LPGAS - Large Power Grid Analysis and Simulation

Fábio Barata fabio.barata@ist.utl.pt
IST - Technical University of Lisbon

Abstract. Power grid analysis is becoming a very challenging circuit design problem which defies the limits of computational resources – processing and memory. Current very large scale integration (VLSI) designs have already exceeded one hundred million nodes and any power grid analysis and verification problem tends to get enormous. Computer resources available are already too short to store and process the many gigabytes (GB) of data involved in the analysis. For that reason, very efficient parallelizable strategies, compatible with distributed data, are absolutely needed to solve this problem.

In this study, Block-Jacobi Preconditioned Conjugate Gradient (BJ PCG) is proposed, a popular linear system solver yet to be applied to the power grid problem. BJ PCG solves a 7.9M node power grid with 300 current sources in 5 hours in a 10 octocore cluster using 9.06GB of memory per computer.

keywords: electromigration, power grid analysis, size problems, distributed computing graphs

1 INTRODUCTION

Power grid analysis is becoming more and more a critical task to ensure the proper functioning of integrated circuits. As the number of transistors in circuits keeps increasing, the density of power distribution networks (PDN or power grids) increases manifold. As a result, analysis of current power grids in VLSI designs threatens to exceed the formidable computational power available and has developed into a considerable problem. In reality, distributed systems and modern technology take an important role in solving the problem, because one computing instance is not enough to support the computational and memory burden.

The motivation for this research was in fact to address the electromigration (EM) problem using Power grid analysis. EM is mass-electron movement in metallic interconnects. This phenomena can cause anomalies such as connection disruptions (caused by void failures and diffusive displacements) and component failure due to heating, eventually diseasing the IC. Measurement or estimation of the current density in interconnects is crucial to study the EM (an empirical model developped by J. R. Black in 1960 estimates the mean time to failure from the current density) and envision the IC lifespan. Even though it is a time-dependent case, the interest of this study is the EM DC problem. One way to address this problem is to model the power using an electric circuit equivalent where power grid connections are replaced with an appropriate electrical model. Then, in order to determine the currents in the power grid and to determine if they exceed some safety values and could lead to EM problems, one can analyse the power grid, solve for the nodes voltages in the model and compute the branch currents. Solving power grid analysis problems is therefore an essential key for EM analysis.

Attempts to solve the power grid analysis problem have brought some high quality solutions with high performance and flexibility. In [1], a distributable direct method was first applied to power grids and showed very good reference results. In [2], an iterative efficient solver based on SOR algorithm brought a lot of flexibility in distributing and scalability. In [3], another iterative solution showed excelent convergence results. These solutions make use of partitioning techniques such as Domain partitioning[1] and Block partitioning (or Geometric partitioning). Also, in [4], a technique that became very popular to VLSI circuit partitioning was presented and applied for the first time in the power grid subject, and it turns out that it can also be applied in conjunction with most of these algorithms.

Preconditioned conjugate gradient algorithms have proved to be very successful in approaching linear system problems over the years. Given the linear nature of the power grid analysis problem, a Block-Jacobi Preconditioned Conjugate Gradient algorithm is proposed in this study, a preconditioner very flexible to parallelize, and very suited for distributed data solutions.

In the next Section, the power grid model and problem formulation will be described. It will also be shown why non-distributed methods are useless for anything other than benchmark results. In Section 3, BJ PCG is shown, as well as comparison with other types of preconditioners. In Section 4, some partitioning strategies that can be used in conjunction with the state of the art power grid analysis algorithms are explained. In Section 5, the results for time and memory performances are illustrated. In Section 6, conclusions are presented.

2 BACKGROUND

For the purpose of an analysis, power grids can be modelled accurately by RC meshes connected to an integrated circuit in multiple points, in order to supply the power needed for functional blocks to work. Hence, in this simplified setting, functional blocks are modelled by current sources that demand some current from the grid. A voltage source supplies the current needed, through connections, in other multiple nodes of the grid, usually uniformly distributed along the grid. Current flip-chip (C4 bump) technology provide a large number of contact V_{DD} nodes, so a current demanded by a functional block is provided by the power supply connections in the surroundings. This is called locality.

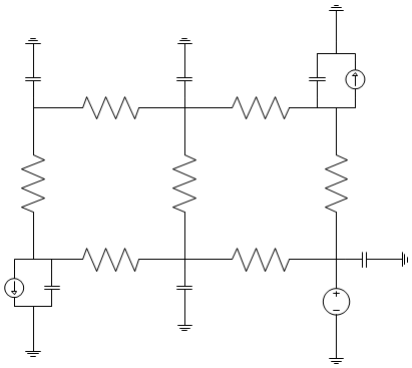

Figure 1: RC circuit mesh example with 6 nodes.

An example of a power grid model is shown in Figure 1.The power grid representation adopted in this article is:

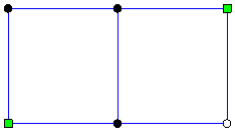

Figure 2: Power grid representation example. Current sources are connected to nodes represented in green squares. Voltage sources are connected to nodes in white circles.

Power grid DC analysis is a linear system problem. The voltage sources are transformed to multiple current sources using a Norton equivalent at the appropriate connection points. The circuit voltage reference is the voltage source amplitude, thus the linear system is given by:

$$G v = B i + B_v i_v \quad (1)$$

where: G is an NxN matrix with NZ non-zero entries, imposed by the KCL rule for each grid node (often called the KCL conductance matrix); B is an NxM incidence matrix of the current sources (ie $B_{j,k}$ = 1 iif current source k is connected to node j); i is an Mx1 vector with current source amplitudes; B_v is an NxK incidence matrix of the Norton equivalent current sources corresponding to the voltage sources and finally i_v is an Kx1 vector with Norton equivalent current sources amplitudes corresponding to the voltage sources.

The equation (1) can be split into two components, leading to:

$$G v = B i + B_v i_v \Leftrightarrow G(v_0 + \Delta v) = B_v i_v + B i \quad (2)$$

where v_0 is the solution to the biasing voltage sources and Δv is the voltage drop caused by the current requirements from the logic blocks. Equation (2) can be split by interposition into:

$$G v_0 = B_v i_v \quad (3)$$
$$G \Delta v = B i \quad (4)$$

The solution to equation (3) is easily obtained by inspection noting that the right-hand side corresponds to the biasing voltage sources. The node voltage at every grid node is therefore equal to V_{DD}. For equation (4), the voltage sources are grounded and the voltage variations is due only to the currents in the logic blocks. From now on, determining Δv is the goal of the power grid analysis.

However simple the problem might seem, the number of nodes in a real power grid is enormous (tens of million nodes), and memory resources are easily exceeded in non-distributed solutions:

Grid Nodes	Current Sources	Time	Memory peak
62k	90	1.91s	130MB
404k	120	14.4s	817MB
1.2M	180	1070s	5.59GB

Table 1: Time and Memory spent using CHOLMOD[5] library for various power grid sizes

3 SOLUTION

The direct methods such as Cholesky factorization and Domain decomposition provide very efficient strategies to tackle the power grid problem. However, they do not provide scalability, and as a good approximation to the result suffices to provide the information needed for the power grid analysis, iterative schemes tend to be a little more efficient in filling the needs. In this study, a very popular linear system solver – BJ PCG – is proposed to be applied to the power grid analysis problem.

3.1 Block-Jacobi Preconditioned Conjugate Gradient (BJ PCG)

The conjugate gradient algorithm is a widely known iterative algorithm for solving systems of linear equations, in which matrix A in linear system Ax = b is symmetric and positive definite. In order to accelerate the rate of convergence, a technique called preconditioning is often used. This technique consists in a transformation leading to the system MAx = Mb. The rational for this procedure is that solving this system is easier, i.e., requires less iterations, than solving the original system, at little additional cost per iteration. This results in an operation applied to the residue r that approximates a solution z in Az = r by solving Mz = r in a cheaper process. The ideal preconditioning operation would give the exact solution z in Az = r and the preconditioned conjugate gradient (PCG) would reach the solution in the first iteration. Obviously, this requires a direct solution of the original system and is therefore fruitless.

Over the years many preconditioners were developed for a variety of problems. The most basic preconditioner is perhaps the so-called Jacobi preconditioner or diagonal preconditioner. This preconditioner works particularly well for diagonally dominant matrices, since its approximation grows better the more preponderant the diagonal values of A are. Another popular option is the SSOR (Symmetric Successive Over-Relaxation) preconditioner, also a relaxation-type preconditioner that uses more of the matrix information as it is derived from a straightforward decomposition of the A matrix into lower, upper and diagonal blocks and is based on the SSOR method[6]. Both preconditioners discussed so far are easy to parallelize and distribute in terms of both initialization and application. For example, the SSOR preconditioner has the same non-zero pattern as the G matrix, hence solve can be parallelized by recurring to a technique very similar to the distributed matrix-matrix multiplication.

Another commonly used option is the set of the so called incomplete factorization (ILU or IChol) preconditioners. The IChol preconditioners consist in computing a partial Cholesky decomposition and keeping only a subset of the elements in the factors. These preconditioners are known for being volatile in structural and numerical ways, providing a good trade-off between initialization and convergence rate. The choices used in this work are the IChol(0) (no fill-in incomplete factorization) and ICholT (incomplete factorization with thresholding). The ICholT preconditioners usually provide very good results, but are difficult to parallelize. .

Among the widest-known preconditioners, Block Jacobi (BJ)[6] provides very simple and effective ways of distributing the algorithm. The BJ preconditioner consists in subdividing the matrix into smaller parts with resort to a partitioning algorithm (in which the partitioning options come forward) and computing the solution for each of the domains in parallel for each iteration. In other words, this preconditioner computes factorization and solution of smaller parts of the grid (just like BI algorithm), and uses them to approximate the whole solution, making use of the locality nature of the power grids.

In Algorithm 3.1, the BJ PCG algorithm pseudo-code is presented.

Algorithm 3.1: Block-Jacobi Preconditioned Conjugate Gradient

Inputs: Part = Partition sets, thresh = convergence threshold, **Outputs:** Solution

1) **for each** p **in** Part
   1) factorize G[p;p] matrix for preconditioning
2) **set** starting solution **to** 0
3) $r = B$
4) **for each** p **in** Part
   1) **Let** z_p and r_p be submatrices of z and r with p row set and all columns
   2) **Solve** $G[p;p]z_p = r_p$ using previously computed factorization
5) $q = z$
6) **for** a **from** 1 **to** M
   1) let z_a , r_a , d_a and q_a be the a columns of z, r, d and q
   2) $d_a = z_a^T r_a$
7) **Loop until break**
   1) **for** a **from** 1 **to** M
      1) let z_a , r_a , q_a and x_a be the a columns of z, r, q and x
      2) $Gqa = G\,q_a$
      3) $\alpha = d_a / \left(q_a^T Gqa\right)$
      4) $x_a = x_a + \alpha q_a, r_a = r_a - \alpha q_a$
   2) $\|r\|_\infty <$ thresh
      1) break
   3) for each set in Part
      1) **Let** z_p and r_p be submatrices of z and r with p row set and all columns
      2) **Solve** $G[p;p]z_p = r_p$ using previously computed factorization
   4) **for** a **from** 1 **to** M
      1) let z_a , r_a and q_a be the a columns of z, r and q
      2) $d_a^{next} = z_a^T r_a$
      3) $q_a = z_a + \frac{d_a^{next}}{d_a} q_a$
      4) $d_a = d_a^{next}$

In Figure 3, a comparison between various widely known preconditioner options[6] is illustrated. Only Incomplete Cholesky factorization is on par with Block Jacobi preconditioner, but does not provide the same distributing capabilities. In fact,

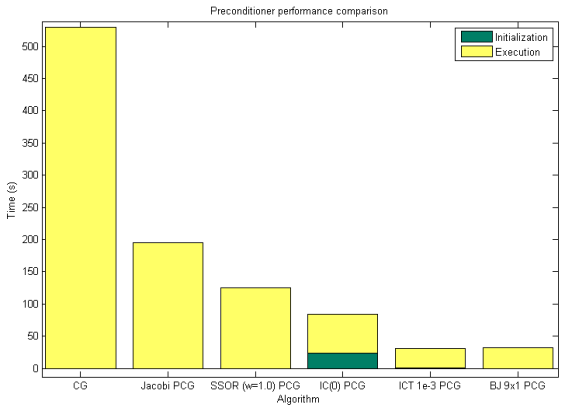

Figure 3: Preconditioner comparison - Time to reach 10^-6 error for a 62k node IBM benchmark power grid with 50 current sources (computed using MATLAB)

Incomplete factorization preconditioners are very difficult to parallelize.

4 PARTITIONING

In order to achieve parallelized solutions and distributed data, the division of the power grid must be accomplished. In this section, graph partitioning (sometimes called clustering) techniques will be studied. Graph partitioning application to power grids has two main purposes: to distribute the resource demands among computing instances; to maximize the similarity of the inner solution (obtained using only inner-partition nodes) and real solution (of the partition nodes, obtained by simulating the whole circuit). That said, the relationship between sets is often measured as the sum of the branch weights interconnecting the sets (which are branch conductancess for the case of power grids). Also, the most attractive partitioners are those which balance the size of each set, avoiding very large sets that lead some instances to drudge while the remaining are idle, limiting the advantages of partitioning.

4.1 Geometric Partitioning

The geometric interpretation is an essential procedure in graph partitioning. The power grids may not be regular structures, but they have geometric patterns. Hence, to partition the grid using geometric information (node position) makes sense and it turns out to be a very effective strategy that provides good results.

The geometric partitioning (GP), also called area-based partitioning and block partitioning, is not particularly hard to achieve: Cut the grid uniformly on the single layer directions x and y.

The GP algorithm results in the following:

Inputs: cd = node coordinates, nx = number of cuts x, ny = number of cuts y, **Outputs:** Partition sets
1) xmin = min cd_x, xmax = max cd_x
2) ymin = min cd_y, ymax = max cd_y
3) width = xmax - xmin, height = ymax - ymin
4) **for** i **from** 0 **to** nx-1
    1) xlow = xmin + i*width/nx, xhigh = xlow + width/nx
    2) **for** j **from** 0 **to** ny-1
        1) ylow = ymin + j*height/ny, yhigh = ylow + height/ny
        2) $S_{i*ny+j} = cd_x \in [xlow, xhigh] \cap cd_y \in [ylow, yhigh]$

4.2 Ratio cut Partitioning

The previous graph partitioning algorithms are based on geometric postulations Geometric or functional block information might be inaccessible or inaccurate, jeopardizing the partitioning performance substantially. In that event, a geometrically independent approach is desired, and one that optimizes the size of all partitions and the interconnecting conductances. In this study, a partitioning algorithm, commonly used in the various VLSI-related areas, is proposed: Ratio cut partitioning.

The Ratio cut is a graph partitioning strategy which targets the following optimization problem[7] (metric):

$$\min_{[V_1 \ldots V_n]} \frac{1}{\sum_k |V_k|(|V|-|V_k|)} \sum_{\substack{i \in V_p \\ j \in V_q \\ p \neq q}} W_{i,j} \qquad (5)$$

where:

V_i – Partition set I, $|V_i|$ is the set size

$W_{i,j}$ – Weight on branch which connects node i to j

In short, Ratio cut is a strategy that attempts to optimize set partitioning such that the sum of interconnecting weights over the product of partition sizes is minimum. The number of partitions is either fixed or variable, depending on the algorithm and/or the application.

The Ratio-Cut Partitioning algorithm can be described in the following procedure:

Inputs: m = number of partitions, W = weighted adjacency matrix, **Outputs:** Partition sets
1) **Compute** diagonal degree matrix
2) **Calculate** the Laplacian L = D-W
3) **Estimate** the Fiedler vector v using the Lanczos algorithm
4) **Sort** v values ascending or descending
5) **Compute** the most profitable m-1 cuts in v by testing all cutting possibilities and using the ratio metric (5)
6) Cluster the graph from the cuts found

5 RESULTS

The power grid analysis is a very expensive process in terms of memory and processing complexities. In this section, BJ PCG will run for various benchmarking and artificially created power grids.

5.1 Environment

The results in this sections make use of two different architectures:

1) Multithreaded architecture:
   1. Multi-core processor with 4 cores – Intel Xeon E5410 2.33GHz
   2. RAM Size: 24GB
   3. Operative System: Linux Fedora 13
2) Distributed architecture:
   1. Cluster with 10 Multi-core processors with 8 cores each – Intel Xeon E5504 2.00GHz
   2. RAM Size: 32GB per processor
   3. OperativeSystem: Linux CentOS 5.4 (Final)

5.2 Power grids

The power grids used to test the BJ PCG performance were of two types: IBM provided

benchmark power grids (http://dropzone.tamu.edu/~pli/PGBench) and artificially created power grids. As the IBM provided power grids are very small in comparison to the target (of 300Million nodes), artificially created power grids with similar characteristics were generated.

5.2.1 IBM benchmark power grids

Real power grids used are IBM benchmark power grids. The V_{SS} part is eliminated and the zero resistance branches are simplified (terminal nodes are fused). The number of current sources is arbitrarily chosen for problem escalation.

Power grid	Nodes	Current sources
IBMPG2	61677	90
IBMPG6	403915	130
IBMPG6-2	403915	300

Table 2: Power grid IBM benchmarks

5.2.2 Artificially created power grids

Artificial power grids are structured. They are generated according to rules such as: each pair of layers (in the z-axis) has the same density, but lower layer pairs have an higher density than upper layer pairs; total resistance of the power grid is constant; each layer only has resistors in either x-axis and z-axis or y-axis and z-axis.

5.3 Comparison between partitioning algorithms

In this section, the partitioning strategies previously discussed will be compared for some power grids in a multithreaded architecture (Section 5.1).

In this section, the conditions are:

1) Number of partitions m
2) BI SOR w = 1.65
3) Acceptable residue error $|r|_{max} = 10^{-3}$

Tables 4 and 5 show comparison among various partitioning strategies.

Power grid	Nodes	Current sources
ART2	782978	160
ART3	782978	300
ART4	1197252	300
ART5	1197252	500
ART6	1197252	750
ART7	1197252	1000
ART8	1197252	1250
ART9	1197252	1500
ART10	1197252	1750
ART11	1197252	2000
ART12	2918100	300
ART13	4998932	300
ART14	7982598	300

Table 3: Artificially created power grids

Power Grid	**m**	**BJ PCG**
IBMPG1	8	8 / 0.095s
IBMPG2	8	21 / 9.25s
ART1	8	33 / 56.5s
IBMPG6	8	47 / 133s

Table 4: Results for BJ PCG solver with GP algorithm (iterations / time)

Power Grid	**m**	**BJ PCG**
IBMPG1	8	11 / 0.114s
IBMPG2	8	26 / 21.4s
ART1	8	46 / 141s
IBMPG6	8	24 / 68.9s

Table 5: Results for BJ PCG solver with Ratio cut algorithm (iterations / time)

The Geometric Partitioning is the most reliable partitioning algorithm to use in conjunction with BJ PCG, with the only exception being IBMPG6.

5.4 Distributed BJ PCG results

In this section, a distributed version BJ PCG will tackle increasingly difficult power grids on a distributed architecture (Section 5.1).

The next table shows some quick examples of how much is the distributed BJ PCG worth in terms of processing (with 8 partitions per computer):

Number of computers	Time (s)	Memory spent per unit (GB)	Speed-up
1	424	4.72	0%
2	381	2.37	11%
3	265	1.61	60%
4	185	1.20	129%
5	175	0.95	142%
6	155	0.79	174%
7	188	0.66	126%
8	139	0.60	205%
9	159	0.57	166%
10	125	0.52	239%

Table 6: Speed-up and memory results for distributed processing over 1 to 10 computing instances, simulating IBMPG6-2 power grid benchmark (404k node with 300 current sources)

It is evident that distributing the data will ease the memory burden over each computer, but the previous table showed that even the processing cost benefits from the distribution and provides some good speed-ups. Sometimes the speed-up may drop a bit (case 7 and 9 computers in the previous sample) due to where the power grid was cut if it has some irregularities.

Table 7 shows results for increasing number of nodes.

Power Grid	Number of Iterations	Time (s)	Memory used/ computer (GB)
IBMPG6-2	80	147	0.53
ART3	127	494	0.96
ART4	160	1036	1.47
ART12	209	4440	3.60
ART13	211	8309	6.18
ART14	248	17660	10.03

Table 7: Block-Jacobi Preconditioned Conjugate Gradient on a 10 octocore cluster results in iterations, time and memory peak per computer

In Figure 4, 5 and 6, the evolution of the previous results can be seen over the number of nodes. Memory is heavily dominated by the residue matrix (r = B-G*s) with dimensions N*M (number of nodes x number of current sources) and which becomes dense after just only a few iterations. So, it is not surprising that the evolution of memory spent is linear with the number of nodes (and will be with the number of current sources too). The projected complexity models the time curve fairly well.

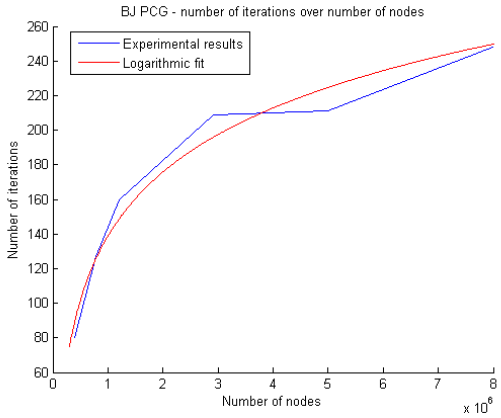

Figure 4: BJ PCG results: number of iterations over number of nodes

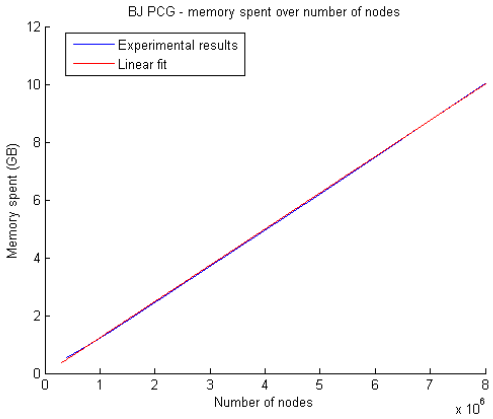

Figure 6: BJ PCG results: memory spent over number of nodes

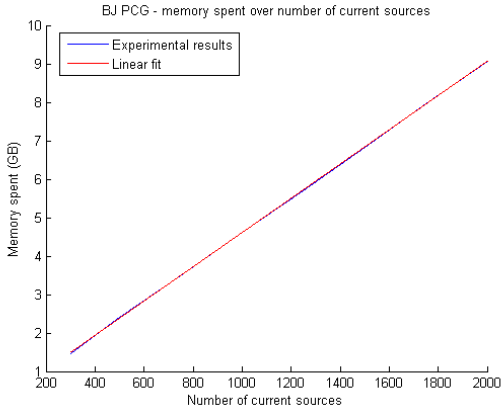

Figure 5: BJ PCG results: memory spent over number of current sources

Next, the algorithm will run for increasing number of current sources:

Power Grid	Number of Iterations	Time (s)	Memory used/ computer (GB)
ART4	160	1036	1.47
ART5	158	2141	2.41
ART6	161	3589	3.48
ART7	159	5062	4.61
ART8	161	6266	5.71
ART9	159	7838	6.84
ART10	159	8911	7.95
ART11	160	10660	9.06

Table 8: Block-Jacobi Preconditioned Conjugate Gradient increasing node number on a 10 octocore cluster results in iterations, time and memory peak per computer

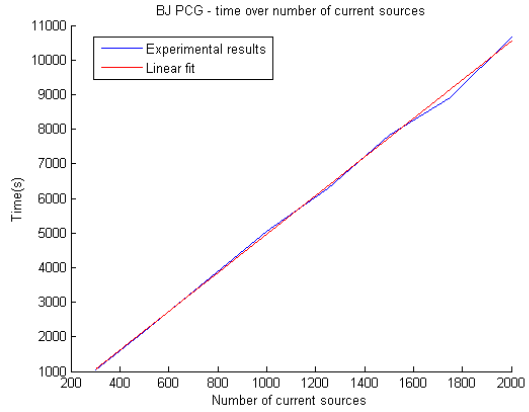

Figure 7: BJ PCG results: time over number of current sources

As expected, the number of iterations does not vary considerably, since the grid is the same and is structurally regular, solving does not depend much on the chosen nodes and all pretty much take the same number of iterations to converge. Also, time and memory complexity are approximately linear with the number of current sources (since factorization times are constant). In , the curves and linear fits can be seen.

The memory resources rise very quickly with the number of nodes and current sources. However, if the right-hand side is made in blocks, the memory drops substantially without increasing the processing times much, and actually decreasing for a low number of blocks. The following table shows the results for varying the number of blocks in ART7 (1.2M node power grid with 1000 current sources):

Number of blocks	Time (s)	Memory used per computer (GB)
1	5062	4.61
5	3212	0.99
10	3395	0.53
20	3870	0.31
40	5339	0.19
60	6145	0.16
100	5270	0.13
1000	40770	0.09

Table 9: Block-Jacobi Preconditioned Conjugate Gradient increasing number of blocks (of a total of 1000 current sources) on a 10 octocore cluster results in iterations, time and memory peak per computer. Please note that 1000 is not dividable by 60

6 CONCLUSIONS

In this work, the power grid analysis formulation was used to determine the voltage values of power grid nodes and indirectly solve the EM problem. For these cases, the problem requirements exceed the limits of what a single computer can do, and parallel solutions compatible with distributed data had to be applied.

Block-Jacobi Preconditioned Conjugate Gradient solved a 7.9M node power grid with 300 current sources in 5 hours in a 10 octocore cluster using 9.06GB of memory.

7 REFERENCES

[1] Quming, Z., Sun, K., Mohanram, K., Sorensen, D. C., *Large power grid analysis using domain decomposition*, In Proc. DATE '06, 2006, pp 27-32

[2] Zhong, Y. and Wong, M.D.F., *Fast Block-Iterative Domain Decomposition Algorithm for IR Drop Analysis in Large Power Grid*, In Proc. 11th ISQED, 2010 Mar., pp. 277-283

[3] Zeng, Z., Li, P. and Feng, Z., *Parallel Partitioning Based On-Chip Power Distribution Network Analysis Using Locality Acceleration*, In Proc. 10th ISQUED, 2009, pp. 776-781

[4] Sheehan, B., *Realizable reduction of RC networks*, IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems, 2007 Aug., pp. 1393-1407

[5] Chen, Y., Davis, T. A., Hager, W. W. and Rajamanickam, S., *Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, v. 35, issue 3, 2008 Oct.

[6] Saad, Y., *Iterative methods for Sparse Linear Systems,* 2000, Jan. 3rd

[7] Hagen, L. and Kahng, A.B., *New Spectral Methods for Ratio Cut Partitioning and Clustering*, IEEE Computer-Aided Design of Integrated Circuits and Systems, v. 11, 1992 Sep., pp. 1074-1085