Increasing the locality of iterative methods and its application to the simulation of semiconductor devices

J. C. Pichel[1], D. B. Heras[2], J. C. Cabaleiro[2], A. J. García-Loureiro[2], and F. F. Rivera[2]

[1] Galicia Supercomputing Center (CESGA), Spain
jcpichel@cesga.es
[2] Electronics and Computing Dpt.
Universidade de Santiago de Compostela, Spain
dora,caba,antonio,fran@dec.usc.es

Abstract. Irregular codes are present in many scientific applications, such as finite element simulations. In these simulations the solution of large sparse linear equation systems is required, which are often solved using iterative methods. The main kernel of the iterative methods is the sparse matrix–vector multiplication which frequently demands irregular data accesses. Therefore, techniques that increase the performance of this operation will have a great impact on the global performance of the iterative method and, as a consequence, on the simulations.
In this paper a technique for improving the locality of sparse matrix codes is presented. The technique consists of reorganizing the data guided by a locality model instead of restructuring the code or changing the sparse matrix storage format. We have applied our proposal to different iterative methods provided by two standard numerical libraries. Results show an impact on the overall performance of the considered iterative method due to the increase in the locality of the sparse matrix–vector product. Noticeable reductions in the execution time have been achieved both in sequential and in parallel executions.
This positive behavior allows the reordering technique to be successfully applied to real problems. We have focused on the simulation of semiconductor devices and in particular on the BIPS3D simulator. The technique was integrated into the simulator. Both sequential and parallel executions have been extensively analyzed in this paper. Noticeable reductions in the execution time required by the simulations are observed when using our reordered matrices in comparison with the original simulator.

Keywords: irregular codes, reordering techniques, data locality, iterative methods, semiconductor devices simulation.

1 Introduction

As a result of the important increase in the performance of the processors over the years, the gap between processors and memory has been widened. This fact causes a gradual imbalance between both factors, which nowadays leads to the latency in the memory accesses becoming one of the main bottlenecks in the performance of a computer. Therefore, it is necessary to develop techniques that minimize this effect, exploiting the memory hierarchy in a more efficient way. This fact is even more important in the case of irregular codes. These codes, due to their unpredictable behavior in the accesses generated, tend to have low spatial and temporal locality.

Irregular codes are present in many scientific applications, such as finite element simulations. In these simulations the solution of large sparse linear equation systems is required, which are often solved using iterative methods [20]. The main kernel of the iterative methods is the sparse matrix–vector multiplication (SpM$\times$V), which frequently demands irregular data accesses. Therefore, techniques that increase the performance of this operation will have a great impact on the global performance of the iterative method and, as a consequence, on the simulations. In this paper, a technique to deal with the problem of increasing the locality of irregular codes as the sparse matrix–vector product is proposed.

The technique that we have developed is based on a locality model presented in previous works by the authors [8], and extensively tested on different uniprocessor, shared and distributed memory multiprocessor architectures [15, 16]. The technique consists of reorganizing the data instead of restructuring the code or changing the sparse matrix storage format. The goal is to increase the grouping of elements in the pattern of the sparse matrix that characterizes the irregular accesses, and thereby to improve the locality in the execution of the code. This reordering is effective for matrices with any structure and for different sparse algebra kernels.

In this work the locality optimization technique has been applied to different iterative methods provided by two numerical libraries, in particular, to SPARSKIT2 [19] sequential library and to PSPARSLIB [21] parallel one. In order to demonstrate the benefits of our proposal, a comparison with METIS library [11] reorderings is provided. Nowadays, METIS is considered a standard in terms of graph partitioning and reordering. Moreover, the proposed reordered technique has been integrated into a parallel 3D simulator of semiconductor devices. In particular, the BIPS3D simulator [7] is studied in this work. The performance of the original simulator, considering sequential and parallel executions, has been compared with the performance achieved using the reordered sparse matrices obtained after applying our technique.

The paper is organized as follows: in Section 2 the related work is discussed. The main characteristics of the BIPS3D simulator and some relevant aspects of the numerical libraries considered in this work are commented in Section 3. The locality properties of the sparse matrix–vector product are detailed in Section 4. Section 5 presents a brief summary of the locality model which the technique presented in this paper is based on, and it also includes the description of the locality improvement technique. In Section 6 the experimental conditions are established, results are displayed and the benefits of applying our technique are discussed. The paper finishes with the main conclusions derived from this work.

2 Related work

The techniques for increasing the locality of irregular codes can be mainly divided into two groups: data reordering techniques and code restructuring techniques.

Standard reordering techniques are considered classical methods for dealing with the problem of increasing the locality in the execution of an irregular code. The most used techniques are the bandwidth reduction algorithms, which derive from the Cuthill-McKee algorithm [5]. In a recent work [17], the authors have also demonstrated the

benefits of using *minimum degree*-based heuristics such as the *approximate minimum degree* algorithm [1] for increasing the locality of this kind of codes on multicore processors. Oliker *et al.* [14] show the benefits that are offered by the application of some of these reordering algorithms to sparse codes when executed on different multiprocessor architectures. One of their main conclusions is that the cache implications of ordering are more important than the communication implications of partitioning. Note that, unlike standard reordering algorithms, the main objective of our technique is to increase the data reuse and, as a consequence, the data locality. In a more recent work, Coutinho *et al.* [4] perform a comparison of different data reordering algorithms for the SpM$\times$V in edge-based unstructured grid computations. However, they only focus on serial executions.

Those techniques based on restructuring the code, like *blocking* or *tiling*, that are broadly used in regular codes, have been successfully applied to different irregular codes such as the product of a sparse matrix by a dense matrix [6, 13] and stationary iterative methods [22]. Im *et al.* [10] propose register and cache blocking as optimization techniques for the SpM$\times$V. In other paper [3], a performance model for the blocked SpM$\times$V is presented, which allows to pick in nearly all cases the actual optimal blocksize. Vuduc *et al.* [24] extend the notion of blocking in order to exploit variable block shapes by decomposing the original matrix to a proper sum of submatrices storing each submatrix in a variation of the blocked CSR format. One of the main drawbacks of these techniques is the strong dependence with the sparsity pattern of the matrix. For example, register blocking only achieves good performance for matrices with small dense-blocks in the pattern. Unlike these solutions, our locality optimization technique is effective for matrices with any pattern [15, 16].

Some authors have demonstrated that both groups of techniques are complementary in the sense that when they are applied together the locality is increased. In particular, Toledo [23] evaluates different standard reordering techniques and combines them with blocking, showing that performance increases significantly depending on the size and sparseness of the considered matrix. Pinar and Heath [18] introduce a reordering technique that favors the creation of dense blocks on the pattern of the sparse matrix, and in this way the efficiency of the blocking technique proposed by Toledo is increased. Moreover, a comparison between their reordering technique and some standard reordering techniques is carried out. In another work [9], a combination of data reordering algorithms and register blocking has been applied to the SpM$\times$V on shared memory multiprocessors, finding little benefit.

The locality improvement technique used in the present paper can also be applied to codes where data are stored using a blocked scheme. An example was published in [16] where a reordering of the sparse matrix in combination with blocking techniques was successfully applied to the SpM$\times$V. The technique was evaluated on different uniprocessors and on distributed memory multiprocessors.

Moreover, there are several papers that deal with the SpM$\times$V optimization problem using compression. In a recent work [12], the authors propose two different compression methods targeting index and numerical values. Williams *et al.* [25] apply an index reduction technique, in which 16-byte indices are used when it is possible. In the same work the authors propose several additional optimization techniques for the SpM$\times$V,

which are evaluated on different multicore platforms. Authors examine among others: software pipelining, prefetching approaches, register and cache blocking, etc. Nevertheless, they do not consider data reordering techniques in order to increase locality.

3 BIPS3D simulator

BIPS3D is a 3D simulator of semiconductor devices. In particular it simulates the electronic behavior of Bipolar and MOSFET transistors [7]. It focuses on solving the Poisson equation for a given transistor, which implies the resolution of a system of linear equations described by a Finite Element Method mesh (FEM sparse matrix) that characterizes the simulated device. A linear system can be expressed as the following:

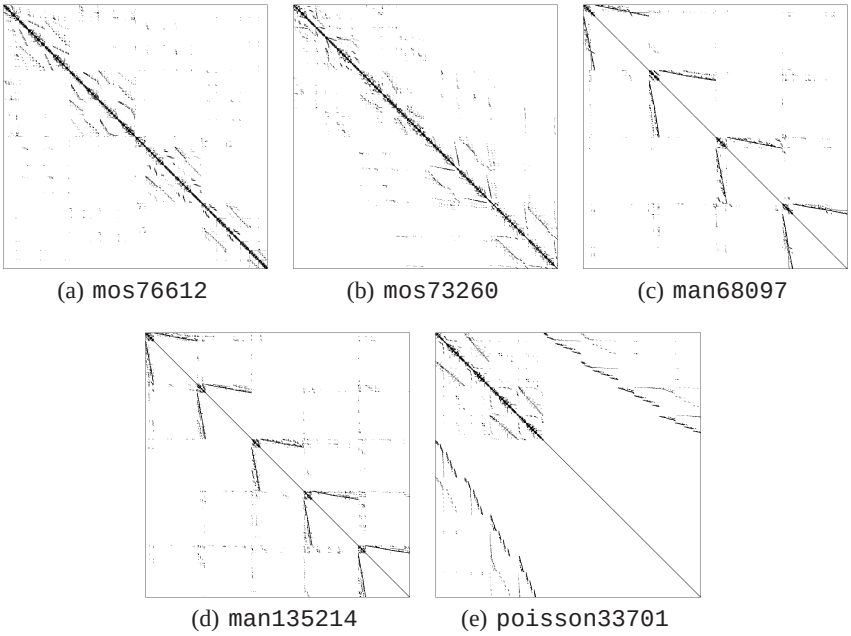
$$Ax = b \tag{1}$$

where A is a matrix, b is the right-hand side vector and x is the unknown vector to be calculated. In order to solve the systems of equations, the simulator uses several iterative methods [20], in which the sparse matrix–vector product is one of the main computational kernels. As we will discuss in the results section, an increase in the performance of this operation will have a great impact on the performance of the whole iterative method, and therefore, an important reduction in the overall time required for the simulation.

BIPS3D, like most simulation applications, consists of three stages: preprocessing, processing and postprocessing. In the preprocessing stage of the BIPS3D simulator the data of the problem are introduced, such as the geometric characteristics of the device, the values of different parameters, etc. Then the FEM mesh of the semiconductor device is generated. Figure 1 shows the sparsity patterns of the five FEM meshes considered in this work (see Section 6 for details). Next, different graph partitioning and reordering techniques are applied to the mesh. In particular, the simulator uses the techniques provided by the METIS library [11], which nowadays is considered as a standard. The main goal of the partitioning is to distribute the mesh (sparse matrix) among the processors in such a way that the time of the parallel execution is optimized. An important factor in the efficiency and robustness of the iterative methods is the *preconditioning* [20]. Preconditioning is simply a means of transforming the original linear system into one which has the same solution, but which is likely to be easier to solve by means of an iterative solver. In this way, in order to obtain this new system, a reordering technique, which is also performed using the METIS library, is applied to the mesh. However, this reordering technique is not taking into account a key factor in the performance of the iterative methods, that is, the locality properties of the sparse matrix–vector product.

The next stage of the simulation corresponds to the processing. During this phase the equations that model the device are solved (sequentially or in parallel) using an iterative method, which constitutes the core of the simulator. With this purpose two sparse algebra numerical libraries are used in the simulator: SPARSKIT2 [19] and PSPARSLIB [21], whose main characteristics are outlined below.

Finally, in the postprocessing stage the results of the simulation are analyzed and visualized.



(a) `mos76612` (b) `mos73260` (c) `man68097`

(d) `man135214` (e) `poisson33701`

Fig. 1. Patterns of several FEM matrices which describe different MOSFET transistors.

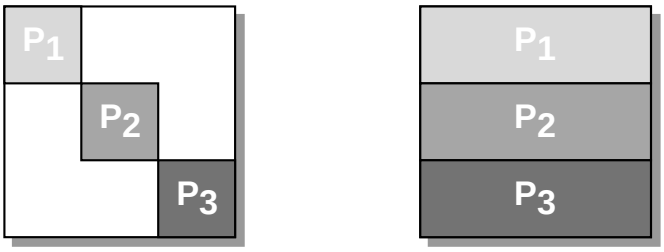
3.1 Numerical libraries

SPARSKIT2 [19] is a sequential library for the solution of sparse systems of linear equations. It is written in Fortran, and it provides several tools for working with this kind of matrices. It includes a module for the solution of linear systems that includes, among others, the following iterative methods: CG (*Conjugate Gradient*), BiCGSTAB (*BiConjugate Gradient Stabilized*) and FGMRES (*Flexible Generalized Minimum Residual*). Some available preconditioners are: ILUT (*dual Threshold ILU*), ILUTP (*ILUT with partial column Pivoting*), ILU(0) and MILU(0) (*Modified ILU*).

PSPARSLIB [21] has also been considered in this work. This is a library of parallel iterative solvers. It is mainly written in Fortran, although it includes a few C modules. The parallel code uses the MPI library. In order to solve a system of equations on a distribute memory multiprocessor, PSPARSLIB uses *domain decomposition solvers* [20]. Domain decomposition methods refer to a collection of techniques based on the divide and conquer principle. If we consider the problem of solving an equation on a domain Ω partitioned into s subdomains Ω_i, such that:

$$\Omega = \bigcup_{i=1}^{s} \Omega_i \tag{2}$$

domain decomposition methods attempt to solve the problem on the entire domain by carrying out a problem solution on each local Ω_i. In this way, the FEM mesh (sparse matrix) is partitioned, and a cluster of nodes representing a physical subdomain is assigned to each processor. Each node belonging to a subdomain is an unknown of the

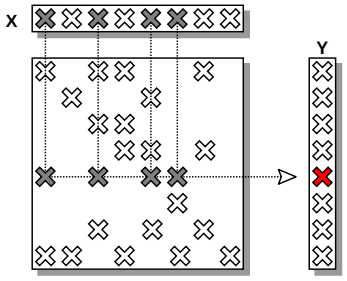


(a) (b)

Fig. 2. Examples of the data distribution of the sparse matrix using three processors on a distributed memory system: using PSPARSLIB (a) and typical by rows (b).

```
DO i = 1, N
    reg = 0
    DO j = PTR(i), PTR(i+1) - 1
        reg = reg + DA(j)·X(INDEX(j))
    END DO
    Y(i) = reg
END DO
```

(a)



(b)

Fig. 3. Code of the sparse matrix-vector product by rows (a), and accesses performed in the i-th iteration (b).

problem. The matrix assigned to a certain processor is split into two parts: the *local matrix*, which acts on the local variables and an *interface matrix*, which acts on the external variables. Figure 2 shows, on the left, an example of local matrices assigned to three processors.

PSPARSLIB provides several domain decomposition techniques such as additive Schwarz, multicolor SOR and Schur complement methods [20]. In this work we have used the additive Schwarz method, which offers good results in all the cases considered.

4 Locality properties of the SpM×V

Let's consider, for the SpM×V operation, that the matrix is stored using the well-known standard Compressed-Sparse-Row format (CSR) for unstructured sparse matrices [20]. DA, INDEX and PTR are the arrays (data, column indices and row pointer) that characterize this format. Using this format, and considering that X is the vector by which the product is performed and Y the result vector, the sequential product by rows can be implemented as displayed in Figure 3(a). N is the number of rows/columns of the matrix and N_Z is the number of entries.

When the access to the sparse matrix is row-wise there are N_Z references to the arrays X, INDEX and DA, N references to Y, and $2(N+1)$ to PTR. Therefore, the total number of references is $3N_Z + 3N + 2$. Figure 3(b) shows an example of the accesses in the i-th iteration of the SpM$\times$V operation. Every array, except X, presents sequential accesses. DA and INDEX are accessed in each iteration of loop j, while arrays Y and PTR are accessed in the iterations of the external loop i.

Arrays Y and PTR present sequential accesses with high spatial locality. In the case of Y there is no data reuse, and therefore, it does not present temporal locality in its accesses. Nevertheless, each element of PTR are reused in two consecutive iterations of loop i, except the elements PTR(1) and PTR$(N+1)$. The contribution of Y and PTR to the total number of cache misses is $\lceil N/L_S \rceil$ and $\lceil (N+1)/L_S \rceil$ respectively, being L_S the cache line size.

DA and INDEX present a high spatial locality due to the sequential accesses to their elements. The contribution of these vectors to the number of misses will be at least $\lceil N_Z/L_S \rceil$ for each one (which corresponds to the number of intrinsic misses produced by these vectors). Note that, although the misses produced by these arrays are mostly intrinsic, sometimes a small number of conflict misses could be produced due to the temporal separation in the consecutive accesses to the elements of the arrays.

Finally, array X shows unpredictable behavior due to the indirection through the INDEX array. As Figure 3(b) shows, the elements of X accessed in each i iteration depend on the irregular pattern of the sparse matrix. So, the irregular access pattern will produce conflict misses, being the maximum contribution of $\lceil N/L_S \rceil$ misses per i iteration. Note that cached lines of X can be replaced by data from other arrays that were accessed in near iterations of the outermost loop with no possibility of reuse. Therefore, it is clear that the greatest potential for locality improvement is on accesses to array X.

5 Locality improvement technique

In this section, first, the locality model in which the reordering technique is based on is presented. Next, the locality improvement technique is described for both sequential and parallel executions.

5.1 Locality model

The locality model is based on the evaluation of the data locality for the sparse algebra code considered. The model is general enough to be applied to any sparse algebra code that can take advantage of a clustering of entries in the pattern of the matrix.

In the model, locality is measured for consecutive pairs of rows or columns of the sparse matrix depending on if the prevailing irregular accesses to the sparse matrix are row-wise or column-wise. In both cases, the locality is based on two parameters: the number of *entry matches* (a_{elems}) and the number of *block matches* (a_{blocks}). For instance, considering accesses to the sparse matrix by rows, the number of entry matches between any pair of rows is defined as the number of nonzero elements in the same column of both rows. The concept of entry matches can be extended to block matches by considering, instead of single entries (an entry is defined as a nonzero element), pieces

of consecutive positions in a row of the matrix pattern of the size of a cache line where there is at least one entry.

Based on these two parameters a magnitude called *distance between rows x and y* can be defined, denoted as $d_i(x, y)$. It is used to measure the locality displayed by the irregular accesses performed by the irregular code on these two rows when they are consecutively accessed. From different distance definitions proposed in [8], in this work we consider the following:

$$d_1(x,y) = n_{\text{blocks}}(x) + n_{\text{blocks}}(y) - 2{*}a_{\text{blocks}}(x,y) \quad (3)$$
$$d_2(x,y) = n_{\text{elems}}(x) + n_{\text{elems}}(y) - 2{*}a_{\text{elems}}(x,y) \quad (4)$$

where $n_{\text{elems}}(x)$ is the number of entries in row *x*, and $n_{\text{blocks}}(x)$ is the number of groups of entries in row *x*. It can be shown that these distances define metrics.

For a given sparse matrix accessed by rows, a quantity, that is inversely proportional to the locality of the data for the whole sparse matrix, can be defined as follows:

$$D_i = \sum_{x=0}^{N-2} d_i(x, x+1), \quad i = 1 \textit{ or } 2. \quad (5)$$

These definitions can be directly extended to columns.

5.2 Windows of locality

The locality model detailed above only provide results based on the locality evaluated on pairs of consecutive rows (or columns) of the sparse matrix. Nevertheless, in the SpM×V case, for example, the reuse of data could be possible in any level of the memory hierarchy during the product of two or more consecutive rows (or columns) of the matrix, as we detailed in Section 4. For this reason a generalization of the distance functions based on the concept of *windows of locality* was proposed [8].

A *window of locality* is a set of w consecutive rows (or columns) of the matrix between which there is a high probability of data reuse when executing the sparse algebra code considered. Based on the distance functions $d_1(x, y)$ and $d_2(x, y)$, we can define the distances between windows of locality g and h as:

$$d_{w1}(g,h) = nl(g) + nl(h) - 2{*}a_{\text{blocks}}(g,h) \quad (6)$$
$$d_{w2}(g,h) = n(g) + n(h) - 2{*}a_{\text{elems}}(g,h) \quad (7)$$

where $n(g) = n_{\text{elemsw}}(g) - a_{\text{elemsw}}(g)$ and $nl(g) = n_{\text{blocksw}}(g) - a_{\text{blocksw}}(g)$. Parameter $a_{\text{elems}}(g, h)$ is a direct extension of the entry matches between windows g and h. $n_{\text{elemsw}}(g)$ is the number of elements of window g, and $a_{\text{elemsw}}(g)$ generalizes the concept of entry matches considering the matches that take place between two or more rows within window g. Note that, by introducing $n(g)$ the possible reuse of data inside g is also considered. Figure 4 shows an example of the calculation of these parameters when $w = 2$. Finally, $n_{\text{blocksw}}(g)$ and $a_{\text{blocksw}}(g)$ are defined in the same way as $n_{\text{elemsw}}(g)$ and $a_{\text{elemsw}}(g)$, although block matches are considered instead of entry matches.

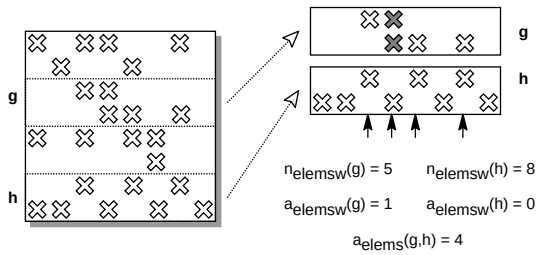


Fig. 4. Example of the calculation of $n_{\mathrm{elemsw}}(g)$, $a_{\mathrm{elemsw}}(g)$ and $a_{\mathrm{elems}}(g,h)$.

These distances are equivalent to the distances measured over pairs of consecutive rows (or columns) of the matrix when the window size is $w = 1$. In this way, the distances evaluated on windows of locality preserve the same properties. Therefore, the indirect estimation of locality defined for a sparse matrix in Equation 5 can now be calculated as a sum over the whole matrix:

$$D_{wtotal_i} = \sum_g d_{wi}(g, g+1), \quad \forall\, g \mid 0 \leq g < \lceil N/w \rceil \quad i = 1 \; or \; 2. \tag{8}$$

5.3 Sequential case

We propose a heuristic technique that modifies the pattern of the sparse matrix according to the locality model described above. In order to increase the locality in the accesses performed by the irregular code, we search for a permutation of rows (and columns) of the matrix that minimizes its total distance D_{wtotal_i} (Equation 8). Figure 5 shows a scheme of this technique. We formulate the problem of locality improvement as a classic NP-complete optimization problem, and we solve it as a graph problem, using its analogy to the traveling salesman problem (TSP). A similar approach has been used by Pinar and Heath [18], which performs a reordering to favor the application of blocking data structures.

Considering the sequential case, the problem is described using a complete weighted graph where each node represents a window of locality of the input sparse matrix. Each edge of the graph has an associated weight that reflects the distance between pairs of windows of locality according to the description of locality given previously. Nevertheless, it is not necessary to work with a complete graph. Given that sparse matrices have a very low density of nonzero elements, most of the weights in the graph correspond with cases where $a_{elems} = 0$ and $a_{blocks} = 0$. Those values, according to the distance definitions, represent the worst cases regarding locality. Therefore, without losing relevant information about the locality, we can use an incomplete weighted graph where only values of a_{elems} and a_{blocks} different from zero are considered. As a consequence, the graph size is noticeably reduced.

Solving the problem of reordering is equivalent to finding a path of minimum length that goes through all the nodes of the graph. This path is represented as a permutation vector that gives the appropriate order of the nodes of the graph, and therefore, a reordered matrix. Due to the fact that all the matrices in this work are symmetric, it is

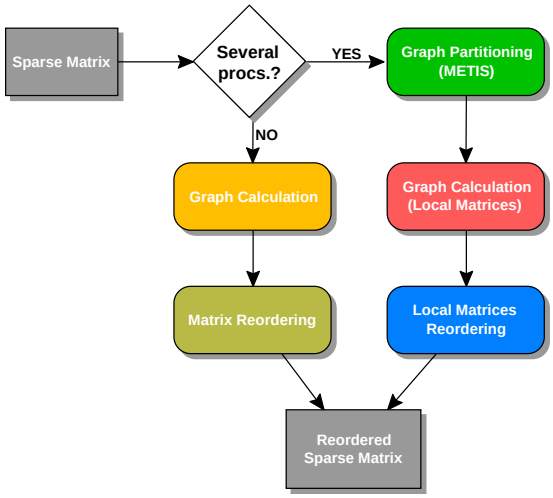


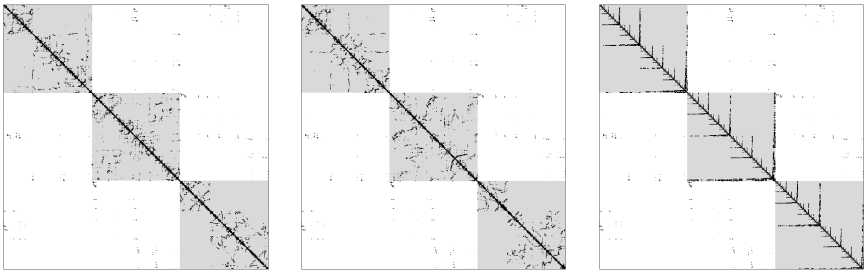
Fig. 5. Scheme of the locality optimization technique.

only necessary to calculate one graph considering rows or columns (i.e., just one permutation vector is required). On the other hand, given that we have measures (distance values) to validate the quality of an ordering, we have opted for focusing on heuristic solutions. After a comparative study of different techniques, the *Chained Lin–Kernighan* heuristic was chosen. In particular, the implementation proposed by Applegate et al. [2] was used.

5.4 Parallel case

Parallel execution is carried out on a distributed memory multiprocessor system. In a previous work a reordering technique based on the locality model described in Section 5.1 for these systems was proposed by the authors [16]. However, that technique cannot be applied directly to this case because of the particularities of the data distribution in PSPARSLIB. Figure 2 shows an example, on the right, of the typical data distribution of a sparse matrix by rows on a distributed memory system considering three processors, and on the left, when PSPARSLIB is used.

The locality optimization technique proposed when using PSPARSLIB is introduced in the scheme of Figure 5. In the first step, when several processors are available, the sparse matrix is partitioned among the processors (local matrices) using the graph partition tools provided by METIS. In this way, the load balance of the reorderings generated by METIS (which are used by the original BIPS3D simulator) and those obtained after applying our proposal, is exactly the same. Once the local matrices are known, the locality optimization process implies the minimization of the total distance for each of them. Therefore, the distance graph of each local matrix must be computed, and later, the reordering heuristic (Chained Lin-Kernighan) is applied to the graphs in order to obtain a minimum length path. In this way, a permutation vector per local matrix is obtained.



(a) d_{w1} (b) d_{w2} (c) METIS

Fig. 6. Patterns of the reorderings of matrix `mos73260` obtained using our technique and METIS for an execution with three processors in PSPARSLIB.

Table 1. Cache hierarchy of the Alpha EV68 processor.

	L1 Instr.	*L1 Data*	*L2*
Size (KBytes)	64	64	4096
Associativity	2	2	Direct mapped
Line size (Bytes)	64	64	64
Latency (Cycles)	1	2	19

As an example, the appearance of the reorderings of matrix `mos73260` for an execution with three processors when using our reordering technique (Figures 6(a) and 6(b)) and METIS (Figure 6(c)) is displayed. In this example a window size $w = 64$ was considered. Local matrices assigned to the processors are colored in grey. The other nonzero elements within the matrices correspond with interface nodes (see Section 3.1 for details).

6 Performance analysis

The results shown in this section have been obtained using a Compaq HPC320 system. It consists of eight ES45 servers, each of them with 4 processors, establishing a system with a total of 32 1GHz Alpha EV68 processors. Table 1 displays the main characteristics of the cache hierarchy of this processor.

On the other hand, the description of the FEM matrices considered in this work is showed in Table 2 (N is the number of rows/columns of the matrix and N_Z is the number of entries). In the original BIPS3D simulator, FEM meshes were reordered using the METIS library. We use these reorderings in order to compare and evaluate the performance of our technique, both in the simulator and in the considered numerical libraries. Given that the number of iterations necessary for the convergence depends on the matrix and the iterative method, the results of Sections 6.1 and 6.2 are per iteration.

In this section, first, the results obtained using SPARSKIT2 (6.1) and PSPARSLIB (6.2) libraries are shown. Next (6.3), the global performance of the BIPS3D simulator is evaluated making up all its components, including the overhead of our reordering

Table 2. Matrix benchmark suite.

Matrices	N	N_Z
mos76612	76612	946154
mos73260	73260	906344
man68097	68097	667011
man135214	135214	1327704
poisson33701	33701	372385

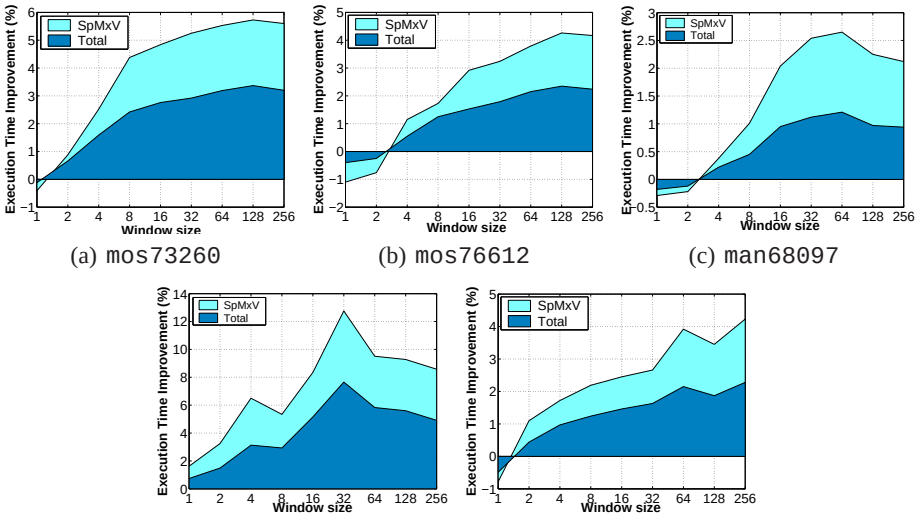
technique. In this way, a real estimation of the execution time of the whole application is achieved.

6.1 SPARSKIT2 library

SPARSKIT2 is the library used in the sequential BIPS3D simulator. Reordered matrices generated using our technique were obtained after a global reordering by rows and columns, as Section 5.3 explains. CG, BiCGSTAB and FGMRES were selected as iterative methods for the tests. This is because, on the one hand, these methods are broadly used in scientific computations, and on the other, the cost per iteration of the sparse matrix–vector product varies among them. Note that although the application matrices are symmetric positive definite, we consider BiCGSTAB and FGMRES to illustrate typical results on nonsymmetric solvers. Usually nonsymmetric solvers have longer orthogonalization phase, and therefore, the importance of the SpM$\times$V is less. As will be shown later, SpM$\times$V in CG means on average about 60% of the total time per iteration, 48% for BiCGSTAB, and about 30% in the case of FGMRES. An incomplete factorization ILU(0) was used as preconditioner because it allows a good convergence of the considered iterative methods in order to solve the Poisson equation.

As will be shown in this section, reorderings guided by distance functions evaluated over pairs of rows or columns (Equations 3 and 4) are inefficient due to the high locality of the original FEM meshes considered and the large size of the distance graphs generated. This last feature means that the reordering heuristic (Lin-Kernighan algorithm) needs a higher number of iterations than the fixed threshold used in our tests in order to find a permutation vector that reduces the total distance (D_i) of the original matrix. In other words, reorderings guided by distance functions evaluated over pairs of consecutive rows or columns require a high computational time to be efficient.

Nevertheless, reorderings guided by the distances evaluated over windows of locality, as the results in this section show, are more efficient than those performed by METIS. Furthermore, these reorderings are generated in less time than the reordered matrices that use distance functions evaluated over pairs of consecutive rows or columns (that is, with window size $w = 1$). The use of window sizes higher than 1 allows us to reduce the number of nodes in the weighted graph to $\lceil N/w \rceil$. It implies an important decrease in the computational time needed for its calculation, together with reductions in the problem size to be managed by the reordering heuristic. Moreover, due to the intrinsic nature of the regular FEM meshes and according to our model, windows of locality are formed by sets of consecutive rows or columns with a high probability of data reuse.



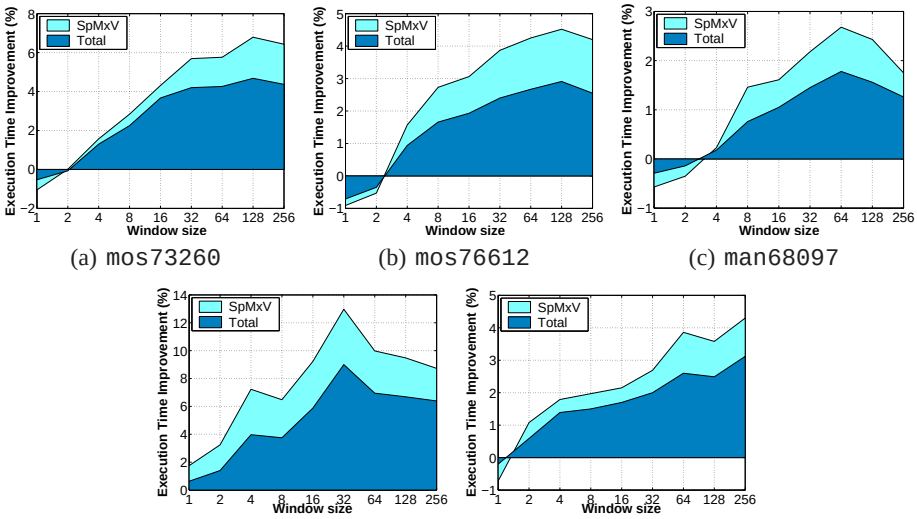
(a) mos73260 (b) mos76612 (c) man68097

(d) man135214 (e) poisson33701

Fig. 7. Execution time improvement (per iteration) obtained by the reorderings guided by the distance functions evaluated over windows of locality using BiCGSTAB.

Next we analyze the behavior of the reorderings guided by distance functions evaluated over windows of locality of sizes from 1 to 256. Figure 7 shows the evolution of the performance when the window size increases using the BiCGSTAB method. Improvements considering only the time spent by the sparse matrix–vector products and the total execution time are displayed. Results displayed are the average of those obtained by the two distance functions (d_{w1} and d_{w2}), because their performance is very similar and only differs, in the worst case, in no more than 3%. It can be observed that the most important improvements are always achieved by reorderings with a window size greater than 1. Specifically, $w = 128$ for `mos73260` and `mos76612` matrices, $w = 64$ in the case of `man68097`, $w = 32$ for `man135214` and finally, $w = 256$ for `poisson33701`. The best results with respect to METIS reorderings are achieved by reordered matrices of `man135214`. Improvements are about 12% if only products are considered, and slightly higher than 7% for the total execution time of the iterative method. It is important to highlight that the search for a path of minimum length with the reordering heuristic (Lin-Kernighan) is fixed to a number of iterations, in such a way that if a higher number of iterations were allowed (at the expense of a greater overhead of the reordering technique), improvements would be more important, although, limited to the number of conflict misses produced by the original matrix.

Figures 8 and 9 show the performance achieved by the reorderings using different window size with CG and FGMRES. First, if we compare the results for these two methods with the results obtained using BiCGSTAB (Figure 7), differences in the influence of the SpM$\times$V operation in the total execution time per iteration are observed. In this way, the influence of the product in the CG method is high (it implies, as we have commented previously, about 60% of the total execution time per iteration), while



(a) mos73260 (b) mos76612 (c) man68097

(d) man135214 (e) poisson33701

Fig. 8. Execution time improvement (per iteration) obtained by the reorderings guided by the distance functions evaluated over windows of locality using CG.

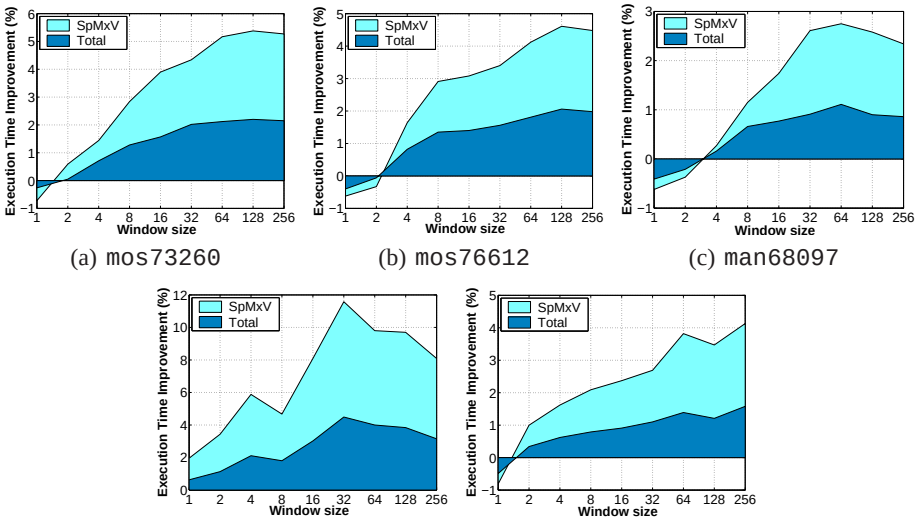
in the case of FGMRES this influence decreases to 30%. On the other hand, although the same reorderings are utilized by the three iterative methods, improvements in the SpM×V operation are different. This is because according to the operations performed per iteration by the methods, there are different data in the cache that can be reused in the next iterations.

For CG and FGMRES, the most remarkable improvements are obtained by the same reorderings showed in the BiCGSTAB analysis. Regarding CG, in the case of the SpM×V time, the improvement ranges from 2.7% (matrix `man68097`) to 13% (matrix `man135214`), and when considering the total execution time, the reduction varies from 1.6% (matrix `man68097`) to 9% (matrix `man135214`). In the case of FGMRES, when only the time spent by the irregular operation is considered, the improvement goes from 2.7% (matrix `man68097`) to 11.6% (matrix `man135214`). When the total time is analyzed, the reductions are, in the worst case, 1.3% for matrix `man68097` and 11.6% when `man135214` is used.

The results for the different iterative methods and for the several FEM meshes considered point out that reordered matrices, obtained using our locality improvement technique, outperform those generated by METIS.

6.2 PSPARSLIB library

In this section the results using the parallel library PSPARSLIB are shown. In this case just the performance of the reorderings using the BiCGSTAB method is displayed. This is because the weight per iteration of the SpM×V for the methods analyzed (CG, BiCGSTAB and FGMRES) is the same as that observed with SPARSKIT2 library (see



(a) mos73260 (b) mos76612 (c) man68097

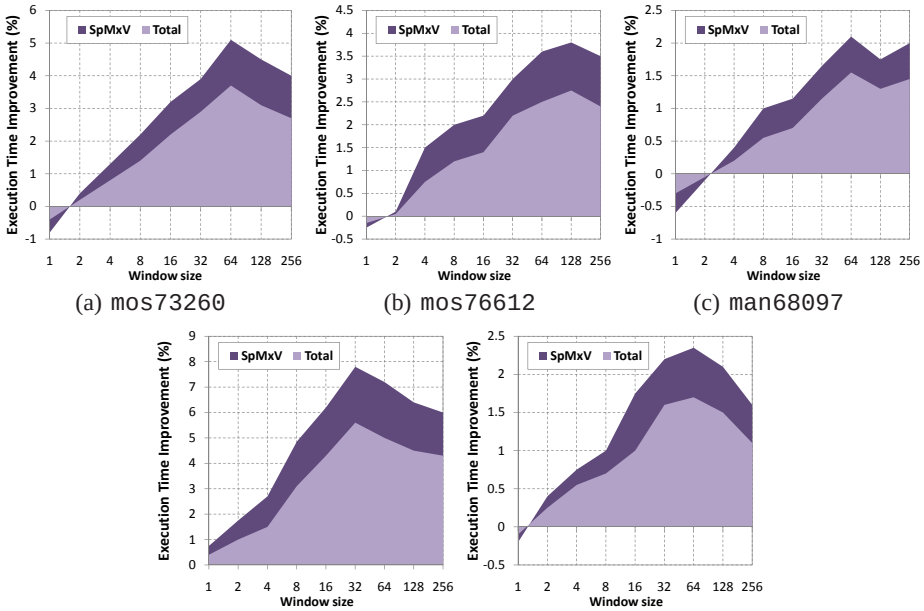
(d) man135214 (e) poisson33701

Fig. 9. Execution time improvement (per iteration) obtained by the reorderings guided by the distance functions evaluated over windows of locality using FGMRES.

Section 6.1). Therefore, in order to show the benefits of our proposal, we have selected the intermediate case, that is, the BiCGSTAB method with about 50% on average of the total time per iteration spent by the sparse matrix–vector product operation. As preconditioner we have utilized the additive Schwarz method with an incomplete factorization ILU(0) of the local matrices (see Section 3.1), which provide good results in terms of convergence for the considered problems.

The behavior of the reorderings using distance functions evaluated over pairs of consecutive rows (or columns) is similar to that observed with the SPARSKIT2 library. Results for the five sparse matrices analyzed in PSPARSLIB suggest that the most efficient reorderings always correspond to window sizes greater than 1, regardless of the number of processors. As example, Figure 10 shows the execution time improvement with respect to METIS reorderings when 8 processors are considered. Note that the values displayed in the graphics correspond with the average performance obtained by the reorderings guided by d_{w1} and d_{w2} (results offered by these two distance functions are very similar). Best performance is achieved for matrices `mos73260`, `man68097` and `poisson33701` when $w = 64$, while for matrices `man135214` and `mos76612` is obtained when $w = 32$ and $w = 128$ respectively. Best results overall are provided by `man135214` reorderings, with reductions up to 7.8% in the SpM$\times$V time and up to 5.6% when the total execution time is considered.

Figure 11 shows, for the five sparse matrices considered, the execution time improvements with respect to METIS using a different number of processors. Reorderings were performed using the window size that experimentally obtained the best performance on average for each matrix. The sizes of the windows correspond to those detailed in the above paragraph. Note that we have experimentally checked that in most



(a) mos73260 (b) mos76612 (c) man68097

(d) man135214 (e) poisson33701

Fig. 10. Execution time improvement (per iteration) achieved by our reorderings in PSPARSLIB considering 8 processors and using BiCGSTAB.

of the cases the best performance for any matrix is obtained with the same window size regardless of the number of processors. An analysis of the results shows that when the number of processors is higher, a decrease in the improvements achieved by our reorderings is observed. This behavior is mainly due to the size reduction of the local matrices when the number of processors increases. In this way, the probability of conflicts within the caches of each processor is lower, and therefore, the benefits of applying our locality improvement technique decrease. Even then, improvements of up to 4% in the total execution time are reached when 32 processors are considered (matrix `man135214`). Best results overall are obtained by matrix `man135214` using two processors, achieving improvements up to 12% when only the product time is measured, and 8% when the total execution time is considered (Figure 11(d)). As in the case of SPARSKIT2 library, matrix `man68097` presents the lowest execution time improvements (Figure 11(c)).

We can conclude that reordered matrices generated by our technique increase the performance with respect to those obtained by METIS when using PSPARSLIB parallel library.

6.3 BIPS3D simulator

In this section, the results integrating all the components of the device simulator are shown. In order to solve the Poisson equation, the BIPS3D simulator uses routines from the SPARSKIT2 library (for the sequential execution), or from the PSPARSLIB library (in the case of parallel execution).

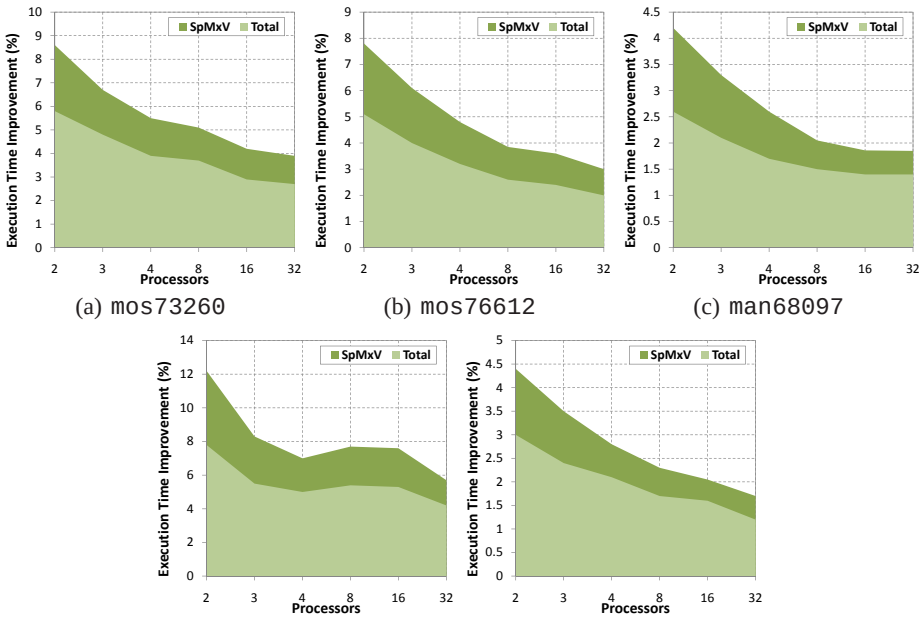


Fig. 11. Execution time improvement (per iteration) achieved by our reorderings in PSPARSLIB considering different number of processors and using BiCGSTAB.

Table 3 displays the sequential execution time of the processing stage of the simulation using the BiCGSTAB method, together with the preprocessing overhead produced by our reordering technique and METIS. The number of iterations required by the iterative method is also provided. The reorderings of matrices `man68097`, `man135214` and `poisson33701` are guided using distance function d_{w2} with a window size w of 64, 32 and 256 respectively. In the case of `mos73260` and `mos76612` reordered matrices, the selected distance function was d_{w1} with $w = 128$.

Note that the time spent performing the reordering using our proposal is lower than the execution time required by METIS for all the meshes analyzed. In comparison with METIS, with the use of our reorderings the reduction in the total simulation time ranges from 5.4% in the worst case (matrix `man68097`) to 12% for matrix `man135214`. We must take into account that the number of iterations required to solve the systems of linear equations is different depending on which matrix is considered (see Table 3). For all the cases studied, the iterative method converges in the same (or slightly different) number of iterations using METIS and our locality optimization technique. However, even if the reordered matrices convergence is better, the observed improvements are mainly due to the increase in the locality of the accesses and the corresponding reduction in the total execution time per iteration.

Finally, the results of the parallel execution of the simulator are analyzed. Table 4 shows the execution times for the processing and reordering stages of the simulator using 8 processors. The number of iterations required by the iterative method is also

	mos73260		mos76612		man68097		man135124		poisson33701	
	METIS	d_{w1}	METIS	d_{w1}	METIS	d_{w2}	METIS	d_{w2}	METIS	d_{w2}
Reordering	4.6	3.6	4.8	3.5	5.2	3.1	12.7	10.1	2.5	1.1
Processing	82.9	77.1	84.1	79.1	77.5	75.2	95.2	84.8	45.1	43.3
Total Time	87.5	**80.7**	88.9	**82.6**	82.7	**78.3**	107.9	**94.9**	47.6	**44.4**
Iterations	1699	1695	1707	1715	1912	1908	2101	2101	1545	1553

Table 3. Sequential simulation time (in seconds) using our reorderings and the ones obtained using METIS.

	mos73260		mos76612		man68097		man135124		poisson33701	
	METIS	d_{w2}	METIS	d_{w2}	METIS	d_{w2}	METIS	d_{w2}	METIS	d_{w2}
Reordering	4.6	4.2	4.8	4.4	5.2	4.0	12.7	13.0	2.5	2.1
Processing	23.2	21.2	23.9	22.1	21.0	20.3	28.0	25.5	14.3	13.8
Total Time	27.8	**25.4**	28.7	**26.5**	26.2	**24.3**	40.7	**38.5**	16.8	**15.9**
Iterations	2591	2602	2325	2309	1877	1876	2360	2374	1757	1763

Table 4. Parallel simulation time (in seconds) using our reorderings and the ones obtained using METIS.

displayed in the table. We have selected the reorderings generated using the distance function d_{w2} and window sizes of 32 (matrix man135214), 64 (matrices mos73260, man68097 and poisson33701) and 128 (matrix mos76612), because they show the best behavior in the previous analysis with PSPARSLIB library (see Section 6.2).

As in the sequential case, preprocessing time needed for reordering using our proposal is lower than the time required by METIS, except for matrix man135214. Even in this case, the total execution time (and therefore, the processing one) is lower when our reorderings are considered. Improvements of up to 9% have been reached for matrix mos73260. Convergence using our reorderings, as in the sequential case commented above, is similar to METIS matrices (see Table 4). According to this, the decreases in the execution time are mainly produced by the improvements in the locality of the sparse matrix-vector product and not due to the convergence of the iterative method.

Therefore, taking into account all the results, we can conclude that the application of the locality optimization technique to a problem of electronic devices simulation is appropriate to increase performance. Lower execution times than those obtained using standard reordering techniques as METIS are achieved. Improvements are obtained both in sequential and parallel executions of the BIPS3D simulator.

7 Conclusions

In this paper a technique to deal with the problem of locality in iterative methods is presented. The main kernel of these methods is the sparse matrix–vector product. Finding a solution to a system of linear equations may require thousands of these operations.

The proposed technique consists of reorganizing the data guided by a locality model instead of restructuring the code or changing the sparse matrix storage format. The goal is to increase the grouping of elements in the pattern of the sparse matrix that charac-

terizes the irregular accesses, and therefore, to improve both the spatial and temporal locality in the execution of the code.

The reordering technique has been applied to several iterative methods from two broadly used numerical libraries. Results show an impact on the overall performance of the considered iterative method due to the increase in the locality of the sparse matrix–vector product. Noticeable reductions in the execution time have been achieved both in sequential and in parallel cases. Moreover, the results of our reordered matrices outperform those observed when the standard METIS tool is used.

This positive behavior allows the reordering technique to be successfully applied to real problems. We have focused on the simulation of semiconductor devices, and in particular, the BIPS3D simulator was studied. The technique was integrated into the simulator, and a comparison with the original one was performed. Both sequential and parallel executions have been extensively analyzed in this paper. According to the results, we have observed that the overhead due to our reordering technique is lower than the one produced by METIS. Moreover, noticeable reductions in the execution time required by the simulations are observed when using our reordered matrices in comparison with the original simulator.

As future work, the problem of the selection of the optimal window size will be considered. Results show that this size depends on the sparsity pattern of the matrix and on the architecture characteristics.

Acknowledgments

This work has been partially funded by project TIN2007-67537-C03-01 of Spanish Ministry of Education and Science (MEC).

References

1. P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.
2. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.
3. A. Buttari, V. Eijkhout, J. Langou, and S. Filippone. Performance optimization and modeling of blocked sparse kernels. *Int. Journal of High Performance Computing Applications*, 21(4):467–484, 2007.
4. A. L. G. A. Coutinho, M. A. D. Martins, R. M. Sydenstricker, and R. N. Elias. Performance comparison of data-reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations. *Int. Journal for Numerical Methods in Engineering*, 66(3):431–460, 2006.
5. E. Cuthill and J. McKee. *Several strategies for reducing the bandwidth of matrices*. Rose and Willoughby, 1972.
6. B. B. Fraguela, R. Doallo, and E. L. Zapata. Memory hierarchy performance prediction for blocked sparse algorithms. *Parallel Processing Letters*, 9(3):347–360, September 1999.
7. A. J. García-Loureiro, J. M. López González, T. F. Pena, and Ll. Prat. Parallel finite element method to solve the 3D Poisson equation and its application to abrupt heterojunction bipolar transistors. *International Journal for Numerical Methods in Engineering*, 49:639–652, 2000.

8. D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Modeling data locality for the sparse matrix-vector product using distance measures. *Parallel Computing*, 27:897–912, 2001.
9. E. J. Im and K. Yelick. Optimizing sparse matrix vector multiplication on SMPs. In *Proc. of the 10th SIAM Conf. on parallel processing for scientific computing*, March 1999.
10. E. J. Im, K. Yelick, and R. Vuduc. SPARSITY: Framework for optimizing sparse matrix-vector multiply. *Int. Journal of High Performance Computing Applications*, 18(1):135–158, 2004.
11. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1998.
12. K. Kourtis, G. Goumas, and N. Koziris. Optimizing sparse matrix-vector multiplication using index and value compression. In *Proc. of the Conference on Computing Frontiers*, pages 87–96, 2008.
13. J. J. Navarro, E. García, J. L. Larriba-Pey, and T. Juan. Block algorithms for sparse matrix computations on high performance workstations. In *Proc. IEEE Int'l. Conf. on Supercomputing (ICS'96)*, pages 301–309, 1996.
14. L. Oliker, X. Li, P. Husbands, and R. Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Review*, 44(3):373–393, 2002.
15. J. C. Pichel, D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Improving the locality of the sparse matrix–vector product on shared memory multiprocessors. In *Proc. of the 12th Euromicro Conf. on Parallel, Distributed and Network-based Processing*, pages 66–71, 2004.
16. J. C. Pichel, D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Performance optimization of irregular codes based on the combination of reordering and blocking techniques. *Parallel Computing*, 31(8–9):858–876, 2005.
17. J. C. Pichel, D. E. Singh, and J. Carretero. Reordering algorithms for increasing locality on multicore processors. In *10th IEEE Int. Conf. on High Performance Computing and Communications*, pages 123–130, 2008.
18. A. Pinar and M. Heath. Improving performance of sparse matrix-vector multiplication. In *Proc. of the AMC/IEEE Supercomputing*, page 30. ACM Press, 1999.
19. Y. Saad. SPARSKIT: a basis toolkit for sparse matrix computations. Technical report, University of Illinois, Urbana, 1994.
20. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics (SIAM), 2003.
21. Y. Saad, G. Lo, and S. Kuznetsov. PSPARSLIB users manual: A portable library of parallel sparse iterative solvers. Technical report, Univ. of Minnesota, Computer Science dept., 1998.
22. M. M. Strout, L. Carter, J. Ferrante, and B. Kreaseck. Sparse tiling for stationary iterative methods. *Int. Journal of High Performance Computing Applications*, 18(1):95–113, 2004.
23. S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. *IBM Journal of Research and Development*, 41(6):711–725, 1997.
24. R. Vuduc and H. Moon. Fast sparse matrix-vector multiplication by exploiting variable block structure. In *High Performance Computing and Communications, volume 3726 of Lecture Notes in Computer Science*, pages 807–816, 2005.
25. S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiply on emerging multicore platforms. In *Proc. of Supercomputing*, 2007.