GPU friendly Fast Poisson Solver for Structured Power Grid Network Analysis*

Jin Shi[1], Yici Cai[1], Wenting Hou[2], Liwei Ma[2], Sheldon X.-D. Tan[3], Pei-Hsin Ho[2], Xiaoyi Wang[1]

[1] EDA Lab, Computer Science Department, Tsinghua University, PRC [2] Synopsys. Inc

[3] Electrical Engineering Department, University of California, Riverside, CA

shi-j03@mails.tsinghua.edu.cn, caiyc@tsinghua.edu.cn, wthou@synopsys.com, lwma@synopsys.com, stan@ee.ucr.edu, pho@synopsys.com, wangxiaoyi00@mails.tsinghua.edu.cn

ABSTRACT

In this paper, we propose a novel simulation algorithm for large scale structured power grid networks. The new method formulates the traditional linear system as a special two-dimension Poisson equation and solves it using an analytical expressions based on FFT technique. The computation complexity of the new algorithm is *O(NlgN)*, which is much smaller than the traditional solver's complexity $O(N^{1.5})$ for sparse matrices, such as the SuperLU solver and the PCG solver. Also, due to the special formulation, graphic process unit (GPU) can be explored to further speed up the algorithm. Experimental results show that the new algorithm is stable and can achieve 100X speed up on GPU over the widely used SuperLU solver with very little memory footprint.

Categories and Subject Descriptors

B.7.2 [**INTEGRATED CIRCUITS**]: Design Aids – *graphics*, *layout*, *placement and routing*, *simulation*, *verification*.

General Terms

Algorithms, Design, Performance, Verification

Keywords

P/G network, Fast Poisson Solver, GPU

1. INTRODUCTION

Reliable on-chip power delivery is one of the major challenges for 90nm and below silicon technology. As the changes in geometry and physical aspects complicate the on-die power/ground (P/G) grid design, P/G grid simulators have to solve a very large scale linear dynamic system to compute the voltage drop distribution.

Many existing works have been proposed to address P/G grid simulation challenges in the past [1-4]. The mainstream solving algorithms can be classified into two classes: 1) direct solver 2) iterative solver. The direct solvers are robust and the resulting factor matrixes are reusable. The iterative solvers are more memory efficient. However, both types of solvers also have problems. For direct solvers, owing to the fill in the calculation of factor matrix, the memory usage of these algorithms tends to be very large. For iterative solvers, the instable performance of preconditioner has been widely criticized. Because today's P/G grids are highly regular and structured, regularity can be explored to speed up the simulation. Work [5] gives out a partition method for large grid simulation based on locality property of multi-layer P/G grids under the flip-chip package. The "grid shell" based algorithm in [5] presents a parallel scheme to solve large scale structured P/G grid. And this concept is used successfully to accelerate the speed of power grid synthesis via a macro model technique [6]. The limitation of this technique is that it just can be applied to a grid with flip-chip package. For grids with wire-bond package, the locality characteristic does not exist. Research work [7] introduces the pattern based preconditioned conjugate gradient (PCG) method to use pattern information of structured grids. However, due to its iterative nature, the stability of the preconditioner still remains a problem.

Recently, the parallel computation power of GPU increases rapidly. GPU is suitable for calculating computation intensive problems. Existing works have explored the GPU for solving linear systems, such as [8-10]. However, typically well-optimized classical solvers for P/G simulation are difficult to be parallelized due to communication and synchronization costs. So it is hard to fully unleash power of GPU to accelerate the traditional P/G grid simulation. In this paper, we present a novel simulation algorithm for structured on-chip P/G grids. This algorithm explores the characteristic of structured P/G grids and formulates the linear system as a two dimension Poisson equation. Because two dimension Poisson equation has analtical solution and can be solved efficiently using Fast Fourier Transformation (FFT) technique, the computation complexity can be reduced to *O(NlgN)*, which is compared favorably with the traditional solving techniques, such as SuperLU and the PCG method, whose computation complexity is about $O(N^{1.5})$. Further, as the related 2D FFT calculation can be easily calculated in parallel using GPU, the final speed up ratio over SuperLU can exceed 100X. Also, experimental results show that due to the use of analytical solution, the new solver is as stable as direct solvers and only consumes small memory without suffering the fill-in problem.

Our paper is organized as follows: Section 2 briefly introduces the background of P/G grid simulation; Section 3 introduces the concept of Poisson Block in both flip-chip package and wire-bond package; Section 4 introduces the formulation of Poisson Equation and its analytical solution; in section 5, we introduce how to handle boundary condition using an iterative algorithm; Section 6 introduce how to accelerate the original solver under GPU environment; Section 7 gives out the experimental results and analyzes the performance of the new algorithm. Finally, section 8 concludes the whole paper.

DAC'09, July 26-31, 2009, San Francisco, California, USA

*This work is supported in part by "The National Science and Technology Major Projects No.2008ZX01035-001-05" and in part by "The National Natural Science Foundation of China (NSFC) No.60776026 and No.60828008"

2. P/G grid Simulation Background

For simplification, we only introduce the formulation of resistive P/G grids here. For inductance and capacitance parasitics, the complete RLC model of a P/G grid can be found in [1]. Also, by using the back-Euler or the trapezoidal-Euler method, the inductors and capacitors can be discretized as resistors and equivalent absorbing current sources at each time stamp. Thus the algorithms proposed in this paper can still be applied RLC P/G grids.

For a P/G grid, a topology adjacent matrix $\boldsymbol{A}$ can be obtained. Each column of matrix $\boldsymbol{A}$ represents an edge in the grid. For a specific column, an edge starts from node i to node j (named e_{ij}), will stamp integer number 1 in the i^{th} row and -1 in the j^{th} row of $\boldsymbol{A}$. Supposing the grid has m rows and n columns, that is total $N = m \times n$ nodes, and E edges, then matrix $A \in R^{N \times E}$. Considering each edge as a resistor which has conductance g_i, then we can use a diagonal matrix $g \in R^{E \times E}$ to represent all resistors in the grid.

Giving any current distribution (using a vector $i \in R^N$ to describe the absorbing current at each node), in order to determine the voltage drop at each node (a vector $x \in R^N$), a linear system which satisfies the Kirchhoff's current law and voltage law can be built as following:

$$Mx' = i',\ M = BgB^T \quad (1)$$

In equation (1), matrix $\boldsymbol{B}$, vector x' and i' can be obtained by deleting all the pad related rows in matrix $\boldsymbol{A}$, vector x and vector $\underline{i}$. Thus, if there are total p pads in the grid, then $B \in R^{(N-p)\times E}$, and $M \in R^{(N-p)\times(N-p)}$, $x', i' \in R^{N-p}$. Due to the sparse characteristic of matrix $\boldsymbol{B}$, matrix $\boldsymbol{M}$ is also a sparse matrix. Furthermore, it is still a definite matrix because of the element fill in pattern [1]. In order to solve the linear system (1), sparse direct solver such as SuperLU [11] has been widely used in industry tools for its efficiency and stability. Also, iterative solver, such as preconditioned conjugate gradient (PCG) method [12] was also applied to solve this linear system. As mentioned in the introduction, both methods have their own problems. To mitigate their problems and further improve the performance of these two methods, we need to explore parallelization schemes and unleash the parallel computing power of GPU.

3. Poisson Block in P/G grid

Research work [5] presents an excellent scheme to divide a large scale P/G grid into small blocks, which can be solved completely and separately without losing any accuracy. It uses a special characteristic of P/G grid with flip-chip package named "locality". "Locality" means that the absorbing current sources only can trigger voltage drop within a local area around it. And beyond this area, the triggered voltage drop will attenuate to zero very fast. However, this property only exists in P/G grids with flip-chip package, because pads are distributed evenly in the grid, and they act as strong current drain points, which prevent the current to go far away from its source point. For P/G grids with wire-bond package, the situation is quite different. Currents have to reach the boundary of the grid to meet the drain points (pads). Thus locality based division cannot be directly applied to solve wire-bond based P/G grids.

Here we extend the locality concept to Poisson Block concept so that we can handle it for both packaging types. A Poisson Block stands for a special area in the grid which satisfies three conditions: 1) the boundary of this area has zero voltage drop. 2) pad nodes only locate in block boundary 3) the internal nodes only can be attached to absorbing current sources or be floating. Similar to [5], because of the independent nature of Poisson Block, each Poisson Block can be solved completely separately. Further, if we look up industrial designs, we can find that lots of cases can be treated as Poisson Blocks. Blow we list some examples to show practical Poisson Blocks. First, a full P/G grid with wide boundary core-ring and assembled with wire-bond technique can be treated as a Poisson Block. Also, the P/G grid inside an IP block with a wide core-ring can also be treated as a Poisson Block. This is because the wide core-ring makes the voltage drop at boundary nodes to be very close to zero, thus the first rule of Poisson Block can be satisfied. For the second and the third definition of Poisson Block, they can be satisfied naturally under wire-bond package type.

Another case for Poisson Block is under flip-chip package model. As the locality property shows, local current sources just trigger voltage drop within a finite area. Thus, if we enlarge the area to a proper size, then the voltage drop at boundary nodes will be close to zero too. Then the first condition of Poisson Block is met. Furthermore, if we treat the pads inside the block as special boundary nodes instead of the internal nodes by using a special algorithm introduced in Section 5, the last two conditions of Poisson Block can be also met. So the local area in a P/G grid with flip-chip package can also be treated as a Poisson Block.

For an example, we place 5 current sources around the center node of a 50×50 grid with uniformly distributed pads. Figure 1 shows the voltage drop distribution triggered by these current sources. In this figure, we can see that the area contains dominant voltage drop is around the center part of the grid. Also, voltage drop decreases as the distance between the center nodes increases. So if we treat the area bounded by the black dashed rectangle as the local area, we can find that the voltage drop at the boundary is close to zero. Also, beside the nature boundary nodes given by the dashed rectangle, we have to add the pad nodes inside the local area as new boundary nodes. Because the voltage drop on pad node is zero too, adding the pad nodes to the boundary will not break the first condition of Poisson Block while it also satisfies the second condition of Poisson Block. Therefore, an extended local area with internal pad nodes satisfies all the conditions of Poisson Block. In the following section, we can see how we can build our new algorithm under the definition of Poisson Block.

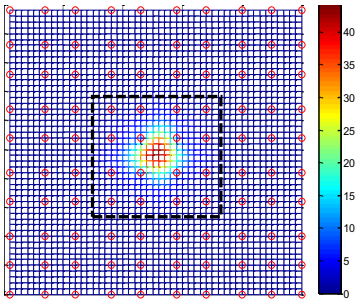

Figure 1. A Poisson Block Case: local area in flip-chip assembled P/G grid (voltage drop unit is milli-volt), red circles stand for pads

4. New Circuit Formulation for Regular Grid and analytical Solution

The classic formulation of P/G grid tries to use a sparse matrix $\boldsymbol{M}$ as in equation (1) to describe the whole linear system. In our new

formulation for regular P/G mesh circuits, we use a dense matrix to describe the relationship between current and voltage drop distribution. Before we introduce our detail formulation, we walk through a simple example to show the basic idea. Figure 2 shows a 3×4 grid. Each node in the grid is numbered. The voltage value of each node is marked as u_1 to u_{12}, and independent current sources i_1 to i_{12} are attached to these nodes.

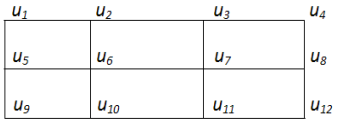

Figure 2. A small grid to show the formulation

We suppose each edge of the grid to have the same resistance r because of the grid regularity. Then we can use a dense matrix $\boldsymbol{U}$ to describe the voltage distribution as shown in equation (2). The relationship of current and voltage distribution which is controlled by Kirchhoff current law can be represented as equation (3).

$$U=\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \\ u_5 & u_6 & u_7 & u_8 \\ u_9 & u_{10} & u_{11} & u_{12} \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -1 & \end{bmatrix}\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \\ u_5 & u_6 & u_7 & u_8 \\ u_9 & u_{10} & u_{11} & u_{12} \end{bmatrix}+$$
$$\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \\ u_5 & u_6 & u_7 & u_8 \\ u_9 & u_{10} & u_{11} & u_{12} \end{bmatrix}\begin{bmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -1 \end{bmatrix}=r\begin{bmatrix} i_1 & i_2 & i_3 & i_4 \\ i_5 & i_6 & i_7 & i_8 \\ i_9 & i_{10} & i_{11} & i_{12} \end{bmatrix} \quad (3)$$

For a more complicated grid, we can get a conclusion that if the grid is regular (metal rails share the same width) and rails are distributed evenly cross the die area (most P/G grid is highly regular), then the Kirchhoff current law can be represented by a matrix equation as shown in (4):

$$T_1U+UT_2=rI \quad (4)$$

Supposing the P/G grid is an $m\times n$ grid, then in (4), $I\in R^{m\times n}$ represents the current distribution in the grid while $U\in R^{m\times n}$ represents the voltage distribution. $T_1\in R^{m\times m}$ and $T_2\in R^{n\times n}$ are tri-diagonal matrices similar to the two matrices used in (3). If we define voltage drop matrix $V=vcc\cdot E-U$ ($\boldsymbol{E}$ is a unit matrix while vcc stands for the supply voltage), then equation (4) can be rewritten as (5):

$$vcc\cdot T_1\cdot E+vcc\cdot E\cdot T_2-T_1V-VT_2=rI \quad (5)$$

From (3) we know that the sum of row of $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$ is zero, then we will have $T_1E=ET_2=0$. So (5) can be simplified to (6):

$$-T_1V-VT_2=rI \quad (6)$$

Equation (6) gives a new formulation to describe the voltage distribution under regular P/G grid. To make the formulation practical, we still need to make some changes. First, equation (6) is similar to a two-dimension Poisson Equation which has analytical result [13]. If we can change the form (6) to the form of Poisson Equation, it will make the problem easy to solve. Poisson Equation has coefficient tri-diagonal matrix $\boldsymbol{P}$ which satisfies the following form as shown in (7). By adding diagonal matrix $\boldsymbol{d_1}$ and $\boldsymbol{d_2}$ to $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$ respectively, we can make the left hand side of equation (6) to match the form of Poisson Equation, as shown in equation (8):

$$P=\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \quad (7)$$

$$(d_1-T_1)V+V(d_2-T_2)=rI+d_1V+Vd_2$$

$$d_1=d_2=\begin{bmatrix} 1 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & 1 \end{bmatrix} \quad (8)$$

Because there are only two nonzeros in matrix $\boldsymbol{d_i}$, if we analyze the matrix d_1V and Vd_2 in the right hand side of equation (8) carefully, we can see that the nonzero elements in d_1V and Vd_2 just represent the boundary elements in matrix $\boldsymbol{V}$. Remember that the boundary voltage drop of a Poisson Block is equal to zero, so if we apply equation (8) to a Poisson Block, it can be further simplified to (9), which is a real two-dimension Poisson Equation.

$$P_1V+VP_2=rI \quad (9)$$

Furthermore, we can derive more general equation for the grids having adjacent two metal layers with different routing widths. The derivation of this equation is very similar to (9) as shown in (10). In (10) r_x and r_y represent the resistance of each metal segment in a horizontal routing layer and a vertical routing layer respectively.

$$\frac{P_1V}{r_x}+\frac{VP_2}{r_y}=I \quad (10)$$

The interesting thing about Poisson Equation is that we have the analytical eigen decomposition of tri-diagonal matrix $\boldsymbol{P}$ as shown in (11):

$$P_1=z_1\Delta_1z_1^T,\ P_2=z_2\Delta_2z_2^T \quad (11)$$

In equation (11), $\boldsymbol{z_1}$ is an $m\times m$ symmetry orthogonal matrix while $\boldsymbol{z_2}$ is an $n\times n$ symmetry orthogonal matrix given by (12).

$$z_1(i,j)=\sqrt{\frac{2}{m+1}}\sin(\frac{i\cdot j\cdot\pi}{m+1}),\ z_2(i,j)=\sqrt{\frac{2}{n+1}}\sin(\frac{i\cdot j\cdot\pi}{m+1}) \quad (12)$$

Also, Δ_1 and Δ_2 are two diagonal matrix given by (13)

$$\Delta_1(i,i)=2(1-\cos(\frac{i\cdot\pi}{m+1})),\ \Delta_2(i,i)=2(1-\cos(\frac{i\cdot\pi}{n+1})) \quad (13)$$

Let $V=z_1\cdot X\cdot z_2^T$ (X is a middle result which is very useful to get the final close form expression), substitute (11) into (10) with the definition of $\boldsymbol{V}$, remember that matrix $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ are orthogonal matrices, then we can get an analytical expression about $\boldsymbol{X}$ as shown in (14):

$$X=(z_1^T\cdot I\cdot z_2)\otimes W \quad (14)$$

In equation (14), operator $\otimes$ means that the result matrix in brackets performs Hadamard matrix multiplication [14] with matrix $\boldsymbol{W}$, where $W\in R^{m\times n}$ is a dense matrix denoted by the following expression:

$$W(i,j)=\frac{1}{\frac{\Delta_1(i,i)}{r_x}+\frac{\Delta_2(j,j)}{r_y}}$$

Finally, from (14) we can get the analytical solution of equation (10) as shown in (15).

$$V=z_1\cdot X\cdot z_2^T=z_1\cdot((z_1^T\cdot I\cdot z_2)\otimes W)\cdot z_2^T \quad (15)$$

Further, due to the symmetry property of matrix z_1 and z_2, (15) can be expressed as (16).

$$V = z_1 \cdot ((z_1 \cdot I \cdot z_2) \otimes W) \cdot z_2 \quad (16)$$

So far, once we know the voltage drop distribution, by using (10), we can get the current distribution. On the other hand, if we know the current distribution, using (16) we also can get the voltage drop distribution. In any direction, the solution is absolutely unique and in an analytical form.

5. Boundary Iteration Process

For wire-bond packaging type, all the pad nodes are located at block boundary, thus all boundary elements of matrix I in equation (16) are unknown because they stand for the current drawn by pads. For flip-chip packaging type, the unknown elements of matrix I may be appeared at the internal position of matrix I because there are pads inside each Poisson Block. Because we cannot know the exactly current drained by pads in advance, thus we cannot directly use (16) to get the voltage drop distribution. This problem is named boundary problem. In order to handle this problem, we develop an analytical iterative method. The basic idea here is to use an iteration process to improve the accuracy of initial guess of the current drawn by pad nodes step by step. Once we set up a boundary current distribution drawn by internal pads, then we can merge them with other nodes' current distribution to complete the matrix I. Then by using the equation (16), we can get the overall voltage drop distribution including the boundary voltage drop. The key observation here is that the voltage drop on pads should be zero. If a guessed boundary current distribution causes the calculated result to be nonzero, we know that the boundary current distribution is not accurate. Further, because the solution of equation (10) is unique, if an iterative process can make the boundary voltage drop to converge to zero, then the boundary current distribution will also converge to the accurate value, which will cause the overall voltage drop distribution to converge to the unique solution of equation (10). Thus, we can use a minimization algorithm to compute the accurate solution.

Specifically, by defining the current vector drawn by pad nodes as i_b, other nodes' current distribution vector as i_n, the boundary voltage drop vector as v_b, then the complete simulation process can be formulated as a minimization process as following:

$$\min_{i_b \in R^{np}} \|v_b\| \quad s.t. \begin{cases} v_b = bond(V) \\ V = z_1 \cdot ((z_1 \cdot I \cdot z_2) \otimes W) \cdot z_2 \\ I = i_b \,\|\, i_n \end{cases} \quad (17)$$

Here we use a function named "*bond*" to stand for the process that extracts the voltage drop at pad nodes from overall drop matrix V to get vector v_b. Also, the construction process of matrix I using vector i_b and i_n is expressed as $I = i_b \,\|\, i_n$. Minimization process (17) tries to find an optimal vector i_b in the geometry space R^{np} (*np* is the number of pads, *m* and *n* are related to the grid size) to minimize the voltage drop on pads nodes. As we known, a Newton gradient method together with a line search strategy can be used to solve the minimization problem given by (17) [15]. Usually, the number of pad nodes is few thousand, so if we can get the gradient direction analytically during the iteration, boundary problem caused by pad nodes can be solved very efficiently.

Suppose v_{ij} to be an element at the i^{th} row and j^{th} column of matrix V, standing for the voltage drop of a pad node. Then the gradient of v_{ij} to any variable i_{st} (at s^{th} row and t^{th} column) in matrix I can be denoted by $\frac{\partial v_{ij}}{\partial i_{st}}$. Expand equation (16) and perform derivative operation on both sides, we can get the analytical expression of the gradient in (18), where $z_1(i,:)$, $z_1(j,:)$, $z_2(s,:)$, $z_2(t,:)$ represent the i^{th} row, the j^{th} row of matrix z_1 and the s^{th} row, the t^{th} row of matrix z_2 respectively.

$$\frac{\partial v_{ij}}{\partial i_{st}} = \sum_{a=1}^{m}\sum_{b=1}^{n} Y_{ab} \quad (18)$$
$$Y = [z_1(i,:)^T \cdot z_1(j,:)] \otimes [z_2(s,:)^T \cdot z_2(t,:)] \otimes W$$

From (18) we know that $\frac{\partial v_{ij}}{\partial i_{st}}$ is the sum of all elements of matrix Y which can be computed by performing Hadamard matrix multiplication on matrix $[z_1(i,:)^T \cdot z_1(j,:)]$, $[z_2(s,:)^T \cdot z_2(t,:)]$ and W successively. As (18) gives a universal form to calculate the gradient, we can use it to calculate the Jacobian matrix J of vector v_b over vector i_b. After the J matrix is known, we can give out the complete flow of our Fast Poisson Solver algorithm as shown in Figure 3. Due to the known of analytical gradient and the small size of the Jacobian matrix, Fast Poisson Solver converges very fast.

Table 1 shows a convergence ratio comparison between the PCG method and our Fast Poisson Solver. For this test, both the PCG solver and the Fast Poisson Solver's convergence threshold are set to 1e-6. Zero fill-in incomplete LU decomposition (ILU(0)) is used as the precondition matrix for the PCG method. From Table 1, it is clear that as the size of test cases increase, the iteration number of the PCG method increases because the performance degradation of the precondition matrix. However, the iteration number of Fast Poisson Solver remains similar as the size increases. There are two reasons for the fast and stable convergence rate of Fast Poisson Solver: 1) the total pad number does not change obviously from case to case 2) Fast Poisson Solver just searches the result in a low dimension space containing all pad nodes while the PCG method searches for a huge space which contains all nodes of the power grid.

Table 1. Convergence Ratio Comparison between PCG Solver and Fast Poisson Solver

Node Num	PCG Iteration	Max PCG Error	Poisson Solver Iteration	Max Poisson Solver Error
1M	709	9.851000E-07	23	3.356580E-07
1.2M	779	9.661470E-07	34	3.185260E-07
1.4M	851	9.805820E-07	32	4.445130E-07
1.6M	922	9.677930E-07	21	4.100990E-07
1.9M	993	9.915360E-07	29	3.679020E-07
2.3M	1063	9.647550E-07	17	7.098380E-07
2.5M	1142	9.809760E-07	19	1.023180E-06
2.9M	1199	9.981070E-07	18	7.964340E-07
3.2M	1266	9.905290E-07	21	1.065640E-06
3.6M	1347	9.739280E-07	25	1.091320E-06
4M	1421	9.951570E-07	25	1.200610E-06

Also, from Figure 3, we can see that in step 2 of the Fast Poisson Solver, the computation time is mainly spent in sub-step 1, sub-step 4 and sub-step 11 during each iteration. This is because when the grid size is increasing all these sub-steps require large size

dense matrix multiplication operations to calculate equation (16) and (18), which are very time consuming. In the following sections, we will introduce a technique to speedup these computation under a GPU environment.

Algorithm Name: Poisson Solver

Input: A P/G grid with current distribution

Output: Voltage drop distribution of the grid

Process:

Step1: Partition the P/G grid as a few Poisson Block $\mathbf{P}_i$ together with sub current distribution $\mathbf{I}_i$

Step2: For each Poisson Block pair ($\mathbf{P}_i$, $\mathbf{I}_i$)

1. Calculate the Jacobian matrix $\boldsymbol{J}$ using equation (18)
2. Set initial guess of boundary current distribution $\boldsymbol{i_b}$ to zero
3. Pack $\mathbf{I}_i$ and $\boldsymbol{i_b}$ to get $\boldsymbol{I}$ matrix in equation (16)
4. Calculate the boundary element of matrix $\boldsymbol{V}$ using equation (16)
5. Pack the boundary element of matrix $\boldsymbol{V}$ into vector $\boldsymbol{v_b}$
6. Soving equation $\boldsymbol{Jf} = -\boldsymbol{v_b}$ to get a decent direction vector $\boldsymbol{f}$
7. Find a scalar α to update $\boldsymbol{i_b}$ in order to get a maximal decrease within an iteration.
8. Update $\boldsymbol{i_b}$ to $\boldsymbol{i_b} = \boldsymbol{i_b} + \alpha \boldsymbol{f}$
9. If not converge, goto 3 else goto 10
10. Pack $\mathbf{I}_i$ and $\boldsymbol{i_b}$ to get $\boldsymbol{I}$ matrix in equation (16)
11. Calculate matrix $\boldsymbol{V}$ using equation (16) to get overall voltage drop distribution

End for

Figure 3. Fast Poisson solver algorithm

6. Speedup on GPU

Recently, GUP has been proved to be very efficient for computing intensive tasks due to its inherent parallel computing powers [8-10]. However, the traditional solvers (SuperLU based and PCG based solvers) for P/G grid are difficult to be optimized to leverage the parallelism in GPU. No more than 10X speed up are reported by using GPU architecture in recent research [8-10]. Luckily, in Fast Poisson Solver algorithm, matrix multiplication is the main time consuming operation to calculate equation (16) and (18). Because the matrixes $\boldsymbol{z_1}$, $\boldsymbol{z_2}$ and $\boldsymbol{W}$ are only related to the grid size, once the grid size is determined, they keep unchanged during the iterative solving process. Thus, in Fast Poisson Solver, we can load them once and reuse them during iteration process. This property is very friendly for GPU computation because it reduces the communication between CPU and GPU dramatically, which is known as the main bottleneck for performance improvement on GPU architecture.

For equation (16), it can be reduced to 5 steps on GPU: (1) to get $t = z_1 \cdot I$ (2) to get $t = t \cdot z_2$ (3) to get $t = t \otimes W$ (4) to get $t = z_1 \cdot t$ (5) to get $t = t \cdot z_2$. Here all steps are standard matrix multiplication operation, which can be assigned to different core of GPU to achieve extremely high computation performance. Also, for step (3), the Hadamard matrix multiplication can be implemented as scalar vector multiplication which can also be calculated parallel on different GPU cores. For equation (18), it can be implemented on GPU in 4 steps: (1) to get $t_1 = z_1(i,:)^T \cdot z_1(j,:)$ (2) to get $t_2 = z_2(s,:)^T \cdot z_2(t,:)$ (3) to get $t = t_1 \otimes t_2$ (4) to get $t = t \otimes W$. Here steps (3) and (4) can also be implemented using vector scalar multiplication while step (1) and step (2) can be implemented using rank one vector multiplication on GPU. Vector scalar multiplication, vector rank one multiplication and matrix multiplication are all defined in BLAS (basic linear algebra subroutine) library as level one and level two functions [16]. In this paper we choose NVIDIA CUDA framework [17] to deploy our GPU application. CUDA supports BLAS library and provides a very friendly C interface for programmer. Thus, all sub calculation steps mentioned above can be mapped to a CUDA function directly which are already implemented efficiently on GPU.

CUDA also supports for parallel 2D FFT library which can accelerate FFT computation speed by 15X in recent report [18]. Here due to the special structure of matrix $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$, we also can benefit by using CUDA 2D FFT library. The improvement for efficiency by using FFT is caused by the matrix multiplication operation. As denoted by (12), all elements in matrix $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ are results of different sine functions. So $f = z_1 \cdot p$ can be calculated using Discrete Sine Transform method (DFT), which results $f = DFT(p)$. Further, according to basic property of DFT, it can be calculated using FFT on an extended vector with double length [19]. Thus, we can get the following formulation:

$$f = DFT(p) = FFT(q), \quad q = [0, p, 0, -p]$$

Ranging vector $\boldsymbol{p}$ column by column in an object matrix, matrix multiplication with $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ can be calculated using 2D FFT algorithm, i.e, $t = z_1 \cdot I = FFT(I')$, $I' = [0, I, 0, -I]^T$. Because the computation complexity of FFT is just around *O(NlgN)*, and matrix multiplication operation has the highest computation complexity in Fast Poisson Solver algorithm, thus the computation complexity of Fast Poisson Solver is bounded by *O(NlgN)*. Compared with the traditional solver's computation complexity $O(N^{1.5})$, such as SuperLU and PCG, even this algorithm is implemented without any parallel computing resources, the run time efficiency will be increased significantly.

For CUDA implementation, another issue is about its accuracy. All data can only be expressed as float numbers in GPU, which will affect lots of applications. However, due to the iterative nature of Fast Poisson Solver，the problem is partially overcome. One can easily set a convergence threshold to control and minimize the final calculation errors, such that the accuracy is enough for voltage drop analysis.

7. Experimental Results

We implement our Fast Poisson Solver using C++ language and compare the performance of different solving techniques under the same environment. The tested 5 different solvers include: 1) direct sparse solver using SupperLU newest version 3.1 [11]. 2) iterative PCG solver using standard zero fill-in incomplete LU decomposition based preconditioner [12]. 3) Fast Poisson Solver implemented only using CPU 4) Fast Poisson Solver implemented using GPU without FFT acceleration. 5) Fast Poisson Solver implemented using GPU with FFT acceleration. We test these solvers in a 64 bits Linux Server with Intel Xeon CPU running at 3GHz with 8GB memory. For GPU environment, we use a NVIDIA Tesla C870 GPU with 128 cores and 1.5GB display memory. The used CUDA library is in version 2.0. We test 10 P/G grid cases. The smallest one contains one million nodes while the largest one contains four million nodes. Table 2 gives out the run time comparison for the five different solvers.

Table 2: Run Time Performance of different Solvers (unit: sec)

Node Num	SuperLU Solver CPU	ILU(0) PCG Solver CPU	Poisson Solver CPU	Poisson Solver GPU	Poisson Solver GPU+FFT
1M	55.7	138.77	15.64	4.94	3.96
1.2M	72.29	185.98	20.93	4.96	4.33
1.4M	93.12	240.28	27.16	5.3	4.28
1.6M	122.62	307.83	36.33	5.37	3.96
1.9M	142.48	382.09	44.87	6.02	3.96
2.3M	178.92	473.07	54.9	6.75	4.31
2.5M	215.78	572.83	66.27	7.69	4.33
2.9M	307.55	728.83	79.16	9.35	4.45
3.2M	365.06	848.4	94.3	10.9	4.54
3.6M	407.2	960.44	110.51	12.6	4.72
4M	521.43	1159.94	125.63	13	4.94

From Table 2, it is clear that as the grid size is increasing, the run time of the SuperLU method and the PCG method increases super-linearly. The CPU version of the new algorithm is about 5X faster than SuperLU and about 10X faster than the PCG solver. For the new GPU-base Fast Poisson Solver, the run time increases much slowly as the case size increases. When the FFT acceleration strategy is used, the run time almost remains constant for all test cases. For the largest case, the GPU version of the Fast Poisson Solver without FFT acceleration gains about 40X speed up over the SuperLU solver and about 90X speed up than the PCG solver. By using FFT acceleration technique, for the largest test case, the GPU version of Fast Poisson Solver gains 100X speed up over the SuperLU solver and 230X speed up over the PCG solver. Analysis results for the GPU active time show that exchanging data between CPU and GPU costs about 4 seconds for each case. So when matrix multiplication and 2D FFT operations can be calculated very fast on GPU, the run-time is mainly used to exchange data between CPU and GPU memories.

It is well known that the traditional solver is hard to be accelerated by using GPU. Reported speed up of the dense direct solver (LU decomposition based algorithm) under GPU in [8] is limited to about 5X while the reported speed up for PCG based solver in [9] is limited to 4X. Thus, compared with these results, the 100X speed up in our algorithm is a substantial performance improvement.

Further, we compare the memory usage of these different solvers. The results are shown in Table 3. From the table, we can see that the memory usage is the same for both the GPU version and the CPU version of Poisson Solver which is about 1.5X larger than the memory usage of the PCG method. For the FFT version, due to the expanding of object matrices, the memory usage is as 3X as that of the PCG solver. Further, the memory usage of SuperLU increases dramatically due to the fill in elements, when the grid size increases. The GPU FFT version of Fast Poisson Solver only uses as about 1/10 of that of SuperLU. For the largest case, SuperLU solver uses 9GB memory while the Poisson Solver just uses 1.1GB memory. These data proves that the memory performance of Poisson Solver is also excellent.

Table 3. Memory usage comparison among 5 solvers (unit: MB)

Node Num	SuperLU Solver CPU	ILU(0) PCG Solver CPU	Poisson Solver CPU	Poisson Solver GPU	Poisson Solver GPU+FFT
1M	1646.30	80	120.49	120.49	277.12
1.2M	2032.81	96.8	145.74	145.74	335.20
1.4M	2445.93	115.2	173.39	173.39	398.79
1.6M	2941.05	135.2	203.43	203.43	467.90
1.9M	3401.31	156.8	235.88	235.88	542.53
2.3M	3987.59	180	270.73	270.73	622.68
2.5M	4547.00	204.8	307.98	307.98	708.36
2.9M	5729	231.2	347.63	347.63	799.55
3.2M	6829	259.2	389.68	389.68	896.26
3.6M	7486	288.8	434.13	434.13	998.49
4M	9158	320	480.98	480.98	1106.25

8. Conclusion

In this paper, we have proposed a new algorithm to solve large-scale structured P/G grid networks. The new method explores the regular structures of P/G networks by a new formulation, which is very amenable for GPU computing. The CPU version of the new algorithm is about 5X faster than widely used direct solver SuperLU and about 10X faster than iterative PCG solver. The GPU-enabled version with the FFT acceleration technique can achieve 100X speed up over the SuperLU Solver and 200X speed up over the PCG Solver. This speed up ratio is superior to the reported performance of existing GPU based solutions. The memory usage of this new algorithm is only as 3X as that of the PCG Solver and is as about 1/10 of the memory usage of SuperLU. Further, the iteration process of the algorithm does not suffer the slower convergence ratio when matrix size increases.

References and Citations

[1]. T. Chen and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods", DAC2001 Proceedings, pp. 559-562

[2]. J. N. Kozhaya, S. R. Nassif and F.N. Najm, "A multigrid-like technique for power grid analysis", IEEE Trans. On Computer Aided Design, vol.21, no.10, Oct. 2002, pp. 1148-1160

[3]. H. Qian, S. R. Nassif, S. S. Sapatnekar, "Random walks in a supply network", DAC 2003 Proceedings, pp. 93-98

[4]. M. Zhao, R. V. Panda, S. S. Sapatnekar and D. Blaauw, "Hierarchical analysis of power distribution networks", IEEE Trans. On Computer Aided Design, vol. 21, no. 2, Feb. 2002, pp. 159–168

[5]. E. Chiprout, "Fast flip-chip power grid analysis via locality and grid shells", ICCAD 2004 Proceeding, pp. 485-488

[6]. J. Singh, S. S. Sapatnekar, "Topology optimization of structured power/ground networks", ISPD 2004 Proceedings, pp. 16–123.

[7]. J. Shi, Y. Cai, S. X.-D. Tan, J. Fan, X. Hong, "Pattern-Based Iterative Method for Extreme Large Power/Ground Analysis", IEEE Trans. on CAD of Integrated Circuits and Systems vol.26, no.4, April 2007, pp. 680-692

[8]. Galoppo N, Govindaraju N K, Henson M, et al. "LU-GPU: Efficient Algorithms for Solving Dense Linear System on Graphics Hardware", ACM/IEEE SC 2005 Proceedings, pp.3-8.

[9]. Buatois L, Caumon2 G, Lévy B., "Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU", Proceedings of High Performance Computation Conference, 2007, pp. 358-371.

[10]. Kanupriya Gulati, John F. Croix, Sunil P. Khatr, Rahm Shastry, "Fast Circuit Simulation on Graphics Processing Units", ASP-DAC 2009 Proceedings, pp. 403-408

[11]. http://crd.lbl.gov/ xiaoye/SuperLU

[12]. Saad. Yousef, "Iterative Methods for Sparse Linear Systems", PWS Publishing Company, 1996.

[13]. Strang G. "Introduction to Applied Mathematics", Wellesley-Cambridge Press, 1986.

[14]. http://en.wikipedia.org/wiki/Matrix_multiplication

[15]. Broyden, C.G., "The Convergence of a Class of Double-Rank Minimization Algorithms", Journal Inst. Math. Applic., Vol. 6, pp. 76-90, 1970.

[16]. http://www.netlib.org/blas/

[17]. http://www.nvidia.com/object/cuda_home.html

[18]. http://developer.nvidia.com/object/matlab_cuda.html

[19]. http://heim.ifi.uio.no/~tom/fastpoissonslides.pdf