

A Hybrid Linear Equation Solver and its Application in Quadratic Placement

Haifeng Qian and Sachin S. Sapatnekar
 University of Minnesota, Minneapolis, MN
 {qianhf,sachin}@ece.umn.edu

Abstract—This paper presents a new hybrid linear equation solver for quadratic placement. The new solver is a combination of stochastic solver and iterative solver: it is proven in this paper that an approximate LDL factorization can be obtained from random walks, and used as a preconditioner for conjugate gradient solver. Testing on real-life placement benchmarks shows a speedup of up to 7.1 times over traditional Incomplete Cholesky preconditioned Conjugate Gradient (ICCG).

I. INTRODUCTION

Placement is a critical and computationally intensive step during the VLSI design cycle that must handle instances of large size. The most widely used approaches fall into several different paradigms: quadratic placement [6][10][18][19], simulated annealing [16], and partitioning-based placement [4]. Of these, quadratic placement, also referred to as analytical or force-directed placement, has emerged as a very popular method, and is the focus of this paper. The essential idea is to define a set of attractive and repulsive forces between the modules, and to iteratively find an equilibrium point that corresponds to the optimal placement. Each iterative step involves the minimization of a cost function, whose components typically include an indirect measure of wire length, plus factors such as congestion, overlap, or timing, and requires the solution of a large set of linear equations to compute new locations for modules/cells. The set of linear equations can be written as $A\mathbf{x} = \mathbf{b}$, where A is a square matrix that is typically symmetric and positive definite, \mathbf{b} is a given vector based on the cost function, and \mathbf{x} is the vector of new coordinates of modules/cells to be computed. The most widely used solver in placers is the Incomplete Cholesky Factorization preconditioned Conjugate Gradient (ICCG) [3][11][15] method: examples of state-of-the-art quadratic placers that use ICCG or its variants include [18] and [19].

In this paper, we propose a hybrid linear equation solver, which combines stochastic solvers [7][14] and iterative solvers, and performs favorably when compared with existing approaches. We prove that an incomplete LDL factorization of A can be obtained from the stochastic solution process of [14], and that it can be used as the preconditioner for preconditioned conjugate gradient (PCG). We argue that the obtained incomplete LDL factors have better quality, i.e., better accuracy-size tradeoffs, than the incomplete Cholesky factor obtained by a traditional method, and we test the proposed solver on real-life placement matrices to support this claim.

II. FRAMEWORK OF THE HYBRID SOLVER

A. The Random Walk Game

This section briefly describes the stochastic portion of the solver, and more details can be found in [14]. For clarity of presentation, the discussion in this paper is limited to R-matrices defined as follows.

Definition 1: Matrix A is said to be an R-matrix if it satisfies these four properties: 1) $A_{i,i} > 0, \forall i$. 2) $A_{i,j} \leq 0, \forall i \neq j$. 3) $A_{i,j} = A_{j,i}, \forall i \neq j$. 4) Matrix A is irreducibly diagonally dominant [17].

[14] defines a random walk “game” as follows. Given a finite undirected connected graph representing a street map, a walker starts

from one of the nodes, and goes to one of the adjacent nodes every day with a certain probability. The walker pays an amount of money, m_i at node i , to a motel for lodging everyday, until he/she reaches one of the homes, which are a subset of the nodes. Then the journey is complete and he/she will be rewarded a certain amount of money, m_0 . The problem is to determine the gain function:

$$f(i) = E[\text{money earned} \mid \text{walk starts at node } i] \quad (1)$$

These gain values satisfy the following linear equations [14]:

$$\begin{aligned} f(i) &= \sum_{j \in \text{neighbors of } i} p_{i,j} f(j) - m_i, \quad \forall i \\ f(\text{a home node}) &= m_0 \end{aligned} \quad (2)$$

where $p_{i,j}$ is the transition probability of going from node i to node j , and note that j can be a home node. Thus a random walk game is mapped onto a system of linear equations. Conversely, it can be verified that given $A\mathbf{x} = \mathbf{b}$, where A is an R-matrix, we can always construct a random walk game that is mathematically equivalent, in which the set of f values is equal to the solution vector \mathbf{x} .

To find the i^{th} entry of \mathbf{x} , one may run a number of walks from node i and take the average of the results. To get the complete solution, one may repeat the process for every entry of \mathbf{x} . This, however, is not the most efficient way, and [14] proposed two speedup techniques, which will play an important role in our theory:

- 1) Every calculated node becomes a new home in the game with an award amount equal to its calculated f value.
- 2) To facilitate re-solves with different right-hand-side vectors, a record is set up for each node in the game. The record for node i keeps a count of the locations where awards are received at the end of each walk started from node i (they possibly include the original home nodes and the new home nodes created by the first speedup technique), and the number of stays at a particular motel during all the walks from node i . If the right-hand-side vector \mathbf{b} changes while matrix A remains the same, only the award amounts and motel prices have changed, and we can assume that the walker receives awards at the very same locations and pays for the very same motels. Using the journey information from the record, new solutions can be computed efficiently without running any extra walks.

B. The Sequential Monte Carlo Method

A second basis for our approach is the sequential Monte Carlo method, which was initiated in [13] and was developed in [7][8].

Let \mathbf{x}' be an approximate solution to $A\mathbf{x} = \mathbf{b}$ found by a stochastic solver, such as that in the previous subsection. Define:

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}' \quad \mathbf{y} = \mathbf{x} - \mathbf{x}' \quad (3)$$

It is easy to verify the following relation:

$$A\mathbf{y} = \mathbf{r} \quad (4)$$

The idea is to iteratively solve (4) using the stochastic solver. In each iteration, an approximate \mathbf{y} is computed and then used to correct the current solution \mathbf{x}' . The algorithm can be written as follows.

Algorithm 1: Sequential Monte Carlo algorithm:

- 1) Stochastic solve $A\mathbf{x} = \mathbf{b}$, find \mathbf{x}_0
- 2) For $j = 0, 1, 2, \dots$, until convergence
- 3) $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j$
- 4) Stochastic solve $A\mathbf{y} = \mathbf{r}_j$, find \mathbf{y}_j
- 5) $\mathbf{x}_{j+1} = \mathbf{x}_j + \mathbf{y}_j$

C. An Initial Hybrid Solver

We now combine the techniques from the previous two subsections and realize their full potential. With the bookkeeping technique from the end of Section II-A, we only need random walks in the initial solve of Algorithm 1, and we can use the record to solve $A\mathbf{y} = \mathbf{r}_j$ without a single extra walk. This results in Algorithm 2. Note that this is only an initial prototype algorithm.

Algorithm 2: An initial hybrid algorithm:

- 1) Solve $A\mathbf{x} = \mathbf{b}$, find \mathbf{x}_0 , create record
- 2) For $j = 0, 1, 2, \dots$, until convergence
- 3) $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j$
- 4) Apply record on $A\mathbf{y} = \mathbf{r}_j$, find \mathbf{y}_j
- 5) $\mathbf{x}_{j+1} = \mathbf{x}_j + \mathbf{y}_j$

The calculations that are used to apply the record are purely linear operations. Therefore, for the approximate solution of $A\mathbf{y} = \mathbf{r}_j$, the overall effect can be written as a matrix-vector multiplication:

$$\mathbf{y}_j = T\mathbf{r}_j \quad (5)$$

where T is a square matrix that represents the process of applying the record of random walks. The matrix T has a special structure that we will discuss in the next section. Thus the computation during one iteration of Algorithm 2 can be represented as follows:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + T\mathbf{r}_j = \mathbf{x}_j + T(\mathbf{b} - A\mathbf{x}_j) = (I - TA)\mathbf{x}_j + T\mathbf{b} \quad (6)$$

Equation (6) is in exactly the same form as a preconditioned Gauss-Jacobi iterative solver, where the preconditioner is T . In other words, Algorithm 2 is an iterative solver with a preconditioner built by an stochastic solver, and thus we call it a *hybrid solver*. Again, this is only a first cut at the hybrid approach. To generalize the approach, the iterative part does not have to be Gauss-Jacobi and potentially can be *any* iterative solver. We will prove in the next section that an incomplete LDL factorization (definition to follow) can be obtained from random walks, and can be used to precondition CG.

III. PROOF OF INCOMPLETE FACTORIZATION

The *complete* LDL factorization of a symmetric and positive definite matrix A is a slight variation of the Cholesky factorization, and is defined as $A = LDL^T$ where L is a lower triangular matrix with all diagonal entries being 1, and D is a diagonal matrix with all positive diagonal values. The *incomplete* LDL factorization is defined as $L'D'L'^T \approx A$ where $D' \approx D$, $L' \approx L$, and the non-zero pattern of L' is a subset of that of L . We now examine the details of the record created by the bookkeeping technique in Section II-A, and prove that it provides an incomplete LDL factorization of matrix A .

A. The Approximate Factorization

Suppose the dimension of matrix A is N , and its i^{th} row corresponds to node i in the game, $i = 1, 2, \dots, N$. Without loss of generality, let us assume that in the stochastic solve, the order in which we solve the nodes is the same as the index order. Therefore, when we are solving node k , the nodes $\{1, 2, \dots, k-1\}$

are already solved and they now serve as home nodes where a random walk ends. The awards for reaching nodes $\{1, 2, \dots, k-1\}$ are $\{x_1, x_2, \dots, x_{k-1}\}$ respectively. Suppose we choose $m_0 = 0$, then $m_i = -\frac{b_i}{A_{i,i}}$, for $i = k, k+1, \dots, N$ [14].

Now, if W_k walks are carried out from node k , we get:

$$x_k = \frac{\sum_{i=1}^{k-1} M_{k,i} x_i + \sum_{i=k}^N J_{k,i} \frac{b_i}{A_{i,i}}}{W_k} \quad (7)$$

where $M_{k,i}$ is the number of walks from node k that end at node $i \in \{1, 2, \dots, k-1\}$, and $J_{k,i}$ is the number of times that walks commencing at node k pass the motel at node $i \in \{k, k+1, \dots, N\}$. Note that the awards received at the original home nodes are ignored since $m_0 = 0$. Moving the $M_{k,i}$ terms to the left side, we obtain:

$$-\sum_{i=1}^{k-1} \frac{M_{k,i}}{W_k} x_i + x_k = \sum_{i=k}^N \frac{J_{k,i}}{W_k A_{i,i}} b_i \quad (8)$$

The above equation can be written in matrix form:

$$Y\mathbf{x} = Z\mathbf{b} \quad (9)$$

where Y and Z are two square matrices of dimension N :

$$\begin{aligned} Y_{k,k} &= 1, \quad \forall k \\ Y_{k,i} &= -\frac{M_{k,i}}{W_k}, \quad \forall k > i \\ Y_{k,i} &= 0, \quad \forall k < i \\ Z_{k,i} &= \frac{J_{k,i}}{W_k A_{i,i}}, \quad \forall k \leq i \\ Z_{k,i} &= 0, \quad \forall k > i \end{aligned} \quad (10)$$

Obviously Y is a lower triangular matrix with unit diagonal entries, Z is an upper triangular matrix, and their entries are independent of \mathbf{b} . Once Y and Z are built, given any \mathbf{b} , one can apply equation (9) and find \mathbf{x} efficiently. The matrix T defined in (5) is simply $Y^{-1}Z$.

From equation (9), we have:

$$Z^{-1}Y\mathbf{x} = \mathbf{b} \quad (11)$$

Since the vector \mathbf{x} in the above equation is an approximate solution to the original equation set $A\mathbf{x} = \mathbf{b}$, it follows that:

$$Z^{-1}Y \approx A \quad (12)$$

Because the inverse of an upper triangular matrix, Z^{-1} , is also upper triangular, equation (12) is in the form of a ‘‘UL factorization’’ of A . The following definition and lemma present a simple relation between UL factorization and the common LU factorization.

Definition 2: The operator $\text{rev}(\cdot)$ is defined on square matrices as follows: given matrix A of dimension N , $B = \text{rev}(A)$ is also a square matrix of dimension N , and satisfies:

$$B_{i,j} = A_{N+1-i, N+1-j}, \quad \forall i, j \in \{1, 2, \dots, N\}$$

Lemma 1: Let $A = LU$ be the LU factorization of a square matrix A , then $\text{rev}(A) = \text{rev}(L)\text{rev}(U)$ is true and is the UL factorization of $\text{rev}(A)$.

The proof is omitted. Applying Lemma 1 on (12), we obtain:

$$\text{rev}(Z^{-1})\text{rev}(Y) \approx \text{rev}(A) \quad (13)$$

Since A is an R-matrix and is symmetric, $\text{rev}(A)$ must be also symmetric, and we can take the transpose of both sides, and have:

$$(\text{rev}(Y))^T (\text{rev}(Z^{-1}))^T \approx \text{rev}(A) \quad (14)$$

The above equation has the form of a Doolittle LU factorization: matrix $(\text{rev}(Y))^T$ is lower triangular and have unit diagonal values; matrix $(\text{rev}(Z^{-1}))^T$ is upper triangular.

Lemma 2: The Doolittle LU factorization of a matrix is unique.

The proof is omitted. Let the Doolittle LU factorization of $\text{rev}(A)$ be $\text{rev}(A) = L_{\text{rev}(A)}U_{\text{rev}(A)}$, and its LDL factorization be $\text{rev}(A) = L_{\text{rev}(A)}D_{\text{rev}(A)}(L_{\text{rev}(A)})^T$. Since equation (14) is an approximate LU factorization of $\text{rev}(A)$, while the exact LU factorization is unique, it must be true that:

$$(\text{rev}(Y))^T \approx L_{\text{rev}(A)} \quad (15)$$

$$(\text{rev}(Z^{-1}))^T \approx U_{\text{rev}(A)} = D_{\text{rev}(A)}(L_{\text{rev}(A)})^T \quad (16)$$

Note that, the above two equations only tell us that from the matrix Y built by random walks, we can obtain an approximate factor $L_{\text{rev}(A)}$. We have yet to prove that the non-zero pattern of $(\text{rev}(Y))^T$ is a subset of that of $L_{\text{rev}(A)}$. The proof is omitted here, and is based on the combination of established theory on the structure of LU factors [2][5][9], equation (10), and the fact that intermediate nodes of a walk from node k must be in the set $\{k, k+1, \dots, N\}$. This and equation (15) give rise to the following lemma.

Lemma 3: $(\text{rev}(Y))^T$ is the L factor of an incomplete LDL factorization of matrix $\text{rev}(A)$.

B. The Diagonal Component

To evaluate the approximate D matrix, we take the transpose of both sides of equation (16), and get:

$$\text{rev}(Z^{-1}) \approx L_{\text{rev}(A)}D_{\text{rev}(A)} \quad (17)$$

Lemma 4: For a non-singular square matrix A , $\text{rev}(A^{-1}) = (\text{rev}(A))^{-1}$.

The proof is omitted. Applying the lemma on (17), we have:

$$\begin{aligned} (\text{rev}(Z))^{-1} &\approx L_{\text{rev}(A)}D_{\text{rev}(A)} \\ I &\approx \text{rev}(Z)L_{\text{rev}(A)}D_{\text{rev}(A)} \end{aligned} \quad (18)$$

Recall that $\text{rev}(Z)$ and $L_{\text{rev}(A)}$ are both lower triangular, that $L_{\text{rev}(A)}$ has unit diagonal entries, and that $D_{\text{rev}(A)}$ is a diagonal matrix, the $\{i, i\}$ diagonal entry in the above equation is simply:

$$\begin{aligned} (\text{rev}(Z))_{i,i} (L_{\text{rev}(A)})_{i,i} (D_{\text{rev}(A)})_{i,i} &\approx 1 \\ (\text{rev}(Z))_{i,i} \cdot 1 \cdot (D_{\text{rev}(A)})_{i,i} &\approx 1 \\ (D_{\text{rev}(A)})_{i,i} &\approx \frac{1}{(\text{rev}(Z))_{i,i}} \end{aligned} \quad (19)$$

Applying Definition 2 and equation (10), we finally have the equation for computing the approximate D factor, given as follows:

$$\begin{aligned} (D_{\text{rev}(A)})_{i,i} &\approx \frac{1}{Z_{N+1-i, N+1-i}} \\ &= \frac{W_{N+1-i} A_{N+1-i, N+1-i}}{J_{N+1-i, N+1-i}} \end{aligned} \quad (20)$$

Definition 3: The operator $\text{rev}(\cdot)$ is defined on vectors as follows: given vector \mathbf{x} of length N , $\mathbf{y} = \text{rev}(\mathbf{x})$ is also a vector of length N , and satisfies: $\mathbf{y}_i = \mathbf{x}_{N+1-i}, \forall i \in \{1, 2, \dots, N\}$.

It is easy to verify that the equation set $A\mathbf{x} = \mathbf{b}$ is equivalent to $\text{rev}(A)\text{rev}(\mathbf{x}) = \text{rev}(\mathbf{b})$.

By now, we have collected the necessary pieces of the proposed hybrid solver, and it is summarized as in the pseudocode below:

Algorithm 3: The final hybrid solver:

- 1) Precondition
- 2) Run random walks, build matrix Y and find diagonal

entries of Z , using equation (10).

- 3) Build $L_{\text{rev}(A)}$ using equation (15).
- 4) Build $D_{\text{rev}(A)}$ using equation (20).
- 5) Given \mathbf{b} , solve
- 6) Convert $A\mathbf{x} = \mathbf{b}$ to $\text{rev}(A)\text{rev}(\mathbf{x}) = \text{rev}(\mathbf{b})$.
- 7) Apply ICCG on $\text{rev}(A)\text{rev}(\mathbf{x}) = \text{rev}(\mathbf{b})$ with the preconditioner $(L_{\text{rev}(A)}D_{\text{rev}(A)}(L_{\text{rev}(A)})^T)^{-1}$.
- 8) Convert $\text{rev}(\mathbf{x})$ to \mathbf{x} .

C. Stopping Criterion

In this subsection, we look at the accuracy control of random walks, which must be independent of the right-hand-side vector \mathbf{b} . We define a stopping criterion on a value that is a function of only matrix A , as follows. Let $H_k = E[\text{length of a walk from node } k]$, let H'_k be the average length of the W_k walks, and the stopping criterion is:

$$P[-\Delta < \frac{H'_k - H_k}{H_k} < \Delta] > \alpha \quad (21)$$

where Δ is a relative error margin, and α is a confidence level, e.g. $\alpha = 0.99$. Practically, this criterion is checked by the inequality:

$$\frac{H'_k}{\sqrt{\text{Var}_k/W_k}} > \frac{Q^{-1}(\frac{1-\alpha}{2})}{\Delta} \quad (22)$$

where Var_k is the variance of the lengths of the W_k walks, and Q is the standard normal complementary cumulative distribution function. Thus, W_k is decided on the fly, and random walks are run from node k until condition (21) is satisfied.

D. Random walks versus ILUT

In this section, we argue that the incomplete LDL factorization produced by random walks has better quality than traditional incomplete Cholesky factor B . In other words, if matrices Y and B have the same number of non-zero entries, and given the same target accuracy requirement, we expect the hybrid solver to converge with fewer iterations than a traditional ICCG preconditioned by $(BB^T)^{-1}$.

Existing techniques perform Gaussian elimination on A , and use a specific strategy to drop insignificant entries during the process. Such a strategy can be pattern-based, such as ILU(0), or value based, such as ILUT, or a combination of multiple criteria [15]. Our argument is based on the fact that, in traditional Gaussian-elimination-based methods, the operations of eliminating different nodes are correlated and the error introduced at an earlier node gets propagated to a later node, while in random walks, the operation on a node is totally independent from other nodes.

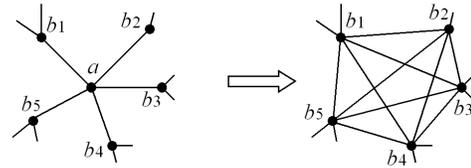


Fig. 1. One step in Gaussian elimination.

Let us use ILUT as an example. Given a symmetric matrix A of dimension N , the *matrix graph* G is defined as a undirected graph with N nodes $\{1, 2, \dots, N\}$ such that an edge exists between two nodes $i \neq j$ if and only if $A_{i,j} \neq 0$. Figure 1 illustrates one step in *complete* Gaussian elimination: removing one node from the matrix graph, and creating a clique among its neighbours. The new edges correspond to fills added to the remaining matrix; at the same time, five non-zero values are written into the L factor. Suppose in

the process of eliminating node a , ILUT decides that the new edge between nodes b_1 and b_2 corresponds to a below-threshold entry, then that entry is dropped from the remaining matrix, and that edge is removed from the remaining graph. Later when the algorithm reaches the stage of eliminating node b_1 , because of that missing edge, no edge is created from b_2 to the neighbors of b_1 , and thus more edges are missing, and this new batch of missing edges will then affect later computations accordingly. Therefore, an error introduced at an early node gets propagated throughout the ILUT process. On one hand, this leads to the sparsity of B , which is desirable; on the other hand, there is no control over error accumulation, and later columns of B can deviate from the exact Cholesky factor by an amount that is greater than the planned threshold of ILUT.

The hybrid solver does not suffer from this problem. When we run random walks from node k and collect the $M_{k,i}$ values to build the k^{th} row of matrix Y , we only know that the nodes $\{1, 2, \dots, k-1\}$ are homes, and this is the only information needed. If for some reason a mistake is made, and the k^{th} row of matrix Y is of lower quality, this error does not affect other rows at all; every row is responsible for its own accuracy, given by equation (21). In summary, because of the absence of error accumulation, we expect the hybrid solver to outperform traditional ICCG.

IV. APPLICATION IN QUADRATIC PLACEMENT

In quadratic placement, the cost of a two-pin net that connects module i and module j is typically defined as [6][10][18][19]:

$$\text{netcost} = w \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

where w is the weight of the net, (x_i, y_i) and (x_j, y_j) are the coordinates of the two modules i and j . The net weight w depends on the optimization goal, for example, it can be a function of timing criticality [10]. The overall cost function of the placement is the sum of the net costs, and is typically in the following form.

$$\text{cost} = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \frac{1}{2} \mathbf{y}^T Q \mathbf{y} + \mathbf{d}_x^T \mathbf{x} + \mathbf{d}_y^T \mathbf{y} \quad (23)$$

where \mathbf{x} and \mathbf{y} are the vectors of unknown x-coordinates and y-coordinates of the modules, Q is a square matrix, \mathbf{d}_x and \mathbf{d}_y are two given vectors. The cost function is minimized by solving the following two sets of linear equations.

$$Q \mathbf{x} + \mathbf{d}_x = 0 \quad Q \mathbf{y} + \mathbf{d}_y = 0 \quad (24)$$

By examining contribution of the cost of each net to matrix Q , it can be verified that Q satisfies the first three conditions of an R-matrix. The fourth condition translates to the requirement that in every connected component of a circuit design, at least one module must be connected to an I/O pin, which is generally true. Thus, Q is an R-matrix, and the proposed hybrid solver can be applied on (24).

A placement flow involves solving (24) repeatedly, typically with different \mathbf{d}_x and \mathbf{d}_y vectors, which contain not only contributions from net costs, but also cost of overlapping modules, congestion or timing criticality, depending on the placement algorithm. Matrix Q may also change during the placement, for example, due to adding friction [19], in which case the preconditioning part of the hybrid solver needs to run again and update the LDL factorization.

V. RESULTS

In this section, we use real-life placement matrices to evaluate the proposed hybrid solver. Computations are carried out on a Linux workstation with a 2.8GHz Intel CPU. The first set of benchmarks, Table I, are matrices generated by an industrial placement tool for real-life circuits. One actual right-hand-side vector instance is

also available for each of them. The second set are the ISPD02 benchmarks [1] and their matrices are generated by the Waterloc placer [19] before the initial placement, and hence contain only connectivity component from the original netlists. A right-hand-side vector with all entries being 1 is used with each of them in the tests shown in Table II. Due to space limitations, only half of each set (the largest matrices) is shown.

In Table I and Table II, we compare the hybrid solver against ICCG with ILU(0) and ICCG with ILUT. The complexity metric is the number of double-precision multiplications needed at the iterative solving stage for the equation set $A \mathbf{x} = \mathbf{b}$, in order to converge with an error tolerance of 10^{-6} , defined as:

$$\| \mathbf{b} - A \mathbf{x} \|_2 < 10^{-6} \cdot \| \mathbf{b} \|_2 \quad (25)$$

LASPack [12] is used for ICCG with ILU(0). MATLAB is used for ICCG with ILUT. Approximate minimum degree ordering (AMD) [2] gives the best performance among available ordering options, and is used for all MATLAB tests. The dropping threshold of ILUT in MATLAB is tuned, and the setting of the hybrid solver is adjusted, such that the sizes of the Cholesky factors produced by both methods are similar, i.e., the C values in the tables are close. For LASPack and MATLAB, the $M1$ values are computed from the following equation.

$$M1 = C \cdot 2 + E + N \cdot 4 \quad (26)$$

According to the PCG pseudo codes in [3] and [15], the above equation is the best possible implementation. The $M1$ values of the hybrid solver is obtained by an embedded detailed count, and in fact equation (26) is roughly true for the hybrid solver as well.

In Table I, the hybrid solver shows up to 8.4 times speedup over ICCG with ILU(0), and up to 7.1 times speedup over ICCG with ILUT. In Table II, the hybrid solver is also at least 2 times faster than ICCG for any matrix with over 1e5 dimension. It is worth noting that m1-m5 are larger and denser than the ISPD02 matrices, and there is a trend that the larger and denser a matrix is, the more the hybrid solver outperforms ICCG. This is consistent with Section III-D: when the matrix is larger and denser, the effect of error accumulation in traditional methods becomes stronger.

Physical runtimes $T1$ and $T2$ are also included. $T1$ is only 3 to 5 times $T2$, and is a relatively low overhead: this makes the hybrid solver preferable even if the equation set is only solved once.

In Table III, the hybrid solver is embedded into the Waterloc placement tool version 1.1 [19], and the new runtimes on the ISPD02 benchmarks are compared with the original runtimes. The replaced original solver is BICGSTAB preconditioned by ILU(0) with reverse Cuthill-McKee ordering (RCM) [5]. The results suggest 5%-10% runtime reduction. The speedup is not as dramatic as in Table I and Table II, because solving linear equations is not the dominant portion of the runtime of this placer. Circuit ibm18 is not included because the placer has certain stability issue. Another purpose of Table III is to show that the hybrid solver is fully capable of supporting a placement tool. For example, for ibm17, the hybrid solver performs preconditioning on 94 different left-hand-side matrices, and solves each matrix with on average 14 different right-hand-side vectors. Note that most of these matrices takes less runtime than the original ibm17 matrix used in Table II, due to internal processing of the placer.

ACKNOWLEDGMENT

The authors would like to thank Gi-Joon Nam for the benchmarks, thank Kristofer Vorwerk and Andrew Kennings for their open-source placer and their help with its usage.

TABLE I

COMPUTATIONAL COMPLEXITY COMPARISON OF THE HYBRID SOLVER, ICCG WITH ILU(0) (LASPACK), AND ICCG WITH ILUT (MATLAB), TO SOLVE FOR ONE RIGHT-HAND-SIDE VECTOR, FOR THE FIRST SET OF MATRICES, WITH 10^{-6} ERROR TOLERANCE. N IS THE DIMENSION OF A MATRIX; E IS THE NUMBER OF NON-ZERO ENTRIES OF A MATRIX; C IS THE NUMBER OF NON-ZERO ENTRIES OF THE CHOLESKY FACTOR; $M1$ IS THE NUMBER OF MULTIPLICATIONS PER ITERATION; I IS THE NUMBER OF ITERATIONS TO REACH 10^{-6} ERROR TOLERANCE; $M2$ IS THE TOTAL NUMBER OF MULTIPLICATIONS; $T1$ IS PRECONDITIONING CPU TIME; $T2$ IS SOLVING CPU TIME; $R1$ IS THE SPEEDUP RATIO OF THE HYBRID SOLVER OVER ICCG WITH ILU(0); $R2$ IS THE SPEEDUP RATIO OF THE HYBRID SOLVER OVER ICCG WITH ILUT.

Ckt	N	E	ICCG with ILU(0)				ICCG with ILUT				Hybrid						R1	R2
			C	$M1$	I	$M2$	C	$M1$	I	$M2$	C	$M1$	I	$M2$	$T1(\text{sec})$	$T2(\text{sec})$		
m1	2.7e5	3.2e6	1.7e6	7.8e6	150	1.2e9	3.9e6	1.2e7	77	9.3e8	3.9e6	1.2e7	12	1.4e8	20.77	6.22	8.3	6.7
m2	4.3e5	5.2e6	2.8e6	1.3e7	122	1.5e9	6.5e6	2.0e7	62	1.2e9	6.5e6	2.0e7	12	2.3e8	33.00	11.90	6.6	5.3
m3	3.5e5	5.5e6	2.9e6	1.3e7	82	1.0e9	5.1e6	1.7e7	27	4.6e8	5.0e6	1.6e7	12	2.0e8	21.67	9.73	5.3	2.4
m4	4.6e5	8.2e6	4.3e6	1.9e7	110	2.1e9	7.5e6	2.5e7	55	1.4e9	8.0e6	2.5e7	13	3.3e8	46.91	17.07	6.3	4.2
m5	8.8e5	9.4e6	5.2e6	2.3e7	159	3.7e9	1.3e7	3.9e7	82	3.2e9	1.2e7	3.7e7	12	4.4e8	68.90	26.09	8.4	7.1

TABLE II

COMPUTATIONAL COMPLEXITY COMPARISON FOR THE ISPD02 BENCHMARKS, WITH 10^{-6} ERROR TOLERANCE. N , E , C , $M1$, I , $M2$, $T1$, $T2$, $R1$ AND $R2$ ARE AS DEFINED IN TABLE I.

Ckt	N	E	ICCG with ILU(0)				ICCG with ILUT				Hybrid						R1	R2
			C	$M1$	I	$M2$	C	$M1$	I	$M2$	C	$M1$	I	$M2$	$T1(\text{sec})$	$T2(\text{sec})$		
ibm10	9.3e4	5.9e5	3.4e5	1.7e6	130	2.1e8	9.4e5	2.9e6	42	1.2e8	1.0e6	2.9e6	19	5.4e7	7.00	1.42	4.0	2.2
ibm11	9.2e4	5.5e5	3.2e5	1.6e6	120	1.9e8	8.5e5	2.6e6	36	9.4e7	9.6e5	2.6e6	20	5.2e7	6.64	1.25	3.6	1.8
ibm12	9.5e4	6.4e5	3.7e5	1.7e6	110	1.9e8	1.0e6	3.1e6	47	1.4e8	1.1e6	3.0e6	20	6.0e7	7.50	1.54	3.2	2.4
ibm13	1.1e5	7.0e5	4.0e5	2.0e6	130	2.5e8	1.1e6	3.3e6	45	1.5e8	1.2e6	3.4e6	20	6.8e7	8.81	1.99	3.7	2.2
ibm14	1.9e5	1.1e6	6.5e5	3.2e6	145	4.6e8	1.7e6	5.3e6	53	2.8e8	2.0e6	5.3e6	26	1.4e8	15.93	4.88	3.3	2.0
ibm15	2.2e5	1.4e6	8.2e5	3.9e6	153	6.0e8	2.2e6	6.7e6	50	3.4e8	2.6e6	7.2e6	23	1.6e8	23.34	6.88	3.7	2.0
ibm16	2.5e5	1.6e6	9.1e5	4.4e6	170	7.5e8	2.5e6	7.6e6	62	4.7e8	2.9e6	7.9e6	21	1.7e8	24.09	6.50	4.5	2.9
ibm17	2.5e5	1.8e6	1.0e6	4.8e6	137	6.6e8	2.9e6	8.5e6	47	4.0e8	3.1e6	8.6e6	20	1.7e8	25.25	6.94	3.8	2.3
ibm18	2.7e5	1.7e6	9.8e5	4.8e6	191	9.1e8	2.6e6	8.0e6	65	5.2e8	3.1e6	8.5e6	23	2.0e8	30.67	8.10	4.7	2.7

TABLE III

RUNTIME COMPARISON INSIDE THE WATERLOO PLACER. $T3$ IS THE PLACER RUNTIME WITH ITS ORIGINAL SOLVER. $T4$ IS THE PLACER RUNTIME WITH THE HYBRID SOLVER. UNIT IS MINUTE.

Ckt	ibm10	ibm11	ibm12	ibm13	ibm14	ibm15	ibm16	ibm17
$T3$	53.4	42.1	57.1	54.1	106.7	117.0	123.9	181.4
$T4$	48.7	39.0	51.0	51.8	101.4	110.1	117.0	170.6

REFERENCES

- [1] S. Adya and I. Markov, ISPD02 Benchmarks [Online]. Available: <http://vlsicad.eecs.umich.edu/BK/ISPD02bench>.
- [2] P. R. Amestoy, T. A. Davis and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.
- [3] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. A. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia, PA: SIAM, 1994.
- [4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can recursive bisection alone produce routable placements?," in *Proc. DAC*, pp. 477–482, 2000.
- [5] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the ACM National Conference*, pp. 157–172, 1969.
- [6] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. DAC*, pp. 269–274, 1998.
- [7] J. H. Halton, "Sequential Monte Carlo," in *Proceedings of the Cambridge Philosophical Society*, vol. 58, pp. 57–78, 1962.
- [8] J. H. Halton, "Sequential Monte Carlo techniques for the solution of linear systems," *Journal of Scientific Computing*, vol. 9, pp. 213–257, 1994.
- [9] P. Heggernes, S. C. Eisenstat, G. Kumfert and A. Pothen, "The computational complexity of the Minimum Degree algorithm," in *Proceedings of 14th Norwegian Computer Science Conference*, pp. 98–109, 2001.
- [10] A. P. Hurst, P. Chong and A. Kuehlmann, "Physical placement driven by sequential timing analysis," in *Proc. ICCAD*, pp. 379–386, 2004.
- [11] D. S. Kershaw, "The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations," *Journal of Computational Physics*, vol. 26, pp. 43–65, 1978.
- [12] LASPack [Online]. Available: <http://www.tu-dresden.de/mwism/skalicky/laspack/laspack.html>.
- [13] A. W. Marshall, "The use of multi-stage sampling schemes in Monte Carlo," *Symposium of Monte Carlo Methods*, pp. 123–140. New York, NY: John Wiley & Sons, 1956.
- [14] H. Qian, S. R. Nassif, and S. S. Sapatnekar, "Random walks in a supply network," in *Proc. DAC*, pp. 93–98, 2003.
- [15] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA: SIAM, 2003.
- [16] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.
- [17] R. S. Varga, *Matrix Iterative Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1962.
- [18] N. Viswanathan and C. C. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. ISPD*, pp. 26–33, 2004.
- [19] K. P. Vorwerk, A. Kennings and A. Vannelli, "Engineering details of a stable force-directed placer," in *Proc. ICCAD*, pp. 573–580, 2004.