Efficient Representation and Analysis of Power Grids

João M. S. Silva
INESC ID
IST - TU Lisbon
Lisboa, Portugal
jmss@algos.inesc-id.pt

Joel R. Phillips
Cadence Research Labs
Cadence Design Systems
Berkeley, California, U.S.A.
jrp@cadence.com

L. Miguel Silveira
INESC ID / Cadence Research Labs
IST - TU Lisbon
Lisboa, Portugal
lms@inesc-id.pt

Abstract

Modern deep sub-micron ULSI designs with hundreds of millions of devices require huge grids for power distribution. Such grids, operating with increasingly low-power voltages, are a design limiting factor and accurate analysis of their behavior is of paramount importance as any voltage drops can seriously impact performance or functionality. As power grid models have millions of unknowns, highly optimized special purpose simulation tools are required to handle the time and memory complexity of solving for their dynamic behavior. In this work, we propose a hierarchical matrix representation of the power grid model that is both space and time efficient. With this representation, reduced storage matrix factors are efficiently computed and applied in the analysis at every time-step of the simulation. Results show an almost linear complexity growth, namely $O(n\log^a(n))$, for some small constant a, in both space and time, when using this matrix representation. Comparisons of our academic implementation with production-quality code proves this method to be very efficient when dealing with the simulation of large power grid models

1 Introduction

In recent years, the relentless trend for high-performance with low-power consumption has raised new challenges to designers. Higher performance has meant increased functionality fueled often by technology scaling. Achieving lower power has been met by specialized design techniques together with supply voltage scaling. However, adding more functionality implies that more devices must be powered, and thus huge power distribution networks are now deployed throughout the design. Moreover, lower supply voltage makes potential voltage drops a more serious concern, as they may seriously impact performance or functionality of the whole design by reducing noise margins and slowing down devices [15]. Ultimately, this may lead to circuit failure due to excessive delay or simply malfunctioning devices. Therefore, it has become clear that in modern circuits, proper design and behavior of power grids is a performance limiting factor. The efficient verification of such grids is thus seen as an essential step in predicting and ensuring correct behavior and performance.

The simulation of power distribution networks is a difficult task owing to the huge number of elements in such circuits [17, 19]. Much research and development has been devoted to this problem, both in the modeling and simulation phases. In this paper we concentrate on the analysis or simulation part of the problem. We will assume a typical model of a power grid to consist of an extremely large RC network, sometimes with millions of nodes, with a complex set of excitation sources and drains. Inductance is sometimes also included in the model, typically to model wire bonding, but by comparison it represents a very small subset of the overall model. In this paper we will restrict ourselves to the RC portion of the model as its analysis is always necessary even if inductance is included. Block techniques can be used to handle the coupling between the large RC block and the inductance-modeled packaging/wiring sub-circuit. The network excitations usually consist of the biasing sources, generally modeled as constant voltage sources, and the connection to the designed-in devices, which act as sources or drains depending on their electrical state. A simplified model for such sources is to consider them as time-varying independent current sources, but more complex models that take into account loading and feedback into the network are sometimes used. This simplified modeling approach has the important implication that the grid model becomes a linear system, thus much easier to simulate. Commercial large scale power grid simulators in use nowadays typically assume a model as described.

From a simulation standpoint, there are two families of techniques for the time analysis of the linear description of the power grid: iterative or direct methods. For DC-type problems that require a single system solution, iterative problems are preferred. However, robust verification techniques require that dynamic analysis be performed, which implies solving for the time-evolution of the system along a given simulation interval. Taking advantage of the problem structure and inspired by knowledge gained in solving similar problems resulting from discretized elliptic partial-differential equations (PDE), efficient algorithms have been proposed based on preconditioned conjugate gradient [5] or Multigrid [16, 20]. These methods show good convergence ratios, but require several iterations for recomputing the solution of the system **for each time-point** (in essence the dynamic problem is similar to solving systems with multiple right-hand sides). Direct methods, on the other hand, find an a-priori system matrix decomposition which can then be used in accelerating the solution at each time-step. Even though the resulting matrix factors are very costly to store and compute, most commercial power grid analysis tools

978-3-9810801-3-1/DATE08 © 2008 EDAA

available nowadays use some form of highly optimized direct solver technology.

In this paper, we explore an approach based on using direct methods to solve the linear subset of equations which results from the formulation of a power distribution network. The novelty of the proposed technique is related to how we address the issues of storage and computational cost. While we still compute a factorization of the system matrix and apply the resulting factors to obtain the solution at each time-step, we propose to use a hierarchical matrix representation of the underlying system based on an $\mathcal{H}$-Matrix representation [3]. With this representation, the increase in matrix density that comes from the factorization is managed to obtain a reduced storage data structure. Such a representation is then efficiently applied to generate the system solution at each time-step in the simulation window. $\mathcal{H}$-Matrices show almost linear complexity in both storage and evaluation. The initial hierarchical factorization procedure, while having a slightly higher complexity, is nevertheless also almost linear. The efficient storage and evaluation properties of the proposed method, characterized by complexities growing as $O(n\log^a(n))$ for some small constant a, immediately pay-off in the fast evaluation that results from the application of the matrix factors to repeatedly obtain the solutions at every time-step in the analysis interval.

The remainder of this paper is organized in the following manner: in Section 2 we present some background on the power grid simulation problem and in Section 3 we introduce $\mathcal{H}$-Matrices theory to the extent required for this paper. Existing codes that support $\mathcal{H}$-Matrices representations [3] assume geometrical information is available which may not always be the case. Therefore an algebraic version of the $\mathcal{H}$-Matrices formulation was developed and will be described. In Section 4, we compare our academic implementation against a highly optimized direct solver using state-of-art storage and reordering algorithms on several 2D and 3D, regular and irregular synthetic systems. Results indicate this method to be very efficient when dealing with the simulation of large power grid models. Finally, in Section 5 conclusions are drawn.

2 Background

For this work we assumed a simplified three-dimensional power grid model like the one depicted in Figure 1. An arbitrary number of metal layers dedicated to power delivery can be considered. This model also assumes that VDD and GND strips, as well as vias, are modeled resistively (R_{strip} and R_{via}, respectively). The coupling resulting from the overlapping between metal strips in different levels is modeled through $C_{overlap}$. Notwithstanding, other kinds of parasitic effects, such as coupling between strips in the same level, etc can be included. While simplified, this type of model is to some extent representative of what is used in commercial tools.

Assuming the extracted netlist consists of n nodes, the network equations can generally be written as:

$$C\frac{dv(t)}{dt} + Gv(t) = i(t) \tag{1}$$

where $C, G \in \mathbb{R}^{n\times n}$ are the matrices modeling the dynamic

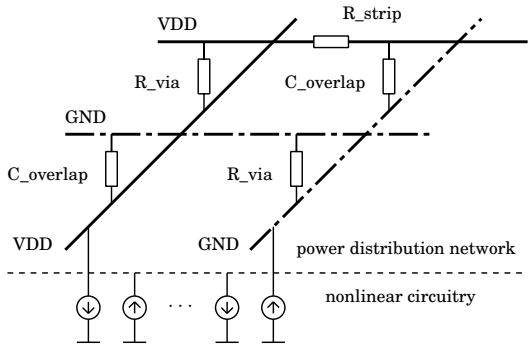

Figure 1. Power grid model.

and static network components, respectively, $v \in \mathbb{R}^n$ is the vector of voltages at the grid nodes, and $i \in \mathbb{R}^n$ the vector of currents imposed at those same nodes. If time-varying independent current sources are assumed at the sources/drains of the network, then the formulation is akin to nodal analysis (NA). In this case, G, which may reflect an unstructured grid, is a sparse matrix very similar to those encountered in (finite-difference) discretized 3D problems (no more than 7 elements per row). For power grids however, the third dimension is shallow, compared to the other dimensions, as chip height is much smaller and less dense than the die area. On the other hand, and since in the assumed model we only have capacitances between different planes in the z direction, C is a 3-diagonal sparse matrix. Both matrices are therefore extremely sparse and fairly regular. Adding additional capacitance coupling increases the density in the C matrix, but most of the properties are retained.

In order to analyze the grid in the time domain, we can use, as an example, Backward Euler's method and discretize the time interval of interest in steps of constant size h, obtaining:

$$\underbrace{\left(\frac{C}{h}+G\right)}_{A} v(t) = \underbrace{i(t)+\frac{C}{h}v(t-h)}_{b} \tag{2}$$

where b is the right hand side at each time-step. Analysis of the power grid then entails solving the above system, $Av = b$ at every time step in the analysis interval.

The trivial solution to Eqn. (2) is $v = A^{-1}b$. Unfortunately, although A is sparse, A^{-1} is full and its inversion is prohibitive. Iterative methods, such as Conjugate Gradient (CG), preconditioned CG (e.g. ICCG – Incomplete Cholesky preconditioned CG [5]) or Multigrid (MG) can be used to solve (2). Multigrid type algorithms are interesting in that they possess optimal theoretical complexity properties. However, in practice, the setup costs and the constant terms associated with the complexity estimates do not seem to provide much advantage, as discussed in [16]. For dynamic analysis of power grid systems, direct methods are still the method of choice, as the cost of computing matrix factors is amortized over all time-step solutions.

3 Hierarchical Matrices

Hierarchical Matrices, or $\mathcal{H}$-Matrices [10, 12], enable matrix operations in almost linear complexity, where "al-

most linear" means linear up to logarithmic factors. $\mathcal{H}$-Matrices are the algebraic counterpart of panel clustering techniques [13] for integral operators. Due to the existence of Green's function for elliptic problems, these techniques can be extended to inverses [2] and factorizations [1, 4] of finite-element and finite-difference type matrices. Given the already noted structural similarity between the power grid formulation and those resulting from elliptic PDEs [16], this implies that efficient representations exist for the inverse and the LU/Cholesky factors of the underlying matrix describing the power grid model. Once that representation is formed, efficient, almost linear computations are within grasp. Theoretical and practical results indicate both storage and computational complexity to grow as $O(n \times \log^a(n))$, for some small constant a. In this section we discuss how to generate such a representation and how to operate with it.

3.1 $\mathcal{H}$-Matrix Fundamentals

$\mathcal{H}$-Matrices are super-matrices (in the sense they may contain either full, low-rank and other super-matrices) with an inherent hierarchy of block splitting. Consider the splitting of A from Eqn. (2) in 2×2 blocks:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{3}$$

If, for instance, A_{11} and A_{22} are fairly dense (large number of nonzero entries), they should be represented by full matrices. On the other hand, if A_{12} and A_{21} are fairly sparse they can be represented in a factorized form:

$$A_{s \times t} \approx MN^T \tag{4}$$

where $M \in \mathbb{R}^{\#s \times k}$, $N \in \mathbb{R}^{\#t \times k}$ and k is the rank of the block up to some accuracy, eps. This representation will be most efficient if $k \ll \#s, \#t$. Obviously, not all blocks will allow such a representation. The goal of finding a $\mathcal{H}$-Matrix representation is akin to determining the right reordering and blocking of nodes (rows/columns), that maximizes the number of blocks that can be represented in factorized form. Note that the low-rank block representation in (4) is approximate. According to the pre-specified accuracy parameter eps, we can automatically control the rank-k used to represent the low-rank blocks in factorized form. An error bound for the low rank approximation of matrix blocks is given in terms of the Frobenius norm as [3]:

$$\|A - \tilde{A}\|_F \leq \frac{3}{2} n^{-1} 3^{-k} \tag{5}$$

If a block is not represented in factorized form, it may be further split and its sub-blocks recursively tested for such a representation. So, $\mathcal{H}$-Matrices are matrices which result from the recursive multilevel splitting of matrix blocks, until low-rank blocks are found and represented in the factorized form, or no further sparsity is available and full matrices must be used. Of course the optimum storage scheme for a sparse stiffness matrix such as A in (2) is well known, with only nonzero elements being stored in an efficient way. The more interesting question is how to represent the corresponding LU or Cholesky factors. While a matrix resulting from an FD discretization in 1D yields no LU or Cholesky fill-in, the same does not apply to 2D matrices and certainly not to 3D matrices, whose factors tend to fill up quickly.

3.2 $\mathcal{H}$-Matrix Splitting Approaches

In this work we use $\mathcal{H}$-Matrix to represent the Cholesky factorization of power grid models. This can be achieved in two distinct approaches: geometrically and algebraically.

3.2.1 $\mathcal{H}$-Matrix Geometric Splitting

In the geometric approach, the criteria to decide whether a block allows a low-rank approximation is based on the geometry of the underlying medium discretization. This criteria is called *admissibility*, meaning whether the block *admits* to be represented by a low-rank factorization or not. The admissibility check procedure, and therefore the reordering and clustering of nodes, must be such that blocks are obtained that allow for a low-rank factorization in terms of the LU or Cholesky factors, not on the original sparse matrix. Consider two sets of nodes in the physical domain, which we term as clusters, and assume that each cluster has a radius corresponding to a bounding box that includes all nodes. If the physical distance between clusters is much larger than the cluster radius, it is likely that the interaction between nodes in separate clusters can be represented by a low-rank approximation (this type of reasoning is akin to a multipole type approximation). If the cluster size is too large to satisfy the admissibility criteria, it is split and each resulting sub-cluster is then checked. The splitting process is repeated until a minimal block size is reached, (n_{min}), or the blocks can be approximated in a low-rank sense. Large blocks are inefficiently approximated by (4) and must be split whenever possible in order to give origin to smaller low rank blocks. This approach is the right one when handling regular structures where geometrical distance is a good criteria for estimating (electrical) influence.

The current release of the HLib package [11] provides code for dealing with matrices where the underlying geometrical information pertaining to the discretizations is available. Therefore the library is readily used for experiments in Section 4.

3.2.2 $\mathcal{H}$-Matrix Algebraic Splitting

An alternative approach is based on the algebraic information of the matrix. This is the technique of choice for irregular structures and likely the best one for common power grids. Whenever two nodes in the matrix are "connected" by an entry exhibiting a large magnitude, this means that the nodes are indeed tightly connected and should be kept together in the splitting process. This approach is akin to the standard heavy edge matching algorithm [14]. In these methods, used for instance in the publicly available graph partitioning tool Metis [14], nodes connected by a large conductance and/or capacitance are favored to be clustered together through the multilevel splitting process. The advantage of this method over the previous one is that no geometric information on the problem is required, since the structure of the $\mathcal{H}$-Matrix relies solely on the matrix entries. On the other hand, being able to follow geometric admissibility conditions leads to a slightly better approximation of the representation. The duality between geometric and algebraic splitting is quite similar to that found when considering Multigrid methods and Algebraic Multigrid methods.

In order to work with matrices from which we have no underlying physical information, we implemented on top of HLib an algebraic approach based on [18]. In the algebraic approach we use the information of the matrix itself and multilevel clustering methods, in this case Heavy Edge Matching (HEM) [14], to obtain the corresponding $\mathcal{H}$-Matrix . In terms of efficiency, this approach is comparable to the geometric approach based on Nested Dissection [4].

3.3 $\mathcal{H}$-Matrix Representation

To build an H-matrix corresponding to a sparse matrix, we first need to build a cluster tree over the matrix index set. This cluster tree describes the hierarchical clustering of the nodes of the matrix (from bottom to top) which yields the hierarchical partitioning of the matrix. Note that the clusters are composed of nodes which may not be adjacent, which implies row-column reordering may be required.

The difference between the algebraic and the geometric approaches for building the cluster tree is that in the geometric approach we use geometric conditions (admissibility conditions) to determine whether or not a block of the matrix will be partitioned further. If a block is admissible, then it can be approximated by a low-rank matrix. On the other hand, in the algebraic approach, the construction of the cluster tree is based on multilevel clustering methods which are widely used in graph partitioning. The basic idea of multilevel clustering is to start from the finest graph which represents the matrix and build clusters over its nodes, then build a coarse graph by merging the nodes in the same cluster, and continue this coarsening process on the coarsened graphs until the graph obtained is small enough. This procedure uses the edge weights from the coarse graphs to make decisions on the merging of nodes. In essence, the cluster tree, is a format to represent and store the matrix. The $\mathcal{H}$-Matrix representation has the same tree structure as the cluster tree. The matrix entries are in fact the leaves of the cluster tree.

3.4 $\mathcal{H}$-Matrices Arithmetic and Complexity

As discussed, our goal is to efficiently compute a hierarchical LL^T Cholesky factorization of the symmetric matrix A in Eqn. (2) and then proceed with forward and backward solves at each time-step (for non-symmetric formulations, an LU factorization is generated in a similar fashion). In this work, this Cholesky factorization is performed with the proper arithmetic functionality provided by the $\mathcal{H}$-Matrices library, which is established in [9].

In terms of complexity, the storage of a rank-k factorized $n \times n$ matrix requires $2 \times k \times n$ (instead of n^2) elements. For the $\mathcal{H}$-Matrix , the storage is $O(n \times log(n) \times k)$, in which k is the worst-case rank of the sub-matrices of the $\mathcal{H}$-Matrix [9]. The Cholesky decomposition in the $\mathcal{H}$-Matrix format requires $O(n \times log^2(n) \times k^2)$ operations and the evaluation of the LL^T factors in each time-step requires $O(n \times log(n) \times k)$ operations, which is proportional to the storage requirements of the $\mathcal{H}$-Matrix representation of the Cholesky factors.

In Section 4 we will see how well these $\mathcal{H}$-Matrix based methods compare with other well-known method for example problems of increasing dimension.

4 Results

In the following, we present the experimental setup used in our work and the corresponding results. The following methods were tested for solving the system resulting from the nodal analysis formulation of power grid models: i) hierarchical Cholesky based on geometric admissibility (**gCh**), ii) hierarchical Cholesky based on algebraic admissibility (**aCh**) and iii) sparse Cholesky (**sCh**). We have decided to account for three measures of efficiency. The first is the setup time, where the hierarchical methods compute the super-matrix structures and the Cholesky decomposition, while **sCh** computes a matrix row/column permutation which tends to minimize Cholesky factors fill-in and the decomposition itself. The second is the solve time, where the Cholesky factors are used to obtain the solution to the desired time-steps in a time analysis. We also measured storage requirements for all methods. Finally, we discuss how the accuracy parameter eps affects the approximation error.

4.1 Experimental Setup

Since our work focuses on the efficiency of power grid simulation, and for the sake of complexity analysis, we decided to work with artificially generated matrices. These represent the main characteristics of a system resulting from the model extraction of a power grid structure. By choosing artificial matrices, we can easily control their size and characteristics in a meaningful way. In the following we will show results for 2D and 3D problems, representing regular and irregular grids, of increasing sizes. In the 3D grids, the number of metal layers has been fixed at 8 (z direction). Irregularity is emulated by generating random matrix entries in the interval $[0, 1[$ and discarding entries smaller than 0.5.

The algorithms tested were implemented in C upon the libraries HLib [3] and Cholmod [6, 7, 8]. HLib provides some code for 2D FEM problems, which has been modified to handle our 2D FD problems. The code related to the algebraic approach, **aCh**, was implemented on the HLib data structures and uses library functions for $\mathcal{H}$-Matrix arithmetic. We will use it for 3D problems. Cholmod, from the SuiteSparse package from the University of Florida (which also provides UMFpack among other well known tools) was used to implement the **sCh** method and was compiled with the `supernodal` option for maximum performance. Both HLib and Cholmod used the same versions of the widely known Lapack and Blas libraries. All codes were integrated in a single executable. The comparisons we establish are thus reasonably fair even though HLib is not a commercial-quality code and our implementation is only a prototype. **sCh** on the other hand, has a slight edge since it is based in highly optimized code.

Finally, experiments were run on a Dual AMD Opteron operating at 2.4 GHz with 16 GB of memory for increasing discretization sizes. In all experiments, the norm of the solution vector was computed in order to verify the correctness of the solution. Time was measured with the function `clock` from `glibc`).

Table 1. Setup times (in sec.).

	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D	16384	0.11	0.68
	65536	0.71	3.52
	262144	6.27	16.70
	1048576	124.27	**72.61**
	4194304	1285.64	**375.41**
	16777216	9921.41	**4068.08**
	number of nodes	sparse (Cholmod)	hierarchical (algebraic)
3D	131072	39.09	81.86
	524288	467.99	**415.58**
	2097152	5137.12	**2230.62**
	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D irreg.	16384	0.12	0.64
	65536	0.70	3.27
	262144	7.15	15.04
	1048576	130.38	**66.95**
	4194304	1307.74	**555.93**
	16777216	9296.28	**7425.37**

Table 2. Solve times per time-step (in sec.).

	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D	16384	0.01	0.01
	65536	0.05	0.05
	262144	0.21	0.24
	1048576	0.92	0.93
	4194304	4.02	**3.72**
	16777216	21.72	**17.74**
	number of nodes	sparse (Cholmod)	hierarchical (algebraic)
3D	131072	0.27	0.39
	524288	1.43	1.80
	2097152	7.26	8.58
	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D irreg.	16384	0.01	0.02
	65536	0.05	0.05
	262144	0.20	0.21
	1048576	0.92	**0.87**
	4194304	4.03	**3.45**
	16777216	21.68	**16.22**

Table 3. Storage requirements (in megabyte).

	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D	16384	8	11
	65536	39	49
	262144	175	210
	1048576	772	873
	4194304	3361	3555
	16777216	14559	**14351**
	number of nodes	sparse (Cholmod)	hierarchical (algebraic)
3D	131072	286	339
	524288	1528	1559
	2097152	7818	**7022**
	number of nodes	sparse (Cholmod)	hierarchical (geometric)
2D irreg.	16384	8	10
	65536	39	47
	262144	175	199
	1048576	772	821
	4194304	3361	**3258**
	16777216	14559	**13191**

4.2 Setup Time

The setup time corresponds to the time spent in creating matrix and vector structures, and computing a-priori factorizations. The results for the setup time are presented in Table 1 respectively for 2D and 3D regular grids and for a 2D irregular grid. From the table, we observe that the hierarchical approaches are more efficient than the sparse solver. For 2D problems at about 1 million nodes, this becomes noticeable. Even though it is difficult to see from the tables, the setup time of the hierarchical methods is also growing at a lower rate than the sparse approach. Theoretically this should be $O(n\log^a(n))$ for small constant a. For the 3D problem this behavior is not so clear since fewer points are available. Interestingly enough, the irregularity of the grid does not seem to affect the sparse solver as much as it affects the geometric hierarchical solver. Still the same type of behavior is noted in this example.

4.3 Solve Time

To compute the solve time, the Cholesky factors were used to solve for a given right-hand-side. Results for solving a single right-hand-side are presented in Table 2 respectively for 2D and 3D regular grids and for a 2D irregular grid. In terms of the solve time, we notice a similar pattern as in the setup time. However, the efficiency of the sparse solver slightly delays the advantage of the hierarchical methods for larger problems. In the 2D cases, only for problems with around 4 million nodes, do we see a clear advantage. For the 3D problem, the size of the grid is not sufficient for the hierarchical approach to show an advantage. Extrapolating the data in the table it is acceptable to assume this will happen for larger size problems.

4.4 Storage Requirements

Finally we look at the storage requirements of the various methods. Only matrix structures were taken into account (since vectors do not add that much to the total memory) and among these, only matrices used repeatedly in the time analysis. We assume other matrix structures can be freed after setup, and only evaluation matrices matter. The storage requirement results are shown in Table 3 again for the 2D and 3D regular grids and for the 2D irregular grids. The results in the tables are again similar to the other measures, but here the sparse solver is more competitive. This

Table 4. $\mathcal{H}$-Matrix approximation error $\|(LL^T)^{-1} \times A - I\|_2$ and required resources vs. the accuracy parameter for a 1024×1024 grid.

eps	setup time (s)	solve time (s)	space (Mbyte)	error
10^{-2}	45.50	0.54	491	9.282e-04
10^{-3}	45.43	0.54	498	5.797e-04
10^{-4}	45.47	0.55	501	1.984e-05
10^{-5}	47.69	0.55	502	6.425e-07
10^{-6}	48.93	0.55	502	6.404e-07

is likely a result of the sparse solver targeting lower fill-in in the matrix factors, while the hierarchical approaches, even though they also indirectly attempt to minimize fill-in, their main target is to compress the representation itself.

4.5 Approximation Error

In Table 4 one can see how the parameter *eps* affects the accuracy of the solution as well as the resource requirements. We can observe that as we increase the accuracy of the approximation, the demand for resources grows minimally. Of course, if the example grid was larger (with several millions of nodes, for instance) the resource needs would increase more strongly with the accuracy. Nevertheless, these examples show quite clearly that the time and space complexities are rising much slower compared to the added precision in the solution. It is also possible to infer that an $eps = 10^{-3}$ seems to be a reasonable initial choice for most of the problems, as confirmed experimentally.

As a final conclusion, we believe that the hierarchical approaches are indeed quite promising when compared to a state-of-the-art sparse direct solvers. We observe that **gCh** can solve 16 million node grids with around 14 GB of RAM at a rate of almost 1 million nodes per second.

5 Conclusions

$\mathcal{H}$-Matrices are hierarchical matrix representation schemes whereby blocks of the matrix are represented by low rank factorizations in a compact form. This representation enables computations and storage in almost linear time. In this way, Cholesky factors can be efficiently computed and represented, leading to fast system storage and solution. Experimental results show that using the hierarchical Cholesky representation requires $n\log^a(n)$, for some small constant a, in both space and time, when using this matrix representation, which proves this method to be very efficient when dealing with the simulation of large power grid models. Comparisons against a state-of-the-art sparse Cholesky code, using a very efficient reordering scheme, shows that the hierarchical matrix representations is very competitive and efficient. Very large problems can be solved with storage proportional to the number of nodes at a rate of about 1 million nodes per second.

References

[1] M. Bebendorf. Hierarchical LU decomposition-based preconditioners for BEM. *Computing*, 2004.

[2] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 2002.

[3] S. Börm, L. Grasedyck, and W. Hackbusch. *Hierarchical matrices*. Max Planck Institute for Mathematics in the Sciences, June 2006.

[4] S. L. Borne, L. Grasedyck, and R. Kriemann. Domain-decomposition based $\mathcal{H}$-LU preconditioners. *LNCSE*, 2005.

[5] T.-H. Chen and C. C.-P. Chen. Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 559–562, Las Vegas, Nevada, U.S.A., June 2001.

[6] T. A. Davis and W. W. Hager. Modifying a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 1999.

[7] T. A. Davis and W. W. Hager. Multiple-rank modifications of a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 2001.

[8] T. A. Davis and W. W. Hager. Row modifications of a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 2005.

[9] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[10] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 1999.

[11] W. Hackbusch, S. Börm, and L. Grasedyck. Hlib package. `http://www.hlib.org/hlib.html/`.

[12] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. part II: Application to multi-dimensional problems. *Computing*, 2000.

[13] W. Hackbusch and Z. P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 1989.

[14] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transactions on VLSI*, 7(1):69–79, 1999.

[15] D. Kouroussis and F. N. Najm. A static pattern-independent technique for power grid voltage integrity verification. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 99–104, Anaheim, California, U.S.A., June 2003.

[16] J. N. Kozhaya, S. N. Nassif, and F. N. Najm. A multigrid-like technique for power grid analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits (TCAD)*, pages 1148–1160, October 2002.

[17] S. R. Nassif and J. N. Kozhaya. Fast power grid simulation. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 156–161, Las Vegas, Nevada, U.S.A., June 2000.

[18] S. Oliveira and F. Yang. An algebraic approach for $\mathcal{H}$-matrix preconditioners. Technical report, University of Iowa, 2006.

[19] S. Pant and E. Chiprout. Power grid physics and implications for cad. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 199–204, New York, NY, USA, 2006. ACM Press.

[20] Z. Zhu, B. Yao, and C.-K. Cheng. Power network analysis using an adaptive algebraic multigrid approach. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 105–108, Anaheim, California, U.S.A., June 2003.