering of the Hamiltonian but the reordering algorithms for a generic 3D Hamiltonian matrix that can be found on the bibliography are far from giving a banded matrix on the general case, and they only reduce the maximum distance between atom labels.

Our proposed algorithm is valid for any cluster of atoms with physical sense in a 3D environment and does not need any reordering at all. Recall that the diagonalization methods, like Jacobi, completely destroy the sparsity [30] and it

2.3. The common Lanczos method

The Lanczos method is based on the search for an orthonormal base to the Kyrlov subspace; a procedure that needs to optimize the Rayleigh quotient in its operation [10]. As we can see in the Algorithm 1, each step $i \to i+1$ requires about $5N$ scalar multiplications and one multiplication of the matrix $\mathcal{H}$ with a vector which implies N^2 aritmethic operations. The complexity in its implementation is $O(N^3) + 5O(N^2)$ where the dominant operation is give by the matrix-vector multiplication $\mathcal{H}\mathbf{w}$. It is also important that it is the unique operation inside the loop where the matrix $\mathcal{H}$ is used. Observe that the matrix remains unchanged along this loop. This leads with the well known property of the Lanczos algorithm: the sparsity of the matrix is preserved.

Algorithm 1 The Lanczos Method

```
procedure LANCZOS(H,q)
    w := [w_1 ··· w_N]^T := q;                          ▷ q satisfies ||q|| = 1
    v = [v_1 ··· v_N]^T := 0;
    γ_1 := 0; i := 1;
    while γ_i ≠ 0 do
        if i ≠ 1 then
            for k := 1, N do
                t := w_k; w_k = v_k/γ_i; v_k := −γ_i t;
            end for
        end if
        v := v + Hw;
        δ_i = w^T v;
        v := v − δ_i w;
        m := i; i := i + 1;
        γ_i = √(v^T v);
    end while
end procedure
```

We point out that the bottleneck is in the product $\mathcal{H}\mathbf{w}$, no matter whether we reorthogonalize or not. That product is always $O(N^2)$, and uses most of the computing time. The rest of the algorithm executes this product a number of times which is proportional to N; this means that the product $\mathcal{H}\mathbf{w}$ is totally responsible for the computational complexity of the algorithm. This property is the milestone of our proposed algorithm.

The limitation of the Lanczos method is due to the use of float-point arithmetic because the orthonormal basis of the Kyrlov subspace can be lose its orthogonal property [28]. Then "Ghost eigenvalues" commonly appear and it can be solved in by using reorthogonalization methods that are widely documented, as in [28, 31]. Since the reorthogonalization procedure does not affect the matrix $\mathcal{H}$, the structure of the algorithm unaltered and the final complexity of the algorithm remains unchanged.

3. An algorithm to compute $\mathcal{H}\mathbf{w}$

The key to our method is that, as commented before, the tight–binding Hamiltonian matrix is internally organized as a block matrix, where a n_{ij} block represents the interactions between the orbitals of the i atom and the j atom.

We have two physical facts that allow us to simplify the operation, which are needed for the proposed optimizations. The first is that over a predetermined cut distance, the effect between the orbitals will be zero, so the related block will also be zero. The second one is that below a predetermined nearest-limit distance, the atom cluster does not make physical sense: we cannot put two atoms as near as we want to, because in that case the cluster won't make physical sense.

We recall that we have described the tight–binding Hamiltonian matrix as $\mathcal{H} = \{\mathcal{H}_{i,j}\}$, where $i, j \in \{1, 2, \ldots, N_{at}\}$ and $\mathcal{H}_{i,j}$ is a $n_{orb} \times n_{orb}$–matrix, where $n_{orb} \ll N_{at}$. Moreover, for each $i \neq j$, the block $\mathcal{H}_{i,j} = \mathcal{O}$ if and only if the distance between atom i and atom j is greater or equal to d^*_{atom}. This d^*_{atom} is obtained from the work of K. Laasonen and R. N. Nieminen [15]. They define the cut distance as:

$$f_{\text{cut}}(d) = \frac{1}{e^{\frac{\|d - r_{\text{cut}}\|}{\Delta}} + 1}, \tag{1}$$

and it represents the distance of the first neighbors and the second neighbors of the crystallized material. Note that the diagonal block elements $\mathcal{H}_{i,i}$ are $n_{orb} \times n_{orb}$-diagonal matrices for $i = 1, 2, \ldots, N_{at}$.

Those facts allow us to use a variation of Compressed Sparse Row format. This variation have two elements: a list of non-zero blocks, and a list of jumps. The block is fixed-size, and its width and height is the number of Löwdin orbitals per atom.

When two atoms are far enough -the distance between them is greater or equal to d^*_{atom}-, all the elements of the block that models their relationship are zero. When two atoms are close enough, all the elements on that block are different to zero. Recall that we model the Hamiltonian as a matrix of blocks, where the i, j block models the relationship between i atom and j. Then, for a given i block row, there will the same number of blocks that atoms can be found on a distance below d^*_{atom} from i atom. This number is low, in fact: using a hexagonal closed packaging -the regular arrangement of identical spheres so that they take up the greatest possible fraction of an infinite 3-dimensional space -so they are packed as densely as possible-[5], with nearest atoms at 1.8 Å, the number of neighbors is 56 on its worst case; so 65 neighborhoods can be a safe worst-case value. That results allow, known the number of atoms of the cluster, store the blocks on an array without using dynamic structures like lists, which are slower to handle. The key point here is that only non-diagonal non-zero block are stored; and they are stored sequentially, first by row; so we can safely suppose that the stored number of blocks is under $N \times 65$.

An special consideration have to be done about the i, i blocks -the block diagonal-. Each one of those blocks is itself a diagonal matrix that it does not depend on the spatial configuration of the cluster, but on the atom itself. The Hamiltonian matrix shows that all the i and all j have at least one non-zero block, i, i, which is a diagonal matrix. So the diagonal block does not need storage, and the part of the product of the block diagonal can be computed on initialization of the accumulators of $\mathcal{H}\mathbf{w}$ operation.

The second array has the key of the operation. The k position of the jump array means the relationship between the k block stored, and the $k + 1$ block stored. A number d below $4 \times n_{orb} \times N$ means that the stored $k + 1$ block is d positions shifted right to the end of the non-blocked Hamiltonian; where the shift is in scalar positions on the Hamiltonian, not in blocks. So a 0 in k position means that k and $k + 1$ blocks are adjacent on matrix; 4 in k position means that k and $k+1$ blocks have a zero block between them, and so on. A number d over $\times n_{orb} \times N$ means "new row". That means that $\mathcal{H}\mathbf{w}$ is as simple as running

along H calculating at the same time four rows of $\mathcal{H}\mathbf{w}$. For a particular k block, 16 products are calculated at the same iteration, and its results are stored on 4 accumulators. Finished that operation, the k position of the jump array is used to increment the reference over w.

This allows that there are computed the minimal number of needed products, and that the structure is not iterated needlessly.

In a formal description of the algorithm to compute $\mathcal{H}\mathbf{w}$ we will use the following notation. For each $i = 1, 2, \ldots, N_{at}$ we define the set

$$\mathcal{Z}_i = \{j \in \{1, 2, \ldots, N_{at}\}, j \neq i : \mathcal{H}_{i,j} \neq \mathcal{O}\},$$

and we take

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_1^T & \mathbf{w}_2^T & \cdots & \mathbf{w}_{N_{at}}^T \end{pmatrix}^T \in \mathbb{R}^N,$$

where $\mathbf{w}_j \in \mathbb{R}^{n_{orb}}$. Then $\mathcal{H}\mathbf{w} =$

$$\begin{pmatrix} \left(\sum_{j=1}^{N_{at}} \mathcal{H}_{1,j}\mathbf{w}_j\right)^T & \cdots & \left(\sum_{j=1}^{N_{at}} \mathcal{H}_{N_{at},j}\mathbf{w}_j\right)^T \end{pmatrix}^T.$$

Clearly, this matrix-vector product uses $N_{at} \times ({n_{orb}}^2 \times N_{at}) = N^2$ operations and does not take advantage of the zero blocks and their location.

Before starting to run Algorithm 1, we need to fill the matrix $\mathcal{H}$. Our strategy will be to efficiently codify the information about the non-diagonal non-zero blocks: there is no reason to store and operate on a huge numbers of zeros. To this end we introduce the following notation.

For each $i = 1, 2, \ldots, N_{at}$ we denote by $n_i = \text{Card } \mathcal{Z}_i$ and set

$$n^* = \max_{1 \leq i \leq N_{at}} n_i.$$

Moreover, by means to the physical properties of the problem $n^* \ll N_{at}$. Now, we denote by π_i, the unique strictly increasing map from $\{1, 2, \ldots, n_i\}$ to $\mathcal{Z}_i$. Then, by using these maps, we can write each component of the matrix–vector product as

$$\sum_{j=1}^{N_{at}} \mathcal{H}_{i,j}\mathbf{w}_j = \sum_{j=1}^{N_{at}} \mathcal{H}_{j,j}\mathbf{w}_j + \sum_{j=1}^{n_i} \mathcal{H}_{i,\pi_i(j)}\mathbf{w}_{\pi_i(j)}$$

for $i = 1, 2, \ldots, N_{at}$. We remark that the number of operations in the matrix-vector product using the maps π_i is equal to

$$N + \sum_{i=1}^{N_{at}} ({n_{orb}}^2 \times n_i) \leq N + N_{at}\left({n_{orb}}^2 \times n^*\right) \tag{2}$$

$$= N(1 + (n_{orb} \times n^*)) \tag{3}$$

where

$$1 + (n_{orb} \times n^*) \ll N.$$

Thus, the matrix–vector product algorithm using $\pi = (\pi_1, \pi_2, \ldots \pi_{N_{at}})$ has a complexity of $O(N)$. In our implementation, we use the pointer arithmetic of the C language to introduce π in our matrix–vector product algorithm.

A standard Lanczos process has a complexity of $O(N^3) + 4O(N)$ in time and $O(N^2)$ in space to compute the eigenvalues. With this proposed algorithm, the complexity would be $O(N^2) + 4O(N)$ in time and $O(N)$ in space. The multiplication constants of complexity are values that depend on the average quantity of atoms on the sphere of influence of any atom: atoms farther than this distance are not taken into account, so the block caused by the crossed interaction between the current atom and one further away is zeroed. This value is related to the cut distance -limited by (1)-, and the ratio of atoms on the surface of the clusters -in most of the clusters the majority of atoms are on the surface of the cluster, as shown in [12]-. It is important to remark that by means of expression (3) and the final implementation we can obtain an upper limit for each ratio.

4. A pseudo-code for $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$

We will show a particular implementation of the algorithm here, but the fact that does not use anymore data structure than arrays allows to implement it on languages like Fortran 77.

The variables used to store $\mathcal{H}$, **w** matrix, and **v**-including **v** the result of the operation are given by:

```
struct structAtom
{
  float x;
  float y;
  float z;
};

typedef struct structAtom tcluster;

tcluster cluster[_num_matrix_atoms];

float *Hbase;
unsigned long int *OcupBase;
float *vBase;
float *wBase;
```

Now we specialize to make the $\mathcal{H}\mathbf{w}$ product for the particular case of Silicon clusters. Note that it also works, with some changes, with any cluster of atoms with four Löwdin orbitals per atom changing the diagonal constants and the calculation of the block; but it can easily be rewritten for any particular number of Löwdin orbitals per atoms. Some optimizations which improve the performance of the algorithm are the following:

- The temporal values of $\mathbf{v}$ are accumulated on Ss, Sx, Sy, and Sz.
- Each diagonal block is considered zero on the algorithm because it is precomputed and stored in Ss,Sx, Sy, and Sz.
- The loop is organized to compute correctly any sequence of empty blocks lines, without exceptions or need of any special cases.

  The proposed algorithm calculates the full core Lanczos operation $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$. Moreover, it is enough to clean $\mathbf{v}$ before calling this algorithm, if only $\mathcal{H}\mathbf{w}$ is needed.

```
  float Ss, Sx, Sy, Sz;
  float *H, *v, *w;
  unsigned long int *ocup;
  unsigned long int diagonal;
  unsigned long int index;


/* Initiation of the algorithm */
  Ss = Sx = Sy = Sz = 0;

  diagonal = 0;

/*  Let's suppose that Abase, OcupBase and vBase */
/* have the matrix on the correct format         */
  H = Hbase;
  ocup = OcupBase;
  v = vBase;

/* _num_matrix_atoms is the N of the demonstration */

  for (index = 0; index < _num_matrix_atoms; index++)
    {
      /*  As far as the central point of the diagonal     */
      /* is always computed of the same way, and it is    */
      /* not stored on the matrix, we used a pre-computed */
      /* element, and store it on the accumulators.       */

      /* First operation is on the diagonal element */
      /* It is the energy contribution of the       */
      /* relationship of the atom with itself.      */
      /* _ORBITAL_something is a physical constant. */

      Ss = _ORBITAL_ss * (wBase[diagonal]);
      diagonal++;
```

```
      Sx = _ORBITAL_xx * (wBase[diagonal]);
      diagonal++;
      Sy = _ORBITAL_yy * (wBase[diagonal]);
      diagonal++;
      Sz = _ORBITAL_zz * (wBase[diagonal]);
      diagonal++;

      /* We inicialize w. */
      w = wBase;

      /* We compute for each non-zeroed block; (*ocup)  */
      /* has the shift to the next non-zeroed block. So */
      /* far it has no sense a jump greater than        */
      /* (4*_num_matrix_atoms)+1, we will take a number */
      /* greater than 4*_num_matrix_atoms+1 as the      */
      /* ``end of line''                                */

      while ((*ocup) < (4 * _num_matrix_atoms + 1))
        {
          w += (*ocup);
        /* Here we make the partial operation H * block */
        /*  Compute the Ss accumulator */
          Ss +=
            (*(H++)) * (*w) + (*(H++)) * (*w) + (*(H++)) * (*w) +
            (*(H++)) * (*w);
          w++;
        /*  Compute the Sx accumulator */
          Sx +=
            (*(H++)) * (*w) + (*(H++)) * (*w) + (*(H++)) * (*w) +
            (*(H++)) * (*w);
          w++;
        /*  Compute the Sy accumulator */
          Sy +=
            (*(H++)) * (*w) + (*(H++)) * (*w) + (*(H++)) * (*w) +
            (*(H++)) * (*w);
          w++;
        /*  Compute the Sz accumulator */
          Sz +=
            (*(H++)) * (*w) + (*(H++)) * (*w) + (*(H++)) * (*w) +
            (*(H++)) * (*w);
          /* Next one */
          ocup++;
        }

/* When we reach here, the line has ended. */
```

```
/* Ok; Ss, Sx, Sy and Sz have accumulated */
/* the right value of v                   */
      *v = Ss;
      v++;
      *v = Sx;
      v++;
      *v = Sy;
      v++;
      *v = Sz;
      v++;
/* And we prepare the next iteration */
    };
```

4.1. Filling the matrix

The process of fill the matrix is $O(N_{at}^2)$. Let $atomX[i]$ be the X coordinate of the i-th atom, $atomY[i]$ the Y coordinate of the i-th atom, and $atomZ[i]$ the Z coordinate of the i-th atom. The basic algorithm to fill the matrix is:

```
float *H, *v, *w;
 unsigned long int *ocup;
 unsigned long int i, j;
 float deltaX, deltaY, deltaZ;
 unsigned long int temp_dist;
 float distance2;


 H = Hbase;
 ocup = OcupBase;

 for (i = 0; i < _num_matrix_atoms; i++)
   {
     temp_dist = 0;
     for (j = 0; j < _num_matrix_atoms; j++)
       {
         if (i != j)
           {
             deltaX = (molecula[i].x - molecula[j].x);
             deltaY = (molecula[i].y - molecula[j].y);
             deltaZ = (molecula[i].z - molecula[j].z);
             distance2 = deltaX * deltaX +
                    deltaY * deltaY + deltaZ * deltaZ;
             if (distance2 > _cutoff_distance_2)
               {
                 temp_dist += 4;
```

```
            }
          else
            {
              *ocup = temp_dist;
              temp_dist = 0;
              ocup++;
                /* Here the block is computed, and
                  stored in H[0],H[1]... H[16] */

               /* End of H[0],H[1]... H[16] computation */
              H+=16;
            }
        }
    }
 /* Next block line */
  *ocup = 4 * (_num_matrix_atoms) + 2;
  ocup++;
}
```

The expressions for the block matrix elements $H[0]$,$H[1]$... $H[16]$ can be obtained from [4].

5. A case study: Silicon clusters

5.1. The Silicon algorithm and constants

This case study will be the Tight-Binding Hamiltonian for Silicon clusters with the orthogonal Tight-Binding parameterization defined by J. D. Chadi in [4], and the parameterized integrals of Slater–Koster as described in [32]. The cut equation used was that described in [15], and its constants were a distance of 3.83 Å, for which was used a cut value r_{cut} of 3.3 Å, and Δ as 0.2 Å on (1) expression. [16] proved that the results of the simulations are robust against short variations in these values. In any other atoms the value of the distance will change, but the order of magnitude will remain similar.

5.2. Numerical results

The described algorithm, developed in C language using a static structure reserved on the heap, was executed on a laptop with AMD Turion 64 at 1.5GHz, 512 KB of cache and 512MB of RAM under Linux operating system. The kernel used has not multicore built-in, and 32 bit-only processor support.

A number of core Lanczos operations $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$ where executed under an artificially generated worst-case scenario of a Silicon spherical cluster at 1.8Å minimal distance using a hexagonal closed packaging, with the algorithm described in this paper.

The results can be seen in Figures, 5, 6 and 7. In each graphic, the x-coordinate is the number of atoms of the Hamiltonian, and the y-coordinate

is the computation time in seconds of the indicated number of $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$ operations.

As the figures show, the computation time of $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$ scales linearly according to the increase in the number of atoms of the Hamiltonian -ergo the size of $\mathcal{H}$ matrix-. The slope of the computation time curve against the number of atoms is not high, so the proposed algorithm shows excellent behavior between 10 and 2500 atoms. With such a fast $\mathbf{v} := \mathbf{v} + \mathcal{H}\mathbf{w}$ computation (ie, 0.01403 seconds for each iteration of a $10^4 \times 10^4$ matrix), it is feasible to use reorthogonalization techniques on huge Hamiltonians in a reasonable time.

The time of the computation for more than 25000 atoms has a linear evolution; but it does not have the linear behavior that it has between 10 and 25000 atoms. The main reason is that for more than 25000 atoms the set does not fit into the cache of the processor. The slope gets worse once the structure does not fit on the memory cache, but the computation time behaves well and allows the use of the algorithm on higher matrices.

6. Conclusions

Our results show that the proposed algorithm reduces the order of complexity of tight–binding Hamiltonian matrices tridiagonalization from a cubic scaling to a quadratic one. It also allows a better storage of the matrix and a more efficient use of memory and cache.

The proposed matrix–vector product can be applied to any Lanczos variation, with or without reorthogonalization. It allows us to work with clusters several orders of magnitude greater, whose energy can be accurately calculated using tight-binding, replacing the pair-potential methods, which are less-precise and more prone to errors.

[1] Alavi, A., and Frenkel, D. Grand-canonical simulations of solvated ideal fermions. evidence for phase separation. *The Journal of Chemical Physics 97*, 12 (December 1992), 9249–9257.

[2] annd Ernst Richter, M. M., and Subbaswamy, K. R. Structural and vibrational properties of fullerenes and nanotubes in a nonorthogonal tight-binding scheme. *Journal of Chemical Physics 104*, 15 (April 1996), 5875–5882.

[3] Aoki, M. Rapidly convergent bond order expansion for atomistic simulations. *Physical Review Letters 71*, 23 (December 1993), 3842–3845.

[4] Chadi, D. Theoretical study of the atomic structure of silicon (211), (311), and (331) surfaces. *Physical review B 29*, 2 (January 1984), 785–792.

[5] Conway, J. H., and Sloane, N. J. A. *Sphere Packings, Lattices, and Groups, 2nd ed.* Springer-Verlag, 1993.

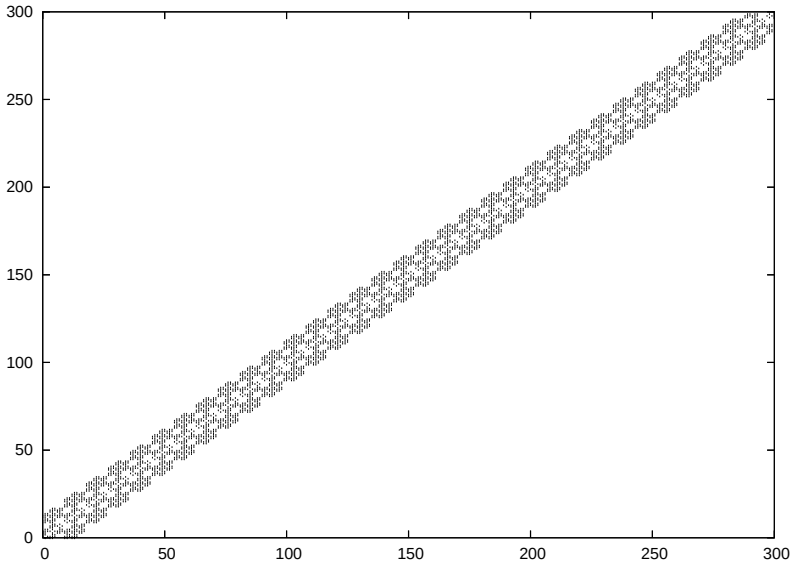

Figure 1: The sparsity pattern of a prolate 300 atom Si cluster at 2.1Å minimal distance with a hexagonal closed packaging.

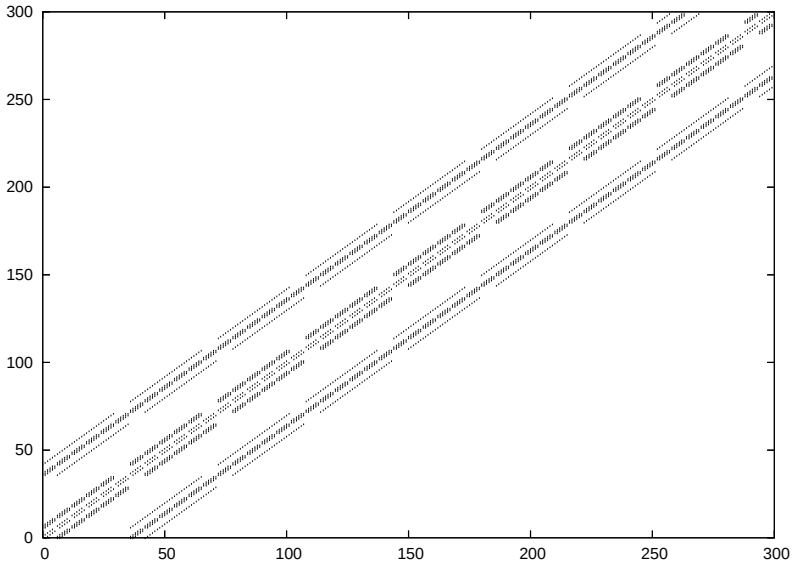

Figure 2: The sparsity pattern of a spherical 300 atom Si cluster at 2.1Å minimal distance with a hexagonal closed packaging.

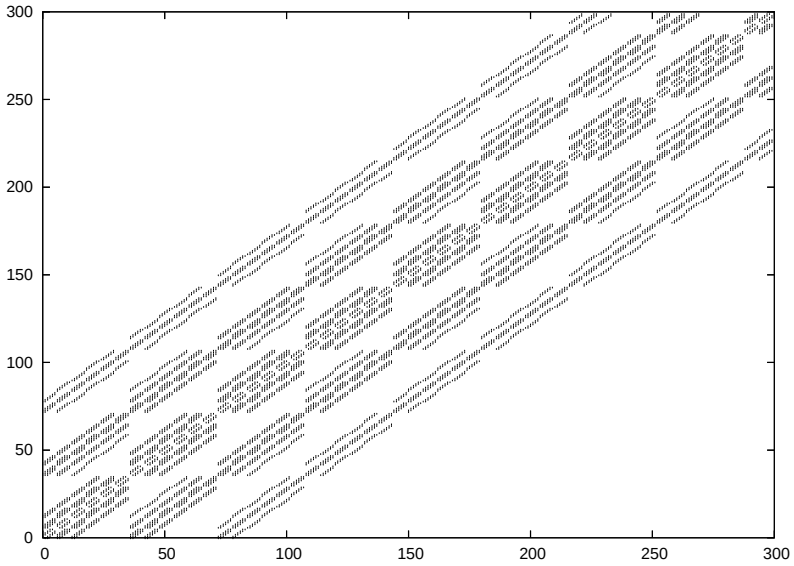

Figure 3: The sparsity pattern of a spherical 300 atom Si cluster at 1.8Å minimal distance with a hexagonal closed packaging. It was used as a worst-case scenario in this paper.

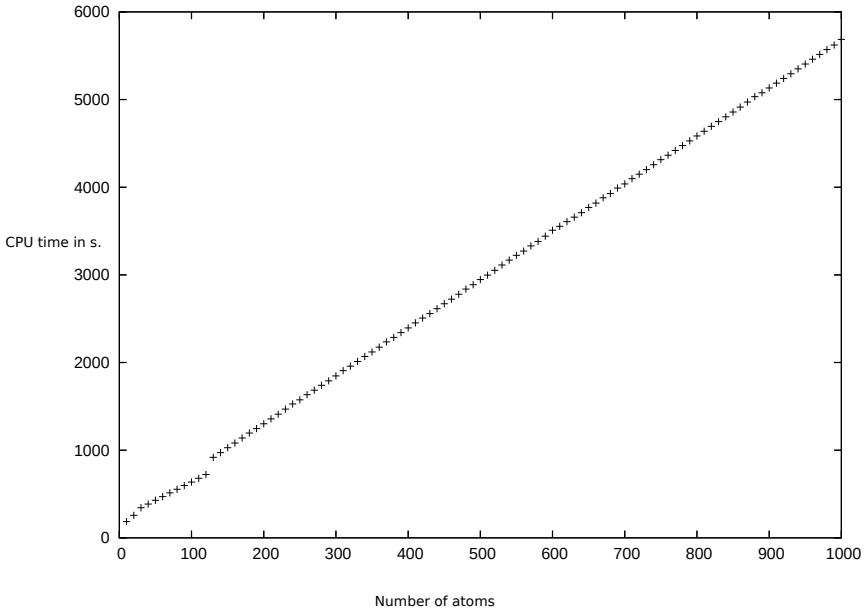

Figure 4: A performance of 10^7 core Lanczos operations -multiplication and addition-, between 10 and 1000 atoms.

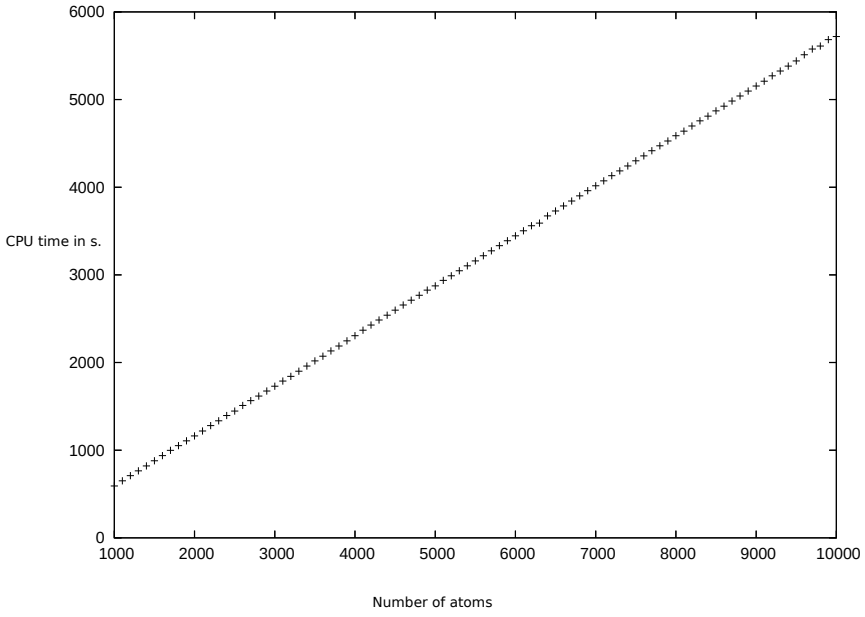

Figure 5: A performance of 10^6 core Lanczos operations -multiplication and addition-, between 100 and 10^4 atoms.

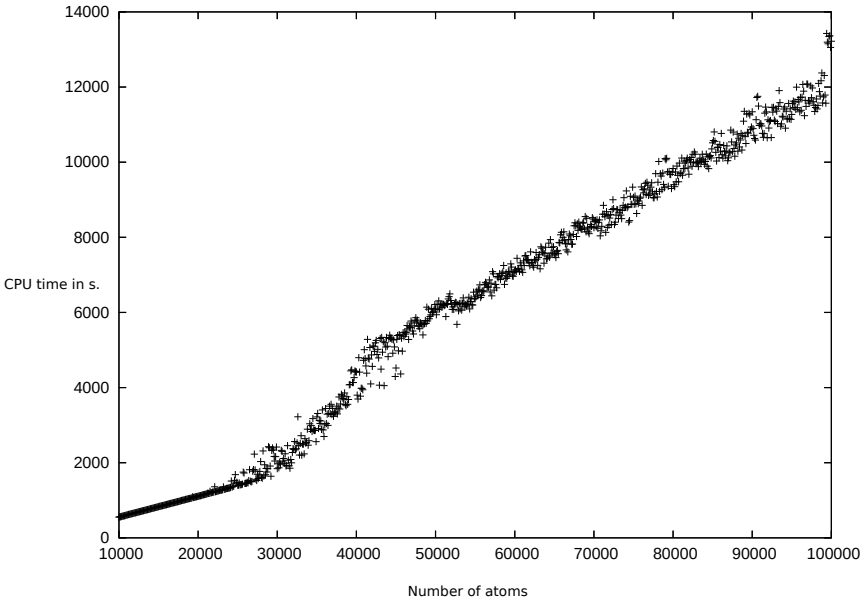

Figure 6: A performance of 10^5 core Lanczos operations -multiplication and addition-, between 10^4 and 10^5 atoms.

[6] Cuthill, E., and McKee, J. Reducing the bandwidth of sparse symmetric matrices. *Proceedings of the 24th Nat. Conf. Assoc. Comp. Mach.* (1969), 157 – 172.

[7] Daw, M. S. Model for energetics of solids based on the density matrix. *Physical Review B 47*, 16 (April 1993), 10895–10898.

[8] Feuston, B. P., Kalia, R. K., and Vashishta, P. Fragmentation of silicon microclusters: a molecular-dynamics study. *Physical Review B 35*, 12 (April 1987), 6222–6239.

[9] Galli, G., and Parrinello, M. Large scale electronic structure calculations. *Physical Review Letters 69*, 24 (December 1992), 3547–3550.

[10] Golub, G. H., and Loan, C. F. V. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996.

[11] Goodwin, L., Skinner, A., and Pettifor, D. Generating transferable tight-binding parameters: application to silicon. *Europhysics Letters 9* (August 1989), 701.

[12] Kaxiras, E., and Jackson, K. Shape of small silicon clusters. *Physical Review Letters 71*, 5 (1993), 727–730.

[13] Kohn, W. Density functional/wannier function theory for systems of very many atoms. *Chemical Physics Letters 208*, 4 (1993), 167–172.

[14] Kohn, W. Density functional and density matrix method scaling linearly with the number of atoms. *Physical Review Letters 76*, 17 (1996), 3168–3171.

[15] Laasonen, K., and Nieminen, R. Molecular dynamics using the tight-binding approximation. *Journal of Physics: Condensed Matter 2* (1990), 1509–1520.

[16] Lemes, M. R. *Estudo do estado fundamental de aglomerados de Silicio via redes neurais*. PhD thesis, Instituto Tecnológico de Aeronáutica - ITA, 1995.

[17] Li, X. P., Nunes, R. W., and Vanderbilt, D. Density-matrix electronic-structure method with linear system-size scaling. *Physical Review B 47*, 16 (April 1993), 10891–10894.

[18] Mauri, F., Galli, G., and Car, R. Orbital formulation for electronic-structure calculations with linear system-size scaling. *Physical Review B 47*, 15 (April 1993), 9973–9976.

[19] Menon, M., and Subbaswamy, K. R. Universal parameter tight-binding molecular dynamics: Application to c_{60}. *Physical Review Letters 67*, 25 (December 1991), 3487–3490.

[20] Menon, M., and Subbaswamy, K. R. Nonorthogonal tight-binding molecular-dynamics study of silicon clusters. *Physical Review B 47*, 19 (May 1993), 12754–12759.

[21] Menon, M., and Subbaswamy, K. R. Structure of Si_{60}. Cage versus network structures. *Chemical Physics Letters 219* (March 1994), 219–222.

[22] Menon, M., Subbaswamy, K. R., and Sawtarie, M. Structure of c_{20}: Bicyclic ring versus cage. *Physical review B 49*, 11 (March 1993), 7739–7743.

[23] Menon, M., Subbaswamy, K. R., and Sawtarie, M. Structure and properties of c_{60} dimers by generalized tight-binding molecular dynamics. *Physical Review B 49*, 19 (May 1994), 13966–13969.

[24] Min, B. J., Lee, Y. H., Wang, C., Chan, C. T., and Ho, K. M. Tight-binding model for hydrogen-silicon interactions. *Physical Review B 45*, 12 (1992), 6839–6843.

[25] Molteni, C., Colombo, L., and Miglio, L. Structural properties of liquid and amorphous GaAs by tight-binding molecular dynamics. *Europhysics Letters 24* (December 1993), 659.

[26] Ordejón, P., Drabold, D. A., Grumbach, M. P., and Martin, R. M. Unconstrained minimization approach for electronic computations that scales linearly with system size. *Physical Review B 50*, 19 (November 1994), 14646–14649.

[27] Ordejón, P., Lebedenko, D., and Menon, M. Improved nonorthogonal tight-binding hamiltonian for molecular-dynamics simulations of silicon clusters. *Physical Review B 50*, 8 (August 1994), 5645–5650.

[28] Paige, C. Practical use of the symmetric lanczos process with reorthogonalization. *BIT 10, 183-195* (1976).

[29] Pan, J., Bahel, A., and Ramakrishna, M. V. Chemistry of nanoscale semiconductor clusters. *Nanomeeting–95* (1997), 1–7.

[30] Saad, Y. *Iterative methods for sparse linear systems.* PWS Publisher Company, 1996.

[31] Simon, H. Analysis of the symmetric lanczos algorithm with reorthogonalization methods. *Linear algebra and its applications* (1984), 101–132.

[32] Slater, J., and Koster, G. Simplified lcao method for the periodic potential problem. *Physical Review 94*, 6 (June 1954), 1498–1524.

[33] Yang, W. Direct calculation of electron density in density-functional theory. *Physical Review Letters 66*, 11 (1991), 1438–1441.