Fast Power Grid Simulation

Sani R. Nassif
IBM Austin Research Laboratory
11501 Burnet Rd., Austin, TX 78758
nassif@austin.ibm.com

Joseph N. Kozhaya
ECE Department,University of Illinois
1308 W. Main St., Urbana, IL 61801
kozhaya@uiuc.edu

Abstract

The decrease in feature size and added chip functionality in large sub-micron integrated circuits demand *larger* grids for power distribution. Since power grids are performance limiting factors [1, 2, 3], then their analysis is important in order to (1) predict the performance and (2) improve the performance if necessary. Thus, there is a clear need for new *efficient*, in terms of both execution time and memory, techniques for power grid analysis.

This paper discusses the modeling of power grids and proposes a new PDE-like multigrid method for the simulation of power grids. The proposed method is very *efficient* and suitable for both DC and transient simulation of power grids.

1 Modeling of Power Grids

The first step in power grid analysis involves modeling the grids and the power *sources* and *drains* [4]. Typically, power distribution within an integrated circuit is done from the top-level metal layer, which is connected to the package, down through inter-layer *vias* and finally to the active devices, as illustrated in Figure 1. The metal wires and vias are well modeled as a linear, time-invariant and passive network consisting of resistive, capacitive and -rarely- inductive elements. For modern integrated circuits such as microprocessors, such a network can easily include millions of nodes and tens of millions of elements.

As for the power *sources* and *drains*, their models can be quite complex. The models for the power sources can be involved enough to include sophisticated package and board models. On the other hand, the models for the power drains can account for the complex interaction between the power grid, the underlying non-linear circuit, and the time-varying signals propagating across the chip. However, the huge size of the power grid makes it infeasible to include any but the simplest models for the power sources and drains. Hence, power sources are modeled as simple constant voltage sources and power drains are modeled as time-varying current sources.

DAC 2000, Los Angeles, California

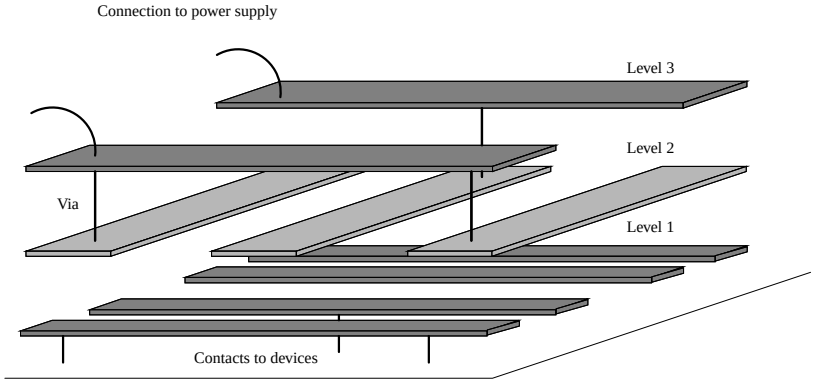

Figure 1: Power grid components.

Given the above, the complete power grid model is composed of a linear network of RLC elements excited by constant voltage sources and time varying current sources. The behavior of such a system can be expressed following the MNA formulation as the following ordinary differential equation:

$$Gx + C\dot{x} = u(t) \tag{1}$$

Where x is a vector of node voltages, and source and inductor currents; G is the conductance matrix; C includes the capacitance and inductance terms, and $u(t)$ denotes the time varying sources modeling the sources and drains.

2 Analysis of Power Grids

Due to the large size of typical power grids, general circuit simulators such as Spice [5] are not adequate for power grid analysis because of CPU time and memory limitation. The inefficiency of standard simulators [2, 6] comes about because (a) they require a lumped element approximation of the circuit which requires the translation of a regular geometrical structure to an expansive set of equivalent circuit elements, and (b) they use general purpose solution methods meant to be robust in the face of stiff systems of equations. By contrast, power grids are well behaved spatially (nearly regular) and temporally (damped). This motivates a special-purpose simulator for power grids which can make use of these properties.

If we apply the Backward Euler integration formula to Eq. 1 we generate a set of linear equations:

$$(G + C/h)x(t+h) = u(t+h) + x(t)C/h \qquad (2)$$

which can be readily simplified to $Ax(t+h) = b$ with $A = G + C/h$ and $b = u(t+h) + x(t)C/h$.

The solution of Eq. 2 requires the inversion (factorization) of the matrix $G + C/h$ which is independent of x, time-invariant, large and sparse. We note, however, that if we hold the time step h constant, then only one initial factorization is required, with a forward/backward solve at each time step. Since, for large matrices, a factorization is significantly more expensive than a forward/backward solve, the use of a constant time step results in large savings. The time step needs to be kept small enough to insure the accuracy of the solution. For application in the analysis of power grids of digital circuits, we find that using 100 steps per clock cycle (i.e. $h = 0.01 * T_{period}$) is sufficient.

To illustrate, we simulate a simple grid of 33 wires in each of the x and y directions, connected to a single voltage source at one of the corners, and loaded with 100 time dependent current sources at random locations. The resulting electrical model has 1089 nodes and a total of 1090 equations. We perform the simulation for 100 time steps. Spice [5] takes 13.3 sec. of CPU time, whereas our simulator implementing the method above takes 0.73 sec. for a net speedup of about 18x. Due to the superlinear dependence of solve time on matrix size, the speedup will be even more dramatic for the much larger systems normally encountered when simulating realistic power grids.

3 Proposed Analysis Method

In a well designed power grid, the grid resistance is much smaller than the equivalent sink resistance since the power grid is required to deliver as constant a voltage as possible to all sinks. This causes local power disturbances, as would be caused by a large localized sink, to be *spread* across an area much larger than that of the sink causing the disturbance. This spreading leads to voltage distributions which are spatially smooth, and motivates solution methods which can make use of this smoothness to speed up the solution process.

3.1 Power Grids as PDEs

We note that the solution of power grids results in a system of linear equations structurally identical to that of a finite element discretization of a two-dimensional parabolic partial differential equation (PDE). This motivates us to consider the power grid problem as a discretization of a continuous PDE where the solution is needed at a spatially fixed set of points.

Recently, the multigrid method (MG) has become the standard for solving smooth PDEs [7]. The basic idea of multi-grid methods is to solve the problem on a coarse grid and map the resulting solution back to the fine grid. Using an *iterative linear solver* for the fine grid, the mapped solution provides an excellent initial point and convergence is rapid. The major tools needed for that are *intergrid transfer operators* which are called restriction and prolongation operators. The restriction operator $\mathcal{R}$ maps the *problem* onto coarser grids while the prolongation operator $\mathcal{P}$ maps the *solution* back to finer grids.

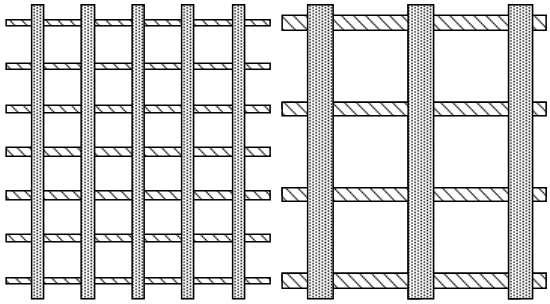

Figure 2: Multiple resolution power grids.

We limit the discussion of intergrid transfer operators to the case where grid spacing is doubled at every level. Consider two grids ω_h (fine) and ω_{2h} (coarse) with equation 2 rewritten as a system of linear equations $A_i x_i = b_i$ where i indicates which grid is being considered. To map the problem from ω_h to ω_{2h}, the restriction operator $\mathcal{R}$ is used: $b_{2h} = \mathcal{R} \times b_h$. To map the solution back, the prolongation operator $\mathcal{P}$ is used: $x_h = \mathcal{P} \times x_{2h}$. To map A_h to A_{2h} we first reduce the grid ω_h to ω_{2h} as described in the next section and then formulate A_{2h} at ω_{2h}. Once the problem is defined at the coarser grid, the *smaller* system of equations can be solved for the node voltages x_{2h}, then this solution is mapped back to the finer grid using the prolongation operator. Note that for power grid problems as defined above, $\mathcal{R} = \mathcal{P}^T$.

3.2 Grid Reduction

In the context of power grid analysis, a natural method for grid reduction is skipping every other wire while doubling wire widths to keep the total resistance constant. This results in a situation like that illustrated in Figure 2. However, typical power grids may be *irregular*, i.e. different edges may have different lengths and different seperation distances. Thus, the reduction algorithm should present a systematic mechanism for reducing any *general* grid. Furthermore, the algorithm should maintain the structure of the original grid so that it can be recursively applied until a coarse enough grid is obtained.

The major objective of the reduction algorithm is to remove as many nodes as possible while maintaining the ability to estimate voltages at the removed nodes by interpolation. The algorithm takes as input a fine grid ω_h and a list of nodes to be *kept* and produces as output a reduced grid ω_{2h} with a smaller number of nodes. The list of *kept* nodes consists of specific nodes of interest, such as corner nodes and nodes where voltage sources are located.

The algorithm makes use of certain status flags, which are explained in Table 1, to decide whether a node is kept or removed. Furthermore, these flags indicate how to interpolate

StatusFlag	Indication
N	No flag (default)
K	Kept
H	Visited Horizontally
V	Visited Vertically
R	Removed

Table 1: Meaning of status flags.

the voltage at a removed node from its kept neighbors. The grid reduction algorithm consists of three passes described as follows:

1. First Pass:

   Update each *kept* node; that is, starting from that node, go along horizontal (vertical) direction and flag all visited nodes with H (V). Flag extremities as *kept*. A node which is visited both horizontally and vertically (flagged with both H and V), is flagged as *kept*.

2. Second Pass:

   For each H (V) node, flag it as removed (R), flag its neighbors along same row (column) as *kept*, and then *update* those neighbors.

   If a node is not flagged (N), then flag it as removed (R), and flag its diagonal neighbors as *kept*. Again update the *kept* nodes.

3. Third Pass (defines interpolation):

   Voltage of a *kept* node is same as that computed at the coarser grid.

   Voltage of an H (V) node which is then flagged as R is interpolated from its row (column) neighbors' voltages.

   Voltage of an N node which is then flagged as R is interpolated from its *diagonal* neighbors which are *kept*.

The *diagonal* neighbors of a node X are defined as those nodes reached by going 2 steps from X first horizontally and then vertically or first vertically and then horizontally. For example, if node Y is the upper neighbor of node X, then the left and right neighbors of Y are diagonal neighbors of X.

The algorithm is illustrated by the irregular grid ω_h shown in Figure 3 which will be reduced to result in the grid ω_{2h}. In our implementation of the grid reduction algorithm, grid nodes are sorted according to their position on the grid from top to bottom, left to right. However, note that this is not a limitation of the algorithm which is robust enough to handle any order of the nodes.

Initially, all nodes have the default status of N except for the nodes which should be kept. In this example, these would be all the corner nodes of the grid (dashed nodes in Figure 3). A tag consisting of two fields is associated with every node of the grid. The left field indicates the status of the node after the first pass and the right field indicates the status of the node after the second pass.

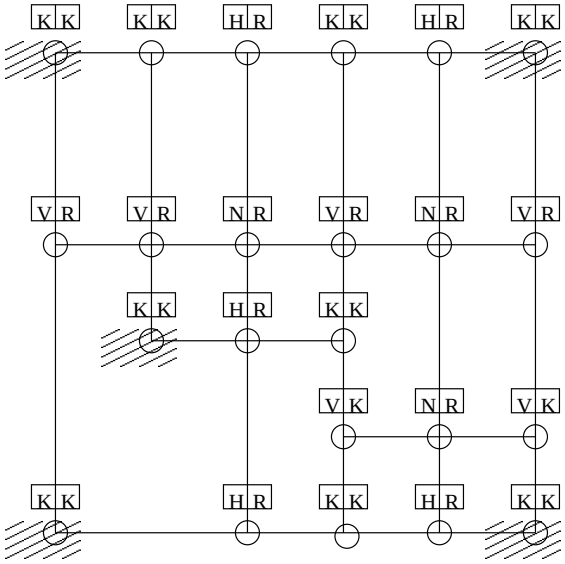

Figure 3: Irregular grid with status flags.

As shown in Figure 3, after the first pass, an edge (row or column) consisting of at least one *kept* node, has its extremities flagged as *kept*. The remaining nodes on that edge are flagged with H or V based on whether the edge is horizontal or vertical. Note that some nodes still have a status flag of N which indicates that these nodes have not been visited during the first pass. Then after the second pass, nodes with a K flag are kept while those with an R flag are removed thus resulting in the coarser grid ω_{2h}.

It remains to describe how the interpolation works. We already pointed out that the status flags indicate which neighbors of a removed node are used for interpolation. Note that the interpolation function takes into consideration the values of conductances between the nodes. So if the voltage at a removed node m is interpolated from the voltages at nodes A and B, then the *linear* interpolation function $INT()$ is defined as:

$$V(m) = INT(V(A), V(B)) = a_0 V(A) + a_1 V(B)$$

where

$$a_0 = \frac{g_{mA}}{g_{mA} + g_{mB}}$$

$$a_1 = \frac{g_{mB}}{g_{mA} + g_{mB}}$$

g_{mA} is the conductance between nodes m and A, and g_{mB} is the conductance between nodes m and B.

The reduction algorithm applied to the irregular grid given in Figure 3 indicates which nodes are kept and which are removed. Furthermore, it defines the interpolation mechanism which is illustrated by Figure 4 where the filled nodes correspond to removed nodes and the blank nodes correspond to kept nodes. The arrows from every removed node indicate which of

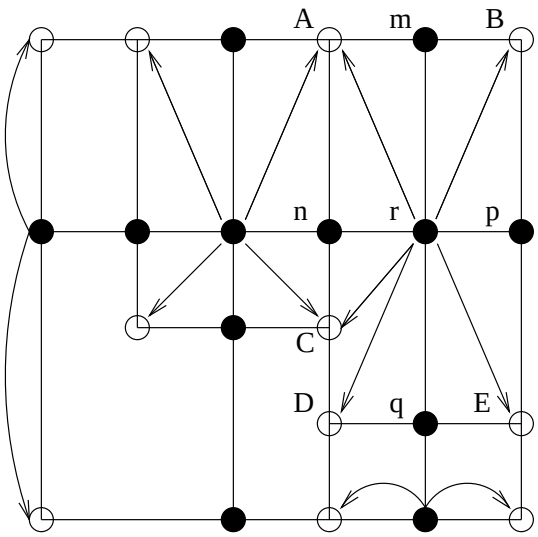

Figure 4: Interpolation from reduced grid nodes.

its neighbors are used for interpolation. For example, the voltage at the removed node r is interpolated from the voltages at its five diagonal neighbors that are kept, A, B, C, D, and E. Using the interpolation function $INT()$ defined above, the voltage at node r is related to the voltages at nodes m, n, p, and q. It follows that:

$$V(r) = INT(V(m),V(n),V(p),V(q))$$

$$V(m) = INT(V(A),V(B))$$
$$V(n) = INT(V(A),V(C))$$
$$V(p) = INT(V(B),V(E))$$
$$V(q) = INT(V(D),V(E))$$

Hence,

$$V(r) = INT1(V(A),V(B),V(C),V(D),V(E))$$

Finally, we point out that if the original grid is *regular*, then the algorithm is optimal. That is, it results in maximal reduction in the number of nodes as illustrated in Figure 5. In that case, every grid reduction results in a linear system with approximately 4x fewer unknowns and consequently 8x smaller CPU time for solution by direct sparse matrix methods.

3.3 Time Domain Analysis with Multi-Grids

In section 2 we pointed out that the fixed time step BE integration method offers large efficiency gains because it requires only one matrix inversion for all time steps. However, this efficiency comes at the cost of requiring the use of a *direct* solver. For this reason we are not able to utilize the classical MG iteration and are motivated to modify it.

Consider the case where we use direct solution methods to solve the original system of equations to get the solution which

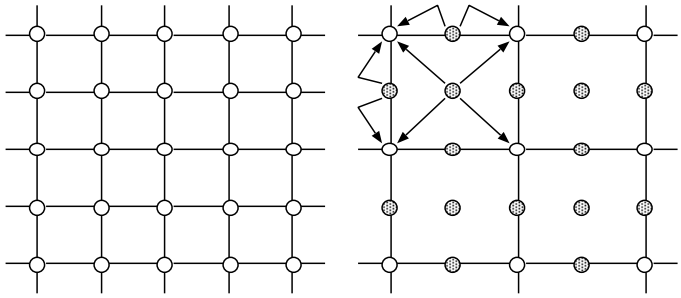

Figure 5: Basic Multigrid operator.

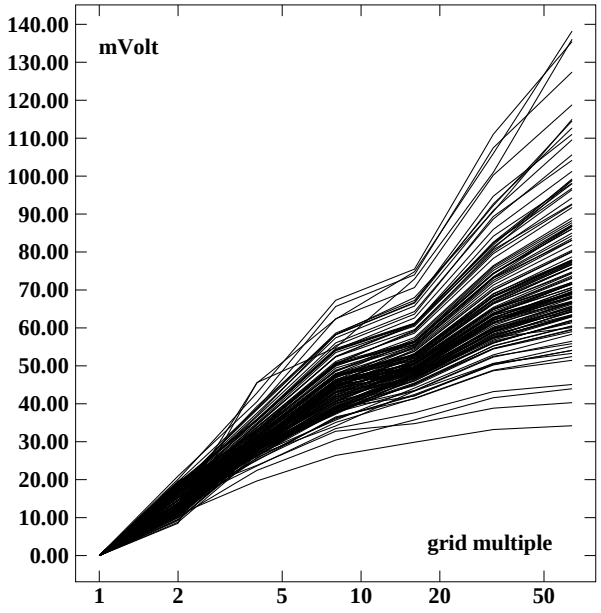

Figure 6: Multiple resolution power grids.

we denote by x^0. We then select a number n of MG-reduced systems, solve them, and use the prolongation operators to get an estimate of the full solution $x^i, i = 1 \cdots n$.

Figure 6 shows a plot of a random selection of components of the error $x^0 - x^i$ for a 66000 node power grid problem. We observe from the plot that the error is smooth and well behaved as a function of i or, equivalently, the logarithm of the spatial step size multiplier h. This stands to reason since we expect the error to be proportional to a power of h depending on the order of the interpolation formula used.

Given the above, we propose to use the values $x^i, i = m \cdots n$ and *extrapolate* to the actual solution x^0. We do this by linearly extrapolating component wise for each element of x, i.e. fitting a linear model of the form $x_i = a_i + b_i log(h)$, thus $x_i^0 = a_i$. We call the method *Multigrid Extrapolation*.

For time domain analysis, we solve for all x^i at each time point and then perform the extrapolation. As above, the LU factors of the system matrices are generated once and reused

$log_2(h)$	h	dimension	cpu time(sec)
7	128	272	2.44
6	64	628	3.25
5	32	1885	3.47
4	16	6355	10.02
3	8	23491	NA
2	4	90631	NA
1	2	355158	NA
0	1	908149	5993.41

Table 2: CPU times for large example.

for all time steps, preserving the efficiency of the fixed time step BE scheme.

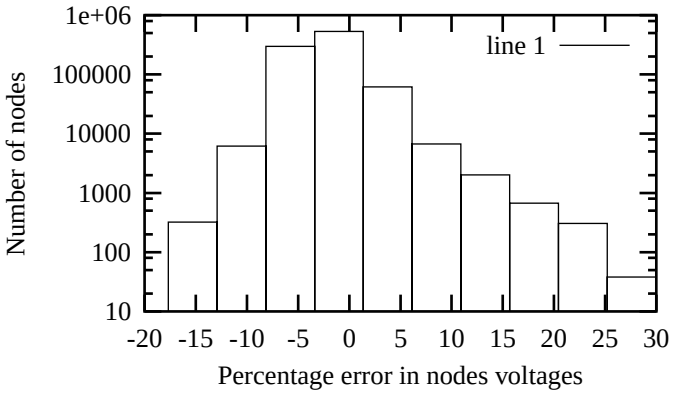

Figure 7: Error in nodes voltages.

4 Experimental results

The proposed multigrid method has been implemented and integrated into a linear simulator written in C++. All experimental results reported in this section were obtained by running the simulations on a 333MHz Pentium II machine with RedHat Linux 6.0 operating system.

The efficiency of the proposed technique is illustrated by applying it to the analysis of the power grid of a large 19M transistor PowerPC processor built in a 0.18μ CMOS process[8]. The *irregular* power grid on the top 3 metal layers consisting of approximately nine hundred thousand nodes was simulated. Several grid reductions are applied and the problem accordingly mapped to the coarser grids as explained in section 3.1. Table 2 shows the number of nodes of the grid at every level as well as the CPU times for solving the resulting linear systems at each of these grids.

The estimated solution $\tilde{x}^0$ is extrapolated from the solutions x^4 through x^7. Thus, there is no need to solve the linear systems resulting from the grids at levels 1, 2 and 3. That is why no CPU times are shown for levels 1, 2 and 3. As shown in Table 2, obtaining the solutions x^4 through x^7 requires a total of about 20 seconds while solving the original system requires about 6000 seconds. Thus, the proposed method offers a speedup of about 300x for the given example. Of course, the multigrid method involves some overhead for setting up the restriction and prolongation matrices as well as reformulating the problem at the smaller grids. But this overhead is small enough to maintain a speedup of two orders of magnitude over existing methods for power grid analysis.

Another significant advantage of the multigrid technique is low memory demand. This follows from the fact that the memory required by any solver is directly proportional to the size of the linear system being solved. Consequently, in the context of power grid analysis, the required memory would be directly proportional to the dimension of the grid. Since the proposed method maps the large grid problem to smaller dimension grids and solves the resulting smaller problems, then it is obvious that it requires less memory than existing methods. As a matter of fact, simulating the power grid of the given example with existing methods imposes the use of an *iterative* solver due to memory limitation. The multigrid method, on the other hand, utilizes *direct* solvers thus promising more speedups for transient analysis.

To verify the accuracy of the proposed technique, the exact solution x^0 is compared to the solution $\tilde{x}^0$ which is extrapolated from the solutions x^4 through x^7. The histogram of the errors in the voltages at the different nodes of the grid is shown in Figure 7. This figure shows that the error distribution approximates a normal distribution with a mean of -2.0%, a standard deviation of 3.0% and a range of -17.7% to 29.9%.

The resulting error is often quite acceptable considering that the power grid drains are rarely known to greater accuracy. Of course, it is possible to perform iterative refinement [9] to reduce the error if necessary. Furthermore, we have observed that the error is related to the geometry of the grid and thus, we are looking into efficient methods for reducing the error.

Finally, we note that the number of levels used for grid reduction introduces a tradeoff between the accuracy of the solution and the speedups achieved. We illustrate this point by solving the same example again but now using one extra level for extrapolating the solution. That is, the approximate solution $\tilde{x}^0$ is now extrapolated from the solutions x^3 through x^7 rather than x^4 through x^7.

This requires an extra overhead of 28.13 CPU seconds but the errors are reduced. The histogram in Figure 8 shows the new error distribution which still approximates a normal distribution but now has a mean of -1.7%, a standard deviation of 2.4% and a range of -13.5% to 22.6%.

5 Conclusion

An efficient PDE-like method for power grid analysis is presented. Initial results on realistic examples show speedups of one to two orders of magnitude over current methods. Note that the proposed method results in speedups for both DC as well as transient analysis. Future work involves developing

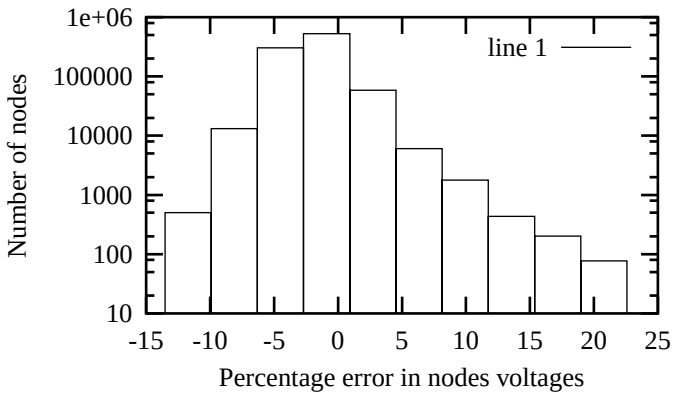

Figure 8: Error in nodes voltages (extra level used).

techniques for obtaining more accurate results by considering different interpolation methods as well as different mapping operators across the grids.

References

[1] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. 35^{th} Design Automation Conference, San Francisco, 1998.

[2] A. Dharchoudhury et. al. Design and analysis of power distribution networks in *PowerPC*^{TM} microprocessors. 35^{th} Design Automation Conference, San Francisco, 1998.

[3] Y. M. Jiang and K. T. Cheng. Analysis of performance impact caused by power supply noise in deep submicorn devices 36^{th} Design Automation Conference, New Orleans, 1999.

[4] H. H. Chen and J. S. Neely. Interconnect and circuit modeling techniques for full-chip power noise analysis. IEEE Transactions on Components and Packaging II, 1998

[5] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, University of California, Berkeley, 1975.

[6] G. Steele et. al. Full-chip verification methods for DSM power distribution systems. 35^{th} Design Automation Conference, San Francisco, 1998.

[7] W. L. Briggs. *A Multigrid Tutorial*. SIAM, 1987.

[8] P. Hofstee et. al. A 1 ghz single-issue 64b powerpc processor. ISSCC 2000.

[9] W. H. Press et. al. *Numerical Recipes in C*. Cambridge University Press, 1988.