Quadratic Placement Revisited

M1.3 : Methodology (Techniques for Deep-Submicron Placement)

Abstract

The basic "quadratic placement" methodology is rooted in works by Wipfler et al. [36], Cheng and Kuh [8], Tsay et al. [32, 33, 34] and many others. It is also, reputedly, an approach that has been used in commercial and in-house tools for placement of standard-cell and gate-array designs. The quadratic placement methodology entails (i) solving recursively generated sparse systems of linear equations, each of which captures one-dimensional placement with a squared-wirelength objective (hence the term "quadratic"), along with (ii) a coupled min-cut partitioning step. In this paper, we dissect the implementation and motivations for quadratic placement: our goal is to provide design and usage intuition for deep-submicron CAD developers and users alike.

After providing necessary background discussion, our paper makes the following contributions. First, we discuss performance enhancements. We show that Krylov subspace engines for solving sparse systems of linear equations are significantly more effective than the successive over-relaxation (SOR) engine originally used in [32]. We also show that interaction between the linear system solver and the coupled min-cut partitioning step introduces novel convergence criteria for the solver. Thus, we propose an *order convergence* approach which can maintain solution quality while using substantially fewer solver iterations. Second, we discuss the motivations and relevance of the quadratic placement approach, in the context of past and future algorithmic technology, performance requirements, and design methodology. Quadratic placement arguably can be boiled down into a variant of classic "min-cut placement". We observe that use of the numerical linear systems solvers with quadratic wirelength objective may historically be due to the pre-1990's weakness of min-cut partitioners: i.e., numerical engines were *needed* to provide helpful hints to min-cut partitioners. Detailed experiments with modern partitioning technology show that the "quadratic placement approach" might no longer require a "quadratic" systems solver. And while quadratic placement remains the state of the art, there are several methodology drivers in deep-submicron performance-driven design that may soon require more novel approaches to the placement problem.

1 Introduction

In the physical implementation of high-performance, high-complexity deep-submicron ASICs, (standard-cell) placement has become a very important step. Given fixed decisions from the upstream stages of microarchitecture design, chip timing, chip planning, logic synthesis and physical floorplanning, it is placement solution quality that is the major determinant of whether timing correctness and routing completion will be achieved in the final layout. A typical ASIC or structured-custom flow might loop many times through (1) floorplanning, (2) timing budgeting, (3) placement, (4) global/detailed routing, (5) parasitic extraction, (6) delay calculation and static timing analysis, and (7) small layout-based resyntheses (ECO's involving gate resizing, buffer insertion, and buffer/wire sizing). (From (7), the loop is typically back to (3), but can be to (1) or earlier stages if necessary.) Reducing the number of passes through this loop is critical to achieving time-to-market, more complete exploration of the design space, and other competitive objectives.

For cell-based placement, the first-order objective has always been obvious: place connected cells closer together so as to reduce total routing and lower bounds on signal delay. This implies a minimum-wirelength placement objective.[1] Because there are many layout iterations, and because fast (constructive) placement estimation is needed in the floorplanner for design convergence, a placement tool must be extremely fast. As instance sizes grow larger, move-based (e.g., annealing) methods may be too slow except for detailed placement improvement. (Other results also suggest that solution quality from annealing will not scale as well with instance size as the solution quality from more "global" approaches [23, 18].) Due to its speed and "global" perspective, the so-called *quadratic placement* technique has received a great deal of attention throughout its development by such authors as Wipfler et al. [36], Fukunaga et al. [13], Cheng and Kuh [8], Tsay and Kuh [32] and others. Indeed, quadratic placement is reputedly an approach that has been used within commercial tools for placement of standard-cell and gate-array designs.

In this paper, we revisit the quadratic placement methodology and address both its mode of implementation and its relevance to future design flow requirements. Section 2 develops notation and synthesizes a generic model of quadratic placement which embodies (i) solution of recursively generated sparse systems of linear equations, each of which captures one-dimensional placement with a squared-wirelength objective, and (ii) a coupled min-cut partitioning step. Section 2 also observes that this generic quadratic placement will typically boil down to a variant of the classic "min-cut placement" approach. The remainder of the paper then dissects the implementations, motivations and purposes behind the quadratic placement approach, via this model. Section 3 briefly summarizes our experiments showing that Krylov-subspace solvers (in particular, biconjugate gradient stabilized) should be used in place of the successive over-relaxation (SOR) solver originally proposed in [32]; this substitution affords significant performance gains. Section 4 analyzes the interaction between the linear system solver and the coupled min-cut partitioning step, and based on this analysis proposes novel convergence criteria for the solver. Specifically, we propose an *order convergence* approach which can maintain solution quality while using substantially fewer solver iterations. Section 5 discusses

[1] Observe that the minimum-wirelength objective applies regardless of whether repeaters are inserted, wires are sized, RC or lumped-C delay models are used, etc. Also, observe that there are two other obvious objectives: balancing congestion, and satisfying path timing constraints [31]. While dense subcircuits must be spread out by the placer to maintain routability (i.e., routing congestion must be evenly distributed), this is a router-dependent issue and is addressed equally by (i) cell library layout and abstraction, (ii) better modeling of the router, and (iii) heuristics within placement. Likewise, path timing constraints must be satisfied in a performance-driven design, but this issue is addressed jointly by (i) synthesis, (ii) driver/buffer/wire sizing optimizations, and (iii) the placement itself. In this discussion, we do not treat congestion and timing as separate issues since they can be addressed by methodology or else folded into a "minimum weighted wirelength" placement objective. Finally, our discussion will focus on the minimum squared wirelength objective as a consequence of the quadratic approach. It should be obvious to the reader that timing criticality and other factors can still be captured through judicious weighting.

the motivations and relevance of the quadratic placement approach in the context of past and future algorithmic technology, performance requirements, and design methodology requirements. Starting from the observation that quadratic placement can be viewed as a variant of classic "min-cut placement", we offer the proposition that use of numerical linear systems solvers with quadratic wirelength objective is historically due to the pre-1990's weakness of min-cut partitioners. In other words, *numerical engines were needed to provide helpful hints to min-cut partitioners.* Detailed experiments with modern partitioning technology show that the generic "quadratic placement approach" may now be implementable without a "quadratic" system solver. And while quadratic placement remains the state of the art, there are several methodology drivers in deep-submicron performance-driven design that may soon require more novel approaches to the placement problem.

2 The Quadratic Placement Methodology

2.1 Notation and Definitions

A VLSI circuit is represented for placement by a weighted hypergraph, with n vertices corresponding to modules (vertex weights equal module areas), and hyperedges corresponding to signal nets (hyperedge weights equal criticalities and/or multiplicities). The two-dimensional layout region is represented as an array of legal placement locations. Placement seeks to place all cells of the design onto legal locations, such that no cells overlap and the chip timing and routability are optimized. As noted above, to first order this corresponds to a minimum wirelength objective. The placement problem is a form of quadratic assignment; and many variants are well-known to be NP-hard [14].

Since the numerical techniques used within the "quadratic placement" approach apply only to graphs (hypergraphs with all hyperedge sizes equal to 2), it is necessary to assume some transformation of hypergraphs to graphs via a *net model.* In this paper, we assume the standard clique model [25]; we have also obtained similar results for the clique model of [32] as well as a directed star model.[2]

The weighted graph $G = (V, E)$ can be represented by an $n \times n$ *adjacency matrix* $\mathbf{A} = (a_{ij})$ where $a_{ij} = w(v_i, v_j)$ if $(v_i, v_j) \in E$, and 0 otherwise.

Definition: The $n \times n$ *Laplacian* $\mathbf{Q} = (q_{ij})$ has entry q_{ij} equal to $-a_{ij}$ if $i \neq j$ and diagonal entry

[2]For a given multipin signal net e, $|e| \geq 2$, the graph edges that represent the net may be constructed in several ways, e.g., a directed star model, an unoriented star model [23, 29], or a clique model (see [4] for a review). In the clique model, $\frac{|e|\cdot(|e|-1)}{2}$ graph edges are created, connecting every pair of vertices in e. The PROUD tool [35] uses a clique model in which each clique edge has relative weight $\frac{8}{|e|^3}$. Lengauer [25] observes that the "standard" clique model has uniform edge weight $\frac{1}{|e|-1}$. See [17, 25] for reviews of other clique models.

The resulting weighted graph representation $G = (V, E)$ of the circuit topology has edge weights $w(v_i, v_j)$ derived by "superposing" all cliques in the obvious manner, i.e., adding clique edge weights for all nets that contain both vertices v_i and v_j. Because the clique model can yield dense and ill-conditioned matrices, in practice nets larger than a certain threshold are ignored, or are represented by randomly selecting $O(|e|)$ graph edges out of the $O(|e|^2)$ edges in the clique. Our results are obtained with thresholding of nets having size > 99 pins.

q_{ii} equal to $\sum_{j=1}^{n} a_{ij}$, i.e., the sum of edge weights incident to vertex v_i.

We will focus on the *linear placement problem*, which seeks to determine coordinates on the one-dimensional real line for all movable vertices, so as to minimize an objective function that depends on the edge weights of G and the vertex coordinates. The particular objective that we consider is minimum squared wirelength.

Definition: The n-dimensional *placement vector* $\mathbf{x} = (x_i)$ corresponds to the physical locations of modules $v_1, \ldots, v_n$ on the real line, i.e., x_i is the coordinate of vertex v_i.

Squared Wirelength Formulation: Minimize $\Phi(\mathbf{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a_{ij}(x_i - x_j)^2$ subject to $\mathbf{Hx} = \mathbf{b}$. This objective can be rewritten as $\Phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}$ (see, e.g., [16]).

We also consider "pad" constraints, which constrain the location of a module to equal some prescribed value (typically, such constraints are due to the pre-placement of I/O pads or other terminals).[3]

Often, the linear placement problem is solved within a top-down placement flow, i.e., a one-dimensional linear placement in the x (horizontal) direction is used to divide the circuit into left and right halves, after which a linear placement in the y (vertical) direction is used to divide the netlist further into quarters, etc. This is the basic idea within the vast majority of quadratic placers in the literature.

2.2 Essential Structure of a Quadratic Placer

We now review the essential components of the quadratic placement paradigm. Our goal here is to establish why quadratic placement methods have historically coupled numerical optimizations with min-cut optimizations or other means of "spreading out" a continuous force-directed placement. For obvious reasons, we make no attempt to provide a description of any commercial tool; as necessary, we will illustrate our discussion by referring to the PROUD algorithm of Tsay et al. [32, 33, 34].

Like other works, PROUD considers the squared wirelength objective $\Phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x}$, subject to the constraint[4] $\mathbf{x}^T\mathbf{x} = 1$. An unconstrained formulation is obtained by considering the objective function $\Phi(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x}$ without the slot constraint, but considering the possibility of fixed (pad) constraints. Assume that there are f fixed modules and c movable modules. The formulation is then written as:

[3]Without pad-type constraints, minimizing squared wirelength would be vacuous: collapsing all movable vertices onto one point would yield the best possible objective function value of $\Phi = 0$.

[4]This constraint is a second-order slot constraint; this nomenclature is due to [8] and refers to the n modules having to be placed onto n legal ("slot") locations. The first-order constraint is that the sum of the x_i values equals the sum of slot location coordinates; the second-order constraint is that the sum of squared x_i values equals the sum of squared slot location coordinates (normalized here to unity); etc. It is the requirement of placement onto discrete slots that makes the problem an intractable (NP-hard) form of quadratic assignment [14].

$$
\begin{aligned}
minimize \quad \Phi(\mathbf{x}) &= \begin{bmatrix} \mathbf{x}_c & \mathbf{x}_f \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{cc} & \mathbf{Q}_{cf} \\ \mathbf{Q}_{fc} & \mathbf{Q}_{ff} \end{bmatrix} \begin{bmatrix} \mathbf{x}_c \\ \mathbf{x}_f \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_c & \mathbf{x}_f \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{cc} x_c + \mathbf{Q}_{cf} \mathbf{x}_f \\ \mathbf{Q}_{fc} x_c + \mathbf{Q}_{ff} \mathbf{x}_f \end{bmatrix} \\
&= \mathbf{x}_c^T \mathbf{Q}_{cc} \mathbf{x}_c + \mathbf{x}_c^T \mathbf{Q}_{cf} \mathbf{x}_f + \mathbf{x}_f^T \mathbf{Q}_{fc} \mathbf{x}_c + \mathbf{x}_f^T \mathbf{Q}_{ff} \mathbf{x}_f
\end{aligned}
$$

where $\mathbf{x}_f$ denotes the vector of *fixed* module positions, $\mathbf{x}_c$ denotes the vector of *movable* module positions, and the Laplacian $\mathbf{Q}$ is partitioned into four corresponding parts $\mathbf{Q}_{cc}$, $\mathbf{Q}_{cf}$, $\mathbf{Q}_{fc}$ and $\mathbf{Q}_{ff}$ with $\mathbf{Q}_{cf}^T = \mathbf{Q}_{fc}$.

Because the optimal positions of all movable modules are inside the convex hull spanned by the fixed modules [35], we can consider the minimization problem for $\Phi(\mathbf{x})$ over this convex hull. Since $\Phi(\mathbf{x})$ is a smooth function over a compact set (in c-dimensional Euclidean space), maximal and minimal values are attained at extremal and/or boundary points; it suffices to consider extremal points only. To find extremal points, we look for zeros of the derivative of the objective function $\Phi(\mathbf{x})$:

$$\nabla\Phi(\mathbf{x}) = 2\mathbf{Q}_{cc}\mathbf{x}_c + 2\mathbf{Q}_{cf}\mathbf{x}_f = 0 \tag{1}$$

which yields

$$\mathbf{Q}_{cc}\mathbf{x}_c = -\mathbf{Q}_{cf}\mathbf{x}_f \tag{2}$$

The right-hand side of this system is a c-dimensional constant vector since $\mathbf{x}_f$ and $\mathbf{Q}_{fc}$ are known. Hence, $\Phi(\mathbf{x})$ is minimized by solving a $c \times c$ system of linear equations.

So far, this development is similar to that of other "force-directed" or "resistive network" analogies (see, e.g., [36, 8, 22, 13, 27]). The essential tradeoff is relaxing discrete slot constraints, and changing the "true" linear wirelength objective into a squared wirelength objective, in order to obtain a continuous quadratic minimization for which a global optimum can be found. Because Manhattan wirelength is separable into horizontal and vertical components, this quadratic optimization can be applied independently in both directions to obtain (x, y) coordinates for each movable module; Figure 1 shows a typical such "global placement" result, with module placements concentrated in the center of the layout region. The key question is how such a "global placement" (actually, a "continuous solution obtained using an incorrect objective") should be mapped back to the original discrete problem.

Two approaches have been used to obtain a feasible placement from a "global placement". The first approach is based on *assignment*, either in one step (to the entire 2-dimensional array of slots) or in two steps (to rows, and then to slots within rows) [13, 20]. The second approach, which has had a substantial impact on industry practice, is based on *partitioning*: the global placement result is used to determine a horizontal or vertical cut in the layout, and the continuous squared-wirelength

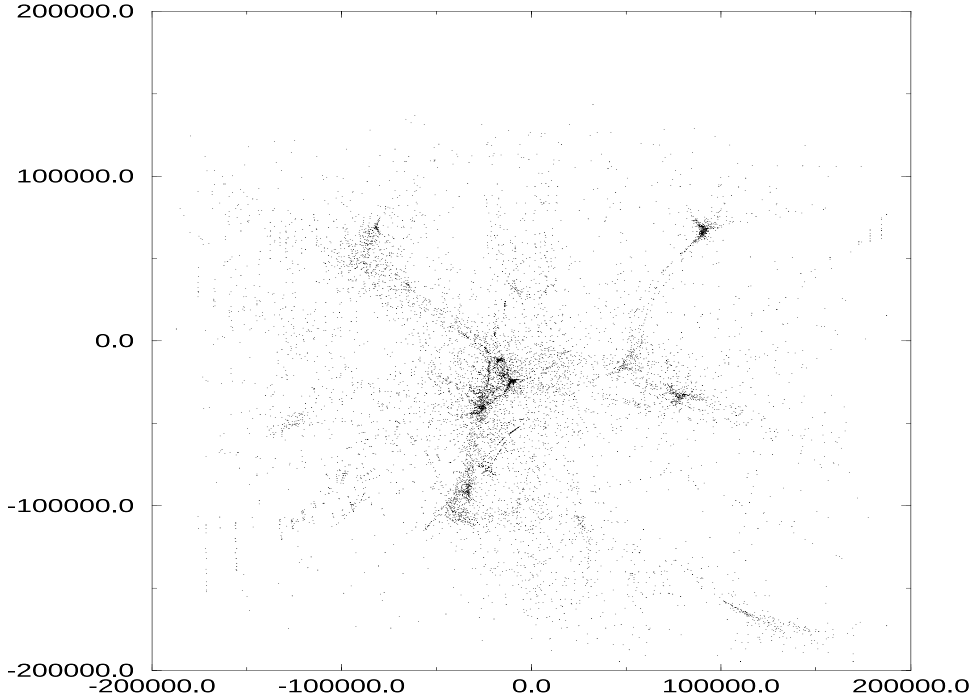

Figure 1: Two-dimensional global placement obtained by minimizing squared wirelength separately in the x and y directions. This example has 12146 cells and 711 I/O pads fixed on four sides of the layout.

optimization is then recursively called on the resulting subproblems (see [8, 22, 23, 27, 36]). The main difficulty with the partitioning-based approach is that the middle of the layout has extremely overlapped module placements: when inducing the recursive subproblems, the obvious "sort the coordinates and split in the middle" partition is highly sensitive to the convergence criteria used in the numerical optimization, and can yield poor results. Thus, iterative improvement is applied to the partition (see, e.g., [23]). It has been difficult for practitioners to imagine any objective for the iterative improvement other than some form of minimum weighted cut. Hence, quadratic placers can be quite similar in structure to top-down min-cut placers, except that cuts are induced from linear placements under the squared-wirelength objective.

Figure 2 describes the quadratic placer PROUD [32, 33, 34], which is more careful than previous works in how it applies partitioning during the top-down min-cut process. Its key innovation is in the analogy to "block Gauss-Seidel" (BGS) in Lines 4-8 of the Figure.[5] In PROUD, after (say) a vertical cut has been made along some centerline, the left and right halves must each be split by horizontal cuts. The BGS approach orders the left half vertically, then fixes its modules and performs terminal propagation (projection) to the centerline. These projected terminals will then influence the vertical ordering of the modules in the right half. The modules of the right half are then fixed and projected

[5] Roughly speaking, "BGS" connotes "keep one part of the solution fixed while working on another part, then fix the second part and work on the first part, etc."

to the centerline, where they influence a new vertical ordering of the left half. Eventually, both orderings converge and can be split to induce subproblems. In this way, PROUD affords more global interaction between siblings in the hierarchy and eliminates some order-dependence. The GORDIAN algorithm of [22, 23] is also innovative in its partitioning step. In GORDIAN, min-cut partitions are constructed from one-dimensional global placements, but are not enforced by a rigid spatial hierarchy as in previous methods. Rather, locations of modules belonging to a given recursive subproblem are constrained to have a common center of gravity; this tends to draw the modules to similar locations but allows some flexibility.[6]

PROUD Algorithm

Input: Netlist, I/O pad locations, Laplacian matrix Q, #BGS iterations
Output: Assignment of c movable cells to locations in the layout

1. Assign all movable modules to a single internal block, all pads to a single I/O block
2. **while** the current number of blocks is $> c$ **do**
3.   **for** $i = 1$ **to** # BGS iterations **do**
4.   **for** each internal block B **do**
5.   Set up the Laplacian $\mathbf{Q}_{jj}$ using $\mathbf{Q}$ for the j movable cells in B
6   Project modules outside B and pads to boundaries of B
            Compute $\mathbf{Q}_{kj}\mathbf{x}_k$ for these k projected modules and pads
7.   Use SOR to solve $\mathbf{Q}_{jj}\mathbf{x}_j = -\mathbf{Q}_{kj}\mathbf{x}_k$
8.   Temporarily cut B into two sub-blocks and assign modules to sub-blocks by sorting $\mathbf{x}_j$
        end for
    end for
9.   Fix the cut of each internal block and permanently assign modules to sublocks.
    Replace each block by its two sublocks.
    (If a block was created by a horizontal split, it will be divided into sublocks by a vertical split in the next iteration. Similarly, vertical splits are followed by horizontal splits.)
  end while
10. **return** final placement result

Figure 2: PROUD Algorithm Template

3 Performance Gains From Krylov-Subspace Solvers

Recall that quadratic placement entails the solution of a series of sparse systems of linear equations. These systems correspond to sparse matrices with dimension on the order of the number of movable modules in the original netlist. Recall too that convergence criteria must be fairly stringent so as to resolve the relative placements of modules near the middle of the layout. Hence, choice of

[6] We emphasize that PROUD and GORDIAN are noted specially for their creative solution to the mapping problem from global placement. Neither of their approaches is directly formulated as "minimum-cut"; indeed, a minimum-cut algorithm cannot easily handle GORDIAN's center of gravity constraints. However, there is a strong context of minimum-cut improvement in the original descriptions of both methods.

Solvers		Preconditioners						
		None	SOR/1.0	SSOR/1.0	SOR/1.95	SSOR/1.95	Jacobi	ILU
CG	Iter	245	—	106	—	316	245	97
	CPU time	39.69	—	37.24	—	110.50	40.74	34.31
GMRes	Iter	378	180	125	220	380	378	107
	CPU time	134.40	79.22	67.43	96.09	206.05	133.12	57.15
BiCGS	Iter	120	74	54	105	128	120	51
	CPU time	38.97	36.43	37.33	51.22	88.90	39.30	35.74
CGS	Iter	172	97	50	110	152	172	58
	CPU time	55.52	47.57	34.68	53.83	104.91	55.55	40.24
TFQMR	Iter	172	90	48	102	152	172	58
	CPU time	59.22	45.75	34.29	52.00	108.64	60.00	41.46
TCQMR	Iter	206	185	94	178	231	206	86
	CPU time	115.23	149.47	104.67	143.40	257.37	116.29	93.60

Table 1: Study of solvers and preconditioners for an industry test case with matrix dimension 29308 and 619492 nonzeros. The maximum number of nonzeros in any row is 137. CPU times are in seconds for a Pentium 150 running Linux 2.0.0; the g++ 2.7.2 compiler was used with -O -m486 option. Dashes (—) indicate that more than 1000 iterations were made, at which point the solver was terminated. The SSOR and SOR preconditioners were run with both $\omega = 1.0$ and $\omega = 1.95$. The same convergence test and tolerance values were used for all runs.

efficient solver and optimization of its application are key steps in developing a quadratic placer. In this section, we review our early experiments with iterative solvers such as Symmetric Successive Overrelaxation (SSOR), BiConjugate Gradient Stabilized (BiCGS), and others (see [6]).[7] The time complexity of such methods obviously depends on both the cost of a single iteration (which is constant during the solution of a given system) and the number of iterations needed until iterates adequately reflect the true solution.

The theory of iterative methods shows that the number of iterations needed to obtain a good approximation in norm depends on the spectrum of the matrix involved [15]. Hence, the idea of a preconditioner – a way to transform the original system to an equivalent one with "improved" spectrum. Because most implementations of preconditioners will update the system after each internal iteration, one must carefully examine the overall efficiency of solver/preconditioner combinations on particular classes of instances: more expensive iterations must be balanced against the number of iterations saved. We have solved a number of test systems with multiple combinations of solvers and preconditioners, and recorded number of iterations as well as CPU usage.

[7] Our motivation is that works such as [32] have used "early" (1950's era) iterative methods, specifically the successive over-relaxation (SOR) iteration, to solve the linear system in Equation (2). Many improved methods for solving sparse symmetric linear systems have been developed in recent years. Indeed, even our studies are not comprehensive, e.g., we did not test the most recent multi-level (multigrid) techniques, which seem quite promising.

Solvers		Preconditioners						
		None	SOR/1.0	SSOR/1.0	SOR/1.95	SSOR/1.95	Jacobi	ILU
CG	Iter	—		—	—	—	—	—
	CPU time	—		—	—	—	—	—
GMRes	Iter	—	—	—	—	—	—	—
	CPU time	—	—	—	—	—	—	—
BiCGS	Iter	324	225	174	—	437	324	118
	CPU time	359.38	370.39	407.50	—	1026.87	363.20	262.82
CGS	Iter	—	—	163	—	—	—	—
	CPU time	—	—	379.44	—	—	—	—
TFQMR	Iter	409	236	163	268	343	409	144
	CPU time	489.76	421.33	396.53	477.92	839.85	490.51	335.72
TCQMR	Iter	—	—	199	515	—	—	237
	CPU time	—	—	763.31	1445.71	—	—	856.15

Table 2: Study of solvers and preconditioners for an industry test case with matrix dimension 117440 and 1676320 nonzeros. The maximum number of nonzeros in any row is 68. CPU times are in seconds for a Pentium 150 running Linux 2.0.0; the g++ 2.7.2 compiler was used with -O -m486 option. Dashes (—) indicate that more than 1000 iterations were made, at which point the solver was terminated. The SSOR and SOR preconditioners were run with both $\omega = 1.0$ and $\omega = 1.95$. The same convergence test and tolerance value were used for all runs.

For pseudocodes of solvers and preconditioners see [6], and for analyses see [15].[8] We found that

[8] A brief review of the numerical methods used is as follows (see [6] for a comprehensive discussion). Iterative methods for solving large systems of linear equations can be classified as *stationary* or *nonstationary*. Stationary methods include Jacobi, Gauss-Seidel, Successive Overrelaxation (SOR) and Symmetric Successive Overrelaxation (SSOR). They are older, easier to implement and computationally cheaper per iteration. Nonstationary methods include Conjugate Gradient (CG), Generalized Minimal Residual (GMRes) and numerous variations. These are relatively newer and notably harder to implement and debug, but provide for much faster convergence. Additional computational expense per iteration (sometimes by a factor of 7) is normally justified by much smaller numbers of iterations. An iteration of the Jacobi method consists of solving once for every variable with respect to the other variables; convergence is slow. Gauss-Seidel uses updated values as soon as they are computed. SOR is a modification of Gauss-Seidel which depends on the extrapolation parameter ω (finding optimal values of ω is non-trivial, and heuristics are typically applied). SSOR has no computationl advantage over SOR as a solver, but is useful as a preconditioner; one iteration is roughly twice as expensive as an SOR iteration.

CG or GMRes will generate a sequence of orthogonal vectors which are residuals of the iterates. CG is very effective when the matrix is symmetric and positive definite, while GMRes is useful for general nonsymmetric matrices. Particular modifications of CG include BiConjugate Gradient (BiCG), Conjugate Gradient Squared (CGS), Quasi-Minimal Residual (QMR) and BiConjugate Gradient Stabilized (BiCGS). In BiCG, two sequences are generated which are only mutually orthogonal. The method is useful for nonsymmetric nonsingular matrices; convergence is irregular and the method can break down. QMR applies a least-squares update to smoothen the irregularity of BiCG; it is more reliable but rarely faster. Two further improvements are given by two variations of Transpose-Free Quasi-Minimal Residual (TFQMR and TCQMR). CGS is a fast method with even more irregular convergence than BiCG, while BiCGS is just another way to smoothen convergence of BiCG. (Based on our experiments, we chose BiCGS for its reliability and satisfactory convergence. When BiCGS is not applicable due to the nature of the matrices, we use GMRes.) Finally, the Chebyshev iteration computes coefficients of a polynomial minimizing the residual norm. The matrix must be positive definite for this iteration to be applied. Solvers which provide smooth convergence can be also used as preconditioners. Direct solvers present a different source of preconditioners for iterative methods, with examples being incomplete Cholesky (ICC), LU-factorization and incomplete LU-factorization (ILU). Here, "incomplete factorization" means that some nonzero matrix elements are ignored during the factorization.

BiConjugate Gradient Stabilized (BiCGS) was among the best solvers; though it does not guarantee convergence always, the method is good even for degenerate (not necessarily symmetric) matrices and in our experience provides more robust convergence than conjugate gradient (CG). As for preconditioners, we found Incomplete LU-factorization and the Successive Overrelaxation family (including SSOR) particularly successful. Note that in many of our tests, using the vacuous preconditioner was surprisingly competitive with using the best nontrivial preconditioner (iteration count was worse, but cheaper iterations led to low overall CPU usage). At the same time, using the wrong preconditioner could easily lead to a 3-fold loss in CPU time. The relative performance depicted in Tables 1 and 2 is representative of our results (in the tables, solver-preconditioner combinations which require more than 1000 iterations are omitted).[9] Table 3 shows the win of using BiCGS over SOR.

Linear system	Size	Nonzeros	BiCGS/SOR, $\omega = 1.0$		SOR, $\omega = 1.95$	
			Iterations	CPU time, sec	Iterations	CPU time, sec
avq_large	25114	159128	29	10.05	191	29.59
case1	29308	619492	88	76.03	256	98.98
case2	39917	732329	108	115.62	1000+	479.44
golem3	100281	371633	30	54.68	110	95.61

Table 3: We assembled *comparable* (i.e., using the same BLAS, etc.) implementations of the SOR and BiCGS (with SOR / $\omega = 1.0$ preconditioner) linear system solvers. Runtimes are for a Pentium 150 running Linux 2.0.0. Benchmarks case1 and case2 are matrices supplied from industry. The convergence test and tolerance values used here were slightly different from those used to generate Tables 1 and 2. Note that the solver implementations can be further optimized, i.e., CPU time usage should be evaluated for relative values only.

[9] Some solvers and preconditioners have variant implementations which can be better suited for different classes of linear systems (see, e.g., [11]) as well as parameters which influence convergence without affecting iteration time complexity. Our solver testbed is based on the PETSc (Portable, Extensible Toolkit for Scientific computation) library; this is free software developed and maintained at Argonne National Laboratory [5]. Salient features of PETSc include the following. (1) PETSc is an object-oriented library which can be linked to Fortran, C and C++ programs. It supports several sparse matrix formats, basic linear algebra for them, matrix factorization, matrix reordering and various linear system solvers. (2) PETSc is designed for multiprocessors and uses the Message Passing Interface (MPI); however, we mostly used it on uniprocessor workstations. (3) PETSc has error reporting and profiling capabilities; it can also report performance and diagnostic information, such as memory usage and number of floating point operations executed. It supports graphics, has several GUI tools for the X-window environment and is ported to a variety of UNIX-like operating systems (we used it on SunOS, Solaris and Linux). (4) Thorough documentation is available for PETSc and its solvers were the fastest of what we have tested. (Indeed, we have reviewed a number of sparse matrix-related software libraries. At the beginning of our project we were planning to use LASPACK, by T. Skalicky at the Dresden University of Technology. Our tests, however, showed that its solvers were too slow. We also looked seriously at Iterative Methods Library (IML++) and Sparse Matrix Library (SparseLib++), both written in C++ by R. Pozo, K. Remington and A. Lumsdaine at NIST. This software is much younger than PETSc and lacks somewhat in sophistication (e.g. in performance fine-tuning). However, it might be the best choice for projects smaller than ours.)

4 Order Convergence Criteria for Partitioning-Based Placement

Any solver from the previous section builds a sequence of iterates which converges to the correct solution $\mathbf{x}$ in Equation (2). How soon the iteration can be stopped determines performance. Typical convergence tests are based on some norm of the *residual* vector for an iterate[10]; this norm is taken to represent error with respect to the true solution. In practice, most norms are equivalent; heuristics (check convergence every j iterations, check differences of consecutive iterates rather than residual vectors, etc.) may be used to reduce the time spent on convergence tests.

In this section, we make the critical observation that a solver – when embedded in a quadratic placer – will ideally apply non-standard convergence criteria. *If we know that the final iterate will be sorted and split to induce an initial solution for the min-cut partitioner, then the iteration should stop when enough convergence has occurred with respect to the ensuing min-cut partitioning.* Put another way, using the linear placement solution to construct an initial min-cut partitioning solution discards a great deal of information; all that we eventually retain are memberships of vertices in "left" and "right" groups. At a minimum, the iteration should be stopped when the left and right groups stabilize. The proper goal is for the iteration to be stopped when subsequent changes in the left and right groups will not affect the eventual min-cut partitioning solution. Observe that this depends on the relative strength and stability of the partitioner.

We now define several heuristic alternatives for what we call *order convergence* criteria. These are based on ordering of coordinates in successive iterates, and are directly motivated by the quadratic placement context.

4.1 Order Convergence Criteria

Consider an iterate x_k from some linear system solver such as BiCGS. The *coordinate* for a module v_i is given by $x_k(i)$. The placer uses the relative sorted order of the coordinates, rather than their absolute values, to assign the modules to sub-blocks. We use a *direct permutation* π_k^+ to represent the ordering induced by the k^{th} iterate. If v_i is the j^{th} module in the ordering induced by x_k, then $\pi_k^+(j) = i$. The *inverse permutation* π_k^- is given by $\pi_k^-(i) = j$.

Table 4 shows the direct and inverse permutations for two iterates x_k and x_{k+1}; there are 11 coordinates with slots $0, 1, \ldots, N = 10$ in the example.

To measure the change in ordering between consecutive iterates, we propose two basic approaches. First, we can measure the maximum change in sorted index that any module experiences between iterates x_k and x_{k+1}:

[10] When solving the system $\mathbf{Ax} = \mathbf{b}$, the residual vector for a given iterate $\mathbf{x_k}$ is $\mathbf{b} - \mathbf{Ax_k}$.

Module Index	0	1	2	3	4	5	6	7	8	9	10
Iterate x_k	20	16	12	1	-99	8	23	99	1	11	-3
Iterate x_{k+1}	20	16	12	1	99	8	23	-99	1	11	-3
Direct permutations											
π_k^+	8	7	6	2	0	4	9	10	3	5	1
π_{k+1}^+	8	7	6	2	10	4	9	0	3	5	1
Inverse permutations											
π_k^-	4	10	3	8	5	9	2	1	0	6	7
π_{k+1}^-	7	10	3	8	5	9	2	1	0	6	4

Table 4: Example showing direct and inverse permutations for two iterates.

- $\mathbf{max\Delta^{++}} = max_{0 \le i \le N}\{|\pi_k^+(i) - \pi_{k+1}^+(i)|\}$

  This is the maximal placewise difference between two direct permutations, i.e., for each coordinate slot compute the two indices to which the values sort in consecutive iterates, then take the maximum difference of these two indices over all slots. In Table 4 $\max\Delta^{++} = 10$ since coordinate slot 4 has value 99 (index 10) in one iterate, and value -99 (index 0) in the other iterate.

In practice, a min-cut partitioner may be biased by locking modules that are at extreme indices in the sorted ordering. This decreases the size of the solution space and reduces runtime. Alternatively, modules might be left unlocked, but the starting solution constructed to capture extreme module coordinates (see the GORDIAN and PARABOLI methodologies, for example [23, 28]).

- $\mathbf{max\Delta^{++}_{10\%}} = max_{0 \le i \le \frac{N}{10};\ \frac{9N}{10} \le i \le N}\{|\pi_k^*(\pi_k^-(i)) - \pi_{k+1}^*(\pi_k^-(i))|\}$

  One may also compute maximal differences over only some percentage (in these examples, 20%) of the indices. In this example, we identify places with 10% smallest and 10% largest values and check how far these values "travel" in the next iterate (in terms of ordering index).

Finally, we have studied the measures

- $\mathbf{Flow^{\%\%}_{10\%}} = \#\{i | (0 \le i \le \frac{N}{10}$ **or** $\frac{9N}{10} \le i \le N)$ **and** $\frac{N}{10} \le \pi_{k+1}^+(\pi_k^-(i)) \le \frac{9N}{10}\}$

  $\mathbf{Flow^{Num}_{10\%}} = \#\{i | (0 \le i \le \frac{N}{10}$ **or** $\frac{9N}{10} \le i \le N)$
  and $x_k(\pi_k^-(\frac{N}{10})) + \varepsilon \le x_{k+1}(\pi_k^-(i)) \le x_k(\pi_k^-(\frac{9N}{10})) - \varepsilon\}$

  Here, we use %% to denote "percent", and # to denote "number of". The basic idea is to compute the "flow" of coordinates through some "percentage barrier". For example, $\mathbf{Flow^{\%\%}_{10\%}}$ computes how many of the lowest 10% and highest 10% of coordinates leave this range in the next iteration. $\mathbf{Flow^{Num}_{10\%}}$ detects the *numerical* range of the lowest 10% and highest 10% in the iterate x_k (i.e., two intervals $[min, a], [b, max]$) and computes how many of them leave this numerical range in the next iteration. In this case, we allow for absolute error ε.

4.2 Experiments With Order Convergence Criteria

We now give representative experimental data which yields intuitions regarding the proper "solver-partitioner interface" (i.e., the point at which solver iterations should be stopped, and an instance fed to the min-cut partitioner). Our examples are derived from benchmark netlists avq_large (25114 cells, 25144 nets and 87017 pins) and golem3 (103048 cells, 144949 nets and 339149 pins) available from the CAD Benchmarking Laboratory (http://www.cbl.ncsu.edu), as well as an industry-supplied test case, Case3 (12857 cells, 10880 nets and 58874 pins). Matrices were obtained as described in Section 2.2 for the x-direction ordering problem with standard clique model and thresholding of nets with > 99 pins. All iterates were saved from both SOR and BiCGS runs with convergence criterion being residual norm $< 10^{-4}$.

4.2.1 Interaction with the Partitioner

We analyzed SOR iterates for each of our test cases by sorting the coordinates and then inducing initial solutions for a Fiduccia-Mattheyses (FM) [12] min-cut partitioner. Our basic experimental protocol was as follows. (1) Initial solutions were induced by pre-seeding some percentage of leftmost and rightmost vertices in the ordering into the initial left and right partitions, respectively. Remaining vertices were assigned randomly into the initial left and right partitions. Three pre-seeding percentages were used: 0%, 20% and 50%; for space reasons we report the more interesting cases of 0% and 50% here. (2) No vertices were "locked" except for fixed pads, which were locked according to whether their coordinates were to the left or right of the median coordinate. The FM partitioner was free to move all non-pad vertices. (3) For each iterate, we generated 1, 10 or 30 pre-seeded initial solutions and ran FM from each. FM was invoked with unit module areas and an exact bisection requirement. (For golem3, we generated only 1 or 5 pre-seeded initial solutions.) (4) For each iterate, we recorded min/avg/max cut costs, as well as the exact composition of the best (min-cut) solution for the iterate.

To reduce experimental runtime, we analyzed only every fifth SOR iterate, i.e., iterates 1, 6, 11, etc. Tables 5, 6 and 7 show how FM solution quality varies with convergence of SOR, and with the amount of information retained from the iterate via pre-seeding. It is clear that strong (50%) pre-seeding prevents FM from returning very poor solutions, i.e., FM *does* need guidance from the solver iterate. At the same time, note that the marginal benefit of using a later iterate, as opposed to a relatively early iterate, is not obvious (cf. avq_large results): there is indeed a potential gain if we can apply order convergence criteria.

Iter	30 FMs				15 FMs				1 FM			
	0 %		50 %		0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	478(762.7)	0	431(499.6)	10	478(783.4)	0	431(506.3)	10	845	46	515	34
2	478(734.5)	0	422(498.2)	19	478(743.7)	0	422(495.6)	19	666	0	498	17
3	487(710.8)	124	426(489.4)	10	528(727.1)	118	426(488.8)	10	899	51	489	22
4	478(762.7)	0	401(490.5)	2	478(783.4)	0	423(484.9)	19	845	46	500	18
5	478(734.5)	0	440(492.9)	0	478(743.7)	0	455(495.1)	25	666	0	485	39
6	478(725.2)	0	425(489.1)	10	478(727.6)	0	425(485.6)	10	666	0	483	18
7	474(741.1)	67	434(497.0)	4	485(735.6)	39	464(504.8)	1	673	60	534	14
8	505(740.4)	1	424(495.4)	11	532(764.2)	49	424(492.3)	11	666	0	502	20
9	478(762.7)	0	431(493.9)	18	478(783.4)	0	431(497.5)	18	845	46	498	18
10	491(742.1)	14	424(492.2)	10	595(754.2)	120	424(491.7)	10	848	91	479	38
11	470(745.2)	114	413(490.4)	3	533(738.6)	57	413(479.5)	3	856	63	503	5
12	478(756.5)	0	424(488.6)	10	478(791.7)	0	424(487.4)	10	845	46	485	39
13	505(747.4)	101	445(503.8)	24	628(790.2)	35	461(510.4)	12	674	91	536	9
14	487(736.7)	124	422(493.6)	18	534(795.5)	98	422(505.0)	18	899	51	507	36
15	478(762.7)	0	428(497.8)	10	478(783.4)	0	428(497.0)	10	845	46	513	6
16	478(734.5)	0	428(490.9)	10	478(743.7)	0	428(496.5)	10	666	0	499	14
17	468(747.8)	2	436(509.3)	8	468(732.0)	2	463(511.4)	26	863	95	505	2
18	509(713.6)	78	423(484.0)	11	654(731.6)	79	423(484.9)	11	799	20	469	39
19	509(744.9)	78	432(493.6)	3	534(747.1)	98	453(496.5)	12	640	50	485	24
20	505(747.4)	101	406(487.7)	11	628(790.2)	35	406(492.4)	11	674	91	502	20
21	505(747.4)	101	440(492.7)	6	628(790.2)	35	440(491.0)	6	674	91	482	16
22	509(744.9)	78	418(492.9)	7	534(747.1)	98	434(488.2)	17	640	50	485	39
23	509(708.3)	78	411(498.3)	7	573(707.3)	78	425(485.1)	19	799	20	493	3
24	505(736.6)	110	423(502.9)	2	639(797.4)	28	440(485.4)	26	698	2	452	35
25	543(753.2)	14	426(493.9)	10	543(764.7)	14	426(494.2)	10	863	95	513	8
26	543(753.2)	14	426(491.8)	13	543(764.7)	14	426(491.4)	13	863	95	522	14
27	509(739.7)	78	450(497.9)	19	532(747.6)	49	458(497.4)	21	666	0	486	16
28	478(734.5)	0	440(494.7)	5	478(743.7)	0	454(493.3)	14	666	0	521	15
29	559(749.4)	141	405(504.9)	6	595(790.1)	15	476(514.5)	16	686	77	507	6
30	478(734.5)	0	442(491.0)	6	478(743.7)	0	461(494.8)	7	666	0	483	18
31	478(734.5)	0	441(491.6)	12	478(743.7)	0	444(491.5)	10	666	0	513	6
32	505(747.4)	101	415(493.2)	6	628(790.2)	35	427(491.5)	8	674	91	502	15
33	478(734.5)	0	424(487.5)	10	478(743.7)	0	424(483.3)	10	666	0	483	18
34	467(698.0)	92	449(505.8)	5	467(699.2)	92	469(510.1)	32	601	80	500	15
35	505(736.6)	110	423(502.7)	18	639(797.4)	28	423(512.4)	18	698	2	515	7
36	478(734.5)	0	427(490.5)	11	478(743.7)	0	427(496.5)	11	666	0	513	6
37	478(734.5)	0	428(499.3)	6	478(743.7)	0	442(501.0)	30	666	0	483	8
38	478(734.5)	0	435(504.0)	2	478(743.7)	0	469(507.7)	32	666	0	500	15
39	478(734.5)	0	419(496.9)	16	478(743.7)	0	419(493.9)	16	666	0	521	11
40	478(734.5)	0	424(500.2)	18	478(743.7)	0	424(497.6)	18	666	0	501	21
41	478(734.5)	0	428(496.5)	21	478(743.7)	0	428(492.7)	21	666	0	501	21
42	478(734.5)	0	424(492.7)	18	478(743.7)	0	424(492.8)	18	666	0	501	21
43	478(734.5)	0	436(485.6)	0	478(743.7)	0	453(492.3)	0	666	0	509	0

Table 5: Min/Avg cut cost and Hamming distance (shown in HD column) for Case3 in 30/10/1 FM runs with 0% and 50% pre-seeded.

The Tables also analyze the best solutions found by FM for each iterate. Since the standard quadratic placement methodology uses the last iterate to induce an initial solution for min-cut partitioning, we also compare the best solution for each iterate against the best solution for the last iterate. In this comparison, to guard against the possibility of finding cuts with similar costs but different structures, we report the "Hamming distance" between the two partitioning solutions.[11]

[11] "Hamming distance" is the minimum number of vertices that need to be moved for one solution to be transformed into the other. Some literature (reviewed in [4]) has previously established that the best partitioning solutions for an instance are likely to be similar in terms of both structure and cut cost. However, to ensure that early termination of

Iter	30 FMs				15 FMs				1 FM			
	0 %		50 %		0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	621(899.6)	334	475(475.1)	0	627(842.1)	72	475(475.3)	0	935	56	475	0
2	567(912.1)	122	475(475.2)	1	669(965.7)	99	475(475.3)	1	1152	11	478	119
3	551(911.0)	319	475(475.4)	0	551(883.7)	130	475(475.4)	0	869	213	475	0
4	567(876.3)	23	475(475.3)	0	691(885.1)	177	475(475.0)	0	891	170	475	0
5	472(913.5)	125	475(475.0)	1	807(975.8)	183	475(475.0)	1	1052	243	475	1
6	482(844.8)	59	475(475.2)	0	482(788.4)	130	475(475.1)	0	926	12	475	0
7	635(896.0)	127	475(475.1)	0	869(963.4)	20	475(475.3)	0	926	250	475	0
8	621(899.6)	334	475(475.0)	0	627(842.1)	72	475(475.0)	0	935	56	475	0
9	621(899.6)	334	475(475.0)	0	627(842.1)	72	475(475.0)	0	935	56	475	0
10	522(891.3)	75	475(475.4)	1	608(830.5)	11	475(475.0)	1	671	332	475	1
11	577(939.4)	19	475(475.0)	0	577(908.0)	170	475(475.0)	0	577	224	475	0
12	433(857.5)	132	475(475.2)	2	433(798.9)	57	475(475.0)	2	737	216	475	2
13	597(895.9)	76	475(475.0)	0	758(948.5)	325	475(475.0)	0	1023	176	475	0
14	603(897.0)	472	475(475.0)	1	603(913.3)	283	475(475.0)	1	927	183	475	1
15	754(911.6)	191	475(475.0)	0	789(899.8)	185	475(475.0)	0	953	222	475	0
16	495(853.9)	111	475(475.1)	0	726(905.2)	152	475(475.3)	0	726	206	475	0
17	420(918.3)	57	475(475.5)	1	701(948.7)	67	475(475.3)	1	1101	42	475	1
18	424(882.9)	94	475(475.0)	1	477(882.5)	132	475(475.0)	1	477	186	475	1
19	621(899.6)	334	475(475.3)	0	627(842.1)	72	475(475.1)	0	935	56	475	0
20	581(914.5)	111	475(475.2)	0	581(871.0)	78	475(475.2)	0	957	112	475	0
21	507(908.1)	28	475(475.1)	2	730(915.4)	266	475(475.0)	2	754	16	475	2
22	618(940.9)	79	475(475.1)	0	742(961.2)	13	475(475.0)	0	742	41	475	0
23	486(888.6)	30	475(475.1)	0	741(934.0)	141	475(475.0)	0	980	174	475	0
24	625(903.8)	69	475(475.0)	0	804(937.6)	63	475(475.0)	0	1049	49	475	0
25	628(926.5)	149	475(475.1)	0	659(944.8)	33	475(475.3)	0	790	188	475	0
26	466(854.6)	355	475(475.0)	1	466(776.2)	166	475(475.0)	1	475	163	475	1
27	537(883.4)	78	475(475.0)	1	537(846.5)	111	475(475.0)	1	830	178	475	1
28	585(948.5)	10	475(475.1)	0	705(918.8)	31	475(475.0)	0	937	132	475	0
29	546(941.4)	78	475(475.0)	0	546(921.5)	111	475(475.0)	0	546	165	475	0
30	503(940.6)	158	475(475.0)	0	503(903.7)	31	475(475.0)	0	503	85	475	0
31	412(913.4)	20	475(475.3)	0	470(945.6)	162	475(475.1)	0	987	226	475	0
32	604(883.1)	177	475(475.0)	0	688(888.0)	63	475(475.0)	0	752	18	475	0
33	423(888.7)	23	475(475.4)	0	772(984.4)	170	475(475.4)	0	1114	47	475	0
34	515(915.0)	101	475(475.0)	1	672(897.5)	207	475(475.0)	1	1010	21	475	1
35	376(872.3)	29	475(475.0)	0	630(948.2)	84	475(475.0)	0	967	254	475	0
36	576(949.5)	63	475(475.0)	0	814(960.1)	189	475(475.0)	0	1045	99	475	0
37	633(885.7)	0	475(475.0)	0	656(942.6)	0	475(475.0)	0	986	0	475	0

Table 6: Min/Avg cut cost and Hamming distance (shown in HD column) for avq_large in 30/10/1 FM runs with 0% and 50% pre-seeded.

4.2.2 Order Convergence Studies

Tables 8 and 9 show order convergence results between consecutive solver iterates for both SOR and BiCGS (with SOR preconditioner). We can see that the Flow-based measures **FlowNum**%% seem particularly promising. There is some indication that the SOR solver advances in discernible "phases"; correlating such behavior against results from Subsection 4.2.1 studies seems worthwhile for developers to pursue. The Tables also show that with a Krylov-subspace method the convergence is not only much faster versus SOR, but also much smoother with respect to our order convergence

the solver will not affect the final result, we measure Hamming distances to confirm similarity in both solution cost and solution structure.

Iter	5 FMs				1 FM			
	0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	3804(4222.6)	419	3434(4043.4)	284	3804	628	3662	337
2	3218(3835.8)	425	3545(3897.4)	311	4127	320	4094	667
3	3482(3847.2)	21	3865(3979.8)	36	3777	333	3937	678
4	3415(3979.2)	154	2872(3910.8)	106	3703	138	4033	820
5	3218(3866.0)	425	3573(3872.2)	186	4433	238	4002	389
6	3218(3501.0)	425	3421(3918.2)	486	3349	180	3421	164
7	3218(3714.4)	425	3869(4185.6)	106	3915	281	4097	612
8	3645(3948.4)	141	3926(4278.4)	206	3645	350	4443	455
9	3218(3772.2)	425	3623(4007.8)	309	4010	355	3901	465
10	3277(3803.0)	203	3402(3932.0)	125	4428	83	4077	576
11	3218(3835.8)	425	3986(4123.0)	79	4127	320	4035	982
12	3218(3501.0)	425	3958(4064.2)	485	3349	180	4147	579
13	3716(4071.2)	431	3457(3972.0)	402	3952	74	3988	126
14	3218(3835.8)	425	3674(3898.4)	183	4127	320	3846	908
15	3218(3928.0)	425	3389(3899.2)	484	4433	238	3389	806
16	3243(3884.2)	4	3456(3824.8)	82	3855	0	4011	658
17	3218(3488.4)	425	3835(4225.2)	474	3349	180	4305	326
18	3218(3846.4)	425	3967(4180.8)	498	3915	281	4557	422
19	3218(3488.4)	425	3790(4247.8)	174	3349	180	4162	1064
20	3218(3501.0)	425	3620(3891.4)	170	3349	180	3620	492
21	3218(3846.4)	425	3433(3675.2)	299	3915	281	3713	783
22	3768(4213.6)	21	3546(3987.8)	326	3768	188	4429	351
23	3218(3835.8)	425	3899(4183.8)	103	4127	320	3899	425
24	3511(4150.4)	203	3864(4004.0)	110	4814	76	3926	771
25	3610(3883.6)	0	3840(4017.2)	0	3855	0	3898	0

Table 7: Min/Avg cut cost and Hamming distance (shown in HD column) for golem3 in 5 and 1 FM runs with 0% and 50% pre-seeded.

criteria.[12]

5 Quadratic Placement Revisited

Recall that the quadratic placement approach can be viewed as a variant of classic "min-cut placement". The previous section showed that a traditional FM min-cut partitioner can benefit a great deal from the hints contained in solver iterates. We now propose an interesting notion: that the use of numerical linear systems solvers with quadratic wirelength objective is historically due to the pre-1990's weakness of min-cut partitioners. Specifically, we show that a truly leading-edge, modern partitioner needs no such hints, i.e., the "quadratic placement" result can effectively be obtained without a "quadratic" system solver. We will then conclude the paper by noting several factors, including deep-submicron design methodology and performance requirements, which suggest that more novel approaches to VLSI placement will be needed in the future.

[12] We do not address the question of efficient tests for order convergence. Certainly, actual sorting would never be used in such tests, and certain variants would be much less practical than others. For most of the tests that seem promising, however, judicious use of sampling and linear-time order statistics (k^{th}-largest element finding) will likely yield acceptable complexity.

SOR iterates				
Iteration	maxΔ++10%	Flow%%10%	FlowNum10%	FlowNum20%
2	9957	1605	420	1921
3	4007	139	160	286
4	3978	121	98	112
5	4068	96	106	49
6	4017	71	66	114
7	4054	62	65	48
8	4054	43	40	44
9	4042	38	38	29
10	4043	25	23	25
11	4047	16	26	18
12	9667	926	781	807
13	5879	926	1275	1232
14	3870	8	10	6
15	3922	8	7	4
16	97	6	6	3
17	4054	4	9	8
18	55	5	2	1
19	4051	4	2	1
20	3922	2	5	1
21	22	2	0	1
22	3921	3	3	3
23	11937	1378	1270	1564
24	11937	1378	822	643
25	4052	2	1	1
26	8	1	0	0
27	9	1	1	0
28	8	0	0	0
29	9	0	0	0
30	9	0	0	0
31	7	0	0	0
32	7	0	0	0
33	5	0	0	0
34	5658	548	901	737
35	5407	548	359	716
36	5	0	0	0
37	4	0	0	0
38	4	0	0	0
39	4	0	0	0
40	5	0	0	0
41	3941	441	1073	342
42	10047	715	511	708
43	4185	700	975	735
44	4066	567	413	539
45	4012	295	158	202
BiCGS iterates				
Iteration	maxΔ++10%	Flow%%10%	FlowNum10%	FlowNum20%
2	24785	2420	315	315
3	23051	2210	47	47
4	20228	1518	0	0
5	16506	2159	0	0
6	4951	1639	0	0

Table 8: SOR- and BiCG-order convergence for case3. Every fifth iterate was saved for each solver. Numbers start with 2 because order convergence tests measure changes between successive iterates.

5.1 Modern Partitioners Don't Need Hints

In the past two years, very significant improvements to FM partitioning technology have been made, primarily in the areas of *tie-breaking* and *multi-level implementation*.[13] Tie-breaking is used to choose among alternate moves that have the same gain; the idea was pioneered in the "lookahead" gain vector of Krishnamurthy [24], and two very strong probabilistic and clustering-based methods have recently been proposed by Dutt and Deng [9] [10] (see [4] for a survey). The multi-level approach integrates hierarchical clustering into FM, generalizing the early "two-phase" approach (e.g., [7]). In

[13] Up through 1994, the state-of-the-art bipartitioning method was PARABOLI [28], which relied on strong hints from a linear placement under linear wirelength objective. The methods that we now cite, all from the 1995-1996 time frame, give substantially better results than PARABOLI.

SOR iterates				
Iteration	maxΔ++10%	Flow%%10%	FlowNum10%	FlowNum20%
2	24791	4156	2512	5023
3	955	152	1024	952
4	689	108	1031	922
5	820	115	1112	894
6	937	86	1118	871
7	1009	82	998	841
8	750	101	821	779
9	739	53	648	761
10	728	41	631	745
11	629	34	749	680
12	20363	2815	1537	3913
13	22586	2816	2512	5023
14	562	48	691	563
15	544	24	603	538
16	529	23	497	529
17	472	24	455	493
18	421	27	417	478
19	365	59	407	425
20	289	26	361	411
21	256	21	344	379
22	227	20	294	303
23	13878	1863	1245	3661
24	19992	1862	2501	4703
25	181	16	207	238
26	199	19	184	161
27	169	23	153	182
28	194	12	134	122
29	213	10	96	91
30	153	10	88	87
31	126	12	74	90
32	178	8	58	64
33	21201	1058	1425	3780
34	21641	543	817	2331
35	5244	239	1014	1932
36	2668	187	900	1400
37	1245	206	980	1073
38	1104	252	779	1023
BiCGS iterates				
Iteration	maxΔ++10%	Flow%%10%	FlowNum10%	FlowNum20%
2	8850	1107	364	603
3	4107	427	162	208
4	4064	60	46	36
5	3925	3	3	3
6	13	1	0	0
7	20	4	0	0
8	9	1	0	0

Table 9: SOR- and BiCGS-order convergence for avq_large. Every fifth iterate was saved for each solver. Numbers start with 2 because order convergence tests measure changes between successive iterates.

the multi-level approach, an initial graph G_0 is clustered to induce the *coarsened* graph G_1, then a clustering of G_1 induces a coarser G_2, etc. until a coarsest graph G_m is constructed. A partitioning solution $P_m = \{X_m, Y_m\}$ is found for G_m (e.g., via FM) and this solution is flattened by one level into $P_{m-1} = \{X_{m-1}, Y_{m-1}\}$. The solution P_{m-1} is then improved, e.g., by applying FM. The *uncoarsening* process continues until a partitioning of the original graph G_0 is derived. Multilevel methods have been well-studied in the scientific computing community [19] [21]. Alpert et al. [3] adapted the Metis package of [21] to VLSI partitioning, and obtained bipartitioning runtimes of under 3 seconds of CPU time on a Sun Sparc-5 for graphs with more than 15,000 vertices and 91,000 edges. A more recent multi-level code [1] was reported to improve solution quality over all previous works, including [28] [9] [10]; we have obtained this partitioning code [2] and integrated it into our testbed.

We repeated the experiments of the previous section using the multi-level FM (ML-FM) partitioner of [1]. Tables 10, 11, and 12 show how ML-FM solution quality varies with convergence of SOR, and with the amount of information retained from the iterate.[14] The conclusions are clear: ML-FM dramatically outperforms FM, and furthermore draws *zero* benefit from using solver iterates (indeed, ML-FM can be hurt when constrained to follow the structure of an iterate in its initial solution). Hamming distance results in the Tables also show that the ML-FM solutions are extremely similar in terms of both structure *and* cut cost, no matter what initial solution is chosen. We observe that if a minimum-cut solution is desired, a viable approach may be to simply call ML-FM on a random initial solution.

5.2 Discussion and Futures

We have synthesized the motivations and structure for a generic "quadratic placement" methodology, and given both performance improvements and a historical context for the approach. We have also shown that – possibly – an implementation of the approach no longer requires an embedded linear systems solver, and indeed can revert back to "min-cut placement" when armed with the latest partitioning technology. We by no means claim that every existing quadratic placer should discard its numerical engine, but – *ceteris paribus* (all else being equal) – we suspect that if a minimum weighted cut is the true objective, ML-FM or other recent partitioners might be invoked with no loss of solution quality.[15]

With respect to the deep-submicron ASIC and structured-custom design methdology, we believe that there are basic drivers that suggest looking beyond "quadratic placement" technology. First, there are well-known limitations to modeling capability within a quadratic placer: path timing constraints, invariance of orderings to "stretched" coordinates (e.g., when horizontal and vertical routing resources differ), the requirement of pre-placed pads to "anchor" the placement, etc. Second, a top-down, performance- and HDL-driven design methodology will have relatively smaller synthesized blocks: in order to gain some predictability for synthesis output, timing, global net topologies, etc. these will not be large enough for a quadratic placer to show its "global awareness" and runtime advantages over an annealing placer. Third, the advent of block-based design (cf. RAPID, Pinnacles, Virtual Socket Initiative, etc.) may mean that the days of large, flat problems are numbered.

[14] Given an initial bipartitioning solution and an 0.5 level of preassignment, ML-FM determines a bottom-up clustering (coarsening) that is compatible with this solution throughout the hierarchy. With an 0.0 level of preassignment, ML-FM has no constraints on its bottom-up clustering.

[15] The reader may observe that we have used only "top-level" instances to test our hypotheses. We chose to use these instances because they have the fewest "pad" constraints and thus the greatest leeway within which a partitioner might benefit from solver hints. (One might argue that when there are relatively many "pad" constraints due to terminal propagation and Rent's rule, the benefits from a strong partitioner will decrease. This is true, but the benefits from solver hints will also decrease correspondingly in such a situation.) The reader may also observe that extremely good min-cuts can underutilize routing resources across the cutline. The fact remains that quadratic placement approaches in the literature do attempt to minimize cutsize when defining hierarchical subproblems. If a quadratic placer were to *benefit* from having a bad partitioner, this would of course bring into question both the objective and the role of partitioning in quadratic placement.)

Iter	15 ML-FMs				5 ML-FMs				1 ML-FM			
	0 %		50 %		0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	318(336.8)	162	329(391.9)	4	330(336.8)	50	329(378.6)	7	310	142	351	8
2	313(334.8)	162	339(393.5)	125	322(336.9)	50	343(368.2)	8	305	139	352	14
3	308(332.2)	156	336(393.4)	11	313(337.0)	34	380(400.2)	229	338	10	393	11
4	308(331.8)	157	333(390.7)	5	313(337.4)	19	345(400.0)	1	336	8	399	27
5	303(327.0)	8	335(380.5)	4	316(329.6)	130	335(387.2)	1	333	9	415	15
6	316(336.3)	15	340(386.3)	4	328(341.2)	139	348(386.0)	8	360	215	348	54
7	301(331.5)	169	340(380.7)	121	314(329.8)	102	340(379.8)	124	330	8	381	4
8	307(333.1)	17	335(382.7)	9	314(330.4)	14	335(370.8)	6	338	18	386	29
9	309(336.8)	13	361(391.9)	43	332(338.6)	111	371(384.6)	23	339	6	383	44
10	306(332.7)	8	345(387.7)	6	315(333.6)	116	392(403.0)	16	315	16	392	30
11	312(343.7)	21	328(381.0)	10	339(350.2)	141	328(379.4)	13	340	6	374	29
12	300(337.7)	13	325(372.1)	15	333(351.8)	133	341(378.0)	39	317	14	355	54
13	300(332.3)	13	343(382.1)	20	300(329.4)	13	349(370.8)	2	330	12	354	6
14	318(333.2)	98	326(384.3)	9	329(334.6)	125	339(380.8)	5	321	24	344	51
15	309(334.0)	149	340(378.7)	1	309(329.2)	149	350(372.8)	23	343	188	361	8
16	309(334.5)	149	335(380.0)	7	309(330.0)	149	360(381.0)	4	329	7	416	10
17	304(334.9)	78	340(384.3)	123	341(343.0)	115	340(384.4)	126	306	121	340	80
18	304(329.0)	78	338(386.2)	107	312(334.6)	19	343(401.8)	90	337	193	466	4
19	304(329.6)	78	336(372.3)	98	304(329.2)	78	336(368.4)	101	327	122	403	175
20	309(332.2)	71	342(382.7)	12	309(324.6)	71	356(388.2)	6	336	3	390	39
21	308(324.1)	6	334(371.9)	0	312(327.2)	148	337(363.0)	16	326	8	343	10
22	308(328.0)	6	351(389.8)	3	308(325.4)	6	390(399.0)	72	335	36	408	6
23	295(330.3)	80	340(381.1)	8	324(337.8)	16	363(391.8)	145	308	24	363	99
24	301(329.3)	164	327(374.9)	120	301(327.4)	164	343(374.8)	21	318	6	353	64
25	313(332.5)	7	327(380.7)	77	317(334.4)	65	327(369.0)	74	330	10	327	120
26	300(330.7)	17	332(385.8)	103	300(335.2)	17	340(376.8)	3	336	10	385	77
27	306(329.3)	146	338(386.3)	3	307(325.6)	73	338(390.4)	6	337	17	426	45
28	306(336.5)	146	326(383.5)	7	323(335.4)	4	332(377.8)	5	325	66	382	15
29	295(331.2)	42	345(387.4)	47	306(328.4)	146	398(400.2)	30	319	128	401	35
30	295(335.1)	42	329(379.4)	5	314(340.4)	84	329(362.6)	2	339	2	329	48
31	312(334.7)	129	334(381.5)	120	325(339.6)	144	334(352.6)	123	359	179	334	77
32	312(336.4)	129	336(392.0)	3	312(335.2)	129	384(406.0)	53	339	16	384	7
33	306(333.8)	22	322(375.5)	17	308(328.8)	18	339(379.2)	136	343	11	339	90
34	306(336.3)	22	345(382.8)	5	306(333.6)	22	353(384.8)	41	344	13	384	27
35	305(338.7)	171	343(390.0)	44	334(338.6)	104	348(390.0)	12	323	19	390	0
36	313(336.1)	103	344(387.7)	34	330(342.8)	4	348(383.0)	2	344	93	348	44
37	305(331.6)	18	334(382.2)	14	313(331.6)	103	341(381.0)	8	337	6	394	7
38	306(337.1)	169	333(380.1)	2	332(337.4)	130	333(379.8)	1	314	127	388	34
39	322(340.9)	102	346(390.5)	8	326(336.2)	23	366(395.0)	8	328	6	374	43
40	316(344.3)	73	335(382.5)	0	324(349.2)	8	335(371.8)	3	344	13	422	29
41	316(339.4)	2	360(396.4)	39	316(334.0)	2	382(392.2)	11	321	91	382	35
42	312(331.2)	0	334(382.4)	11	314(332.6)	29	382(408.6)	162	333	12	382	116
43	312(330.3)	0	324(386.3)	0	312(332.8)	0	350(397.8)	0	322	0	412	0

Table 10: Min/Avg cut cost and Hamming distance (shown in HD column) for Case3 in 15/5/1 ML-FM runs with 0% and 50% pre-seeded.

Synthesized glue logic will be spread out over disconnected, heterogeneous regions; the resulting chip planning - block building - assembly flow harkens back to classic floorplanning and global routing issues, and does not play to the strengths of quadratic placement. Certainly, quadratic placers are the state of the art today. But it remains to be seen whether other approaches will more effectively solve tomorrow's placement problems.

Iter	15 ML-FMs				5 ML-FMs				1 ML-FM			
	0 %		50 %		0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	190(218.3)	94	206(250.3)	31	190(212.4)	94	222(294.6)	232	218	145	222	20
2	193(212.1)	203	197(219.4)	339	195(212.0)	149	197(215.8)	244	199	258	221	71
3	198(214.9)	191	196(245.2)	92	198(211.8)	191	216(244.6)	58	247	203	229	49
4	180(213.1)	157	192(269.7)	347	203(216.4)	90	204(222.2)	264	251	74	228	43
5	191(213.2)	193	202(257.1)	326	203(214.6)	112	220(320.6)	14	211	153	220	266
6	198(222.9)	155	205(238.8)	108	210(223.2)	191	205(269.8)	13	206	82	235	106
7	201(221.2)	124	184(248.1)	315	201(217.2)	124	193(211.6)	233	216	145	193	19
8	184(213.6)	149	175(242.0)	300	201(224.0)	127	208(228.8)	34	244	108	218	230
9	184(214.1)	149	190(234.5)	331	184(209.2)	149	210(220.6)	118	212	216	210	370
10	196(215.3)	183	197(263.3)	344	214(219.2)	123	197(282.0)	249	237	139	197	3
11	193(210.9)	121	186(237.1)	288	196(211.6)	183	209(263.4)	77	202	245	212	3
12	197(217.3)	153	182(252.1)	300	202(210.8)	173	195(267.8)	78	211	204	217	246
13	197(217.1)	153	193(225.0)	16	197(211.4)	153	193(215.6)	111	229	199	211	219
14	181(214.7)	183	176(235.5)	297	182(207.2)	169	213(276.0)	4	200	88	240	99
15	192(215.7)	103	180(240.3)	311	193(214.6)	145	204(232.6)	39	233	231	263	76
16	170(221.4)	148	206(242.0)	10	218(231.8)	42	210(273.8)	244	217	177	251	298
17	201(225.9)	110	186(258.3)	330	201(213.4)	110	186(261.2)	235	211	177	466	182
18	185(218.0)	69	187(231.1)	330	216(226.0)	67	188(220.4)	185	205	198	188	67
19	193(214.5)	107	188(232.4)	305	193(207.4)	107	206(228.6)	2	213	17	206	254
20	200(221.9)	148	202(258.1)	79	216(231.4)	69	204(262.6)	29	224	204	427	47
21	182(217.5)	155	202(217.7)	97	205(216.2)	53	203(207.4)	89	195	237	204	6
22	175(215.3)	163	192(222.3)	291	209(217.6)	188	192(205.4)	196	189	60	205	8
23	180(213.4)	176	204(258.5)	51	180(209.4)	176	211(228.0)	17	222	191	266	97
24	179(212.8)	178	192(226.1)	332	179(216.2)	178	209(231.0)	252	207	172	209	0
25	190(219.4)	154	206(236.1)	38	199(218.4)	105	206(219.6)	57	176	50	240	0
26	190(213.6)	154	188(221.7)	312	215(225.0)	57	204(220.0)	27	226	188	221	316
27	191(217.5)	152	195(218.3)	32	199(218.8)	150	195(217.6)	63	220	107	221	263
28	186(214.5)	188	190(241.3)	315	206(215.8)	43	190(220.0)	220	214	119	201	254
29	186(212.7)	188	196(231.9)	293	186(208.8)	188	196(216.0)	198	238	8	196	54
30	178(208.9)	177	201(246.7)	348	206(212.4)	136	217(232.8)	72	215	185	242	21
31	178(213.9)	177	184(259.6)	291	178(210.0)	177	184(219.8)	196	201	247	255	230
32	194(215.9)	201	196(236.5)	27	198(224.4)	115	196(217.4)	68	216	199	205	1
33	189(213.0)	166	204(254.9)	310	203(215.8)	147	204(214.4)	215	214	200	221	247
34	181(212.1)	185	210(233.8)	301	211(218.6)	124	212(261.2)	206	205	203	223	9
35	181(210.6)	185	199(219.3)	81	181(204.4)	185	199(219.8)	14	199	222	199	266
36	185(218.3)	0	194(216.8)	310	190(206.4)	183	194(221.4)	215	234	246	226	326
37	185(218.3)	0	194(212.0)	0	185(229.4)	0	209(218.0)	0	211	0	214	0

Table 11: Min/Avg cut cost and Hamming distance (shown in HD column) for avq_large in 15/5/1 ML-FM runs with 0% and 50% pre-seeded.

References

[1] C. J. Alpert, *Multi-way Graph and Hypergraph Partitioning*, Ph.D. Thesis, UCLA Computer Science Department, 1996.

[2] C. J. Alpert, *personal communication*, June 1996.

[3] C. J. Alpert, L. W. Hagen and A. B. Kahng, "A Hybrid Multilevel/Genetic Approach for Circuit Partitioning", *Proc. ACM/SIGDA Physical Design Workshop*, Reston, Virginia, April 1996, pp. 100-105.

[4] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration, the VLSI Journal*, 19(1-2), pp. 1-81, 1995.

[5] S. Balay, W. Gropp, L. Curfman McInnes, B. Smith, "PETSc 2.0 User's Manual", Argonne National Laboratory, 1995 `http://www.mcs.anl.gov/petsc/petsc.html`

Iter	2 ML-FMs				1 ML-FM			
	0 %		50 %		0 %		50 %	
	Min(Avg)	HD	Min(Avg)	HD	Cut	HD	Cut	HD
1	1794(1802.0)	382	1809(1826.0)	256	1794	452	1843	452
2	1811(1821.0)	328	1820(1837.5)	278	1831	352	1820	429
3	1727(1743.5)	34	1787(1789.0)	32	1727	259	1791	409
⋮	⋮							
23	1826(1835.5)	328	1843(1890.0)	98	1845	390	1843	209
24	1797(1826.0)	178	1829(1839.0)	189	1797	76	1849	593
25	1797(1799.0)	0	1870(1874.5)	0	1801	0	1870	0

Table 12: Min/Avg cut cost and Hamming distance (shown in HD column) for golem3 in 2 and 1 ML-FM runs with 0% and 50% pre-seeded.

[6] R. Barret, M. Berry, T. Chan et al, "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", SIAM 1995, `http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html`

[7] T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms", *Proc. ACM/IEEE Design Automation Conf.*, pp. 775-778, 1989.

[8] C. K. Cheng and E. S. Kuh. "Module Placement Based on Resistive Network Optimization", *IEEE Transactions on Computer-Aided Design*, cad-3, 1984, pp. 218–225.

[9] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Technical Report*, Department of Electrical Engineering, University of Minnesota, Nov. 1995.

[10] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", to appear in *Proc. ACM/IEEE Design Automation Conf.*, 1996.

[11] H. C. Elman, M. P. Chernesky, "Ordering Effects on Relaxation Methods Applied to The Discrete One-Dimensional Convection-Diffusion Equation", SIAM J. Numer. Anal., 30 (5), October 1993, pp.1268-1290.

[12] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/IEEE Design Automation Conf.*, pp. 175-181, 1982.

[13] K. Fukunaga, S. Yamada, H. S. Stone, and T. Kasai, "Placement of Circuit Modules Using a Graph Space Approach", 1983, pp. 465–471.

[14] M. R. Garey, D. S. Johnson, "Computers and Intractability : A Guide to the Theory of NP-completeness", New York : W.H. Freeman, 338p.

[15] W. Hackbush, "Iterative Solution of Large Sparse Systems" Springer Verlag, 1994, 428pp.

[16] K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science*, No.17, 1970. pp. 219-229.

[17] M. Hanan, P. K. Wolff, and B. J. Agule, "A Study of Placement Techniques." *J. Design Automation and Fault-Tolerant Computing*, vol. 2, 1978, pp. 28–61.

[18] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "Quantified Suboptimality of VLSI Layout Heuristics", *Proceedings of Design Automation Conference*, 1995, pp. 216-221.

[19] B. Hendrickson and R. Leland, "A Multilevel Algorithm for Partitioning Graphs", Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[20] K. M. Just, J. M. Kleinhans, and F. M. Johannes, "On the Relative Placement and the Transportation Problem for Standard-Cell Layout", *Proceedings of Design Automation Conference*, 1986, pp. 308-313.

[21] G. Karypis and V. Kumar, "Unstructured Graph Partitioning and Sparse Matrix Ordering", *Technical Report*, Department of Computer Science, University of Minnesota, 1995 (see http://www.cs.umn.edu/~kumar for postscript and code).

[22] J. Kleinhans, G. Sigl, and F. Johannes, "GORDIAN: A New Global Optimization/Rectangle Disection Method for Cell Placement", *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, 1988, pp. 506-509.

[23] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE Transactions on Computer-Aided Design.* 10 (3), March 1991. pp. 356-365.

[24] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. Computers*, 33(5), pp. 438-446, 1984.

[25] T. Lengauer , *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons Ltd.

[26] Numerical Recipes in C: The Art of Scientific Computing, ISBN 0-521-43108-5; Cambridge University Press and Numerical recipes Software 1992 `http://world.std.com/~nr`

[27] R. H. J. M. Otten, "Automatic Floorplan Design", *Proceeding of Design Automation Conference*, 1982, pp. 261-267.

[28] B. M. Riess, K. Doll, and F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 646–651.

[29] B. Riess, G. Ettelt , "SPEED: Fast and Efficient Timing Driven Placement", *ISCAS*, 1995.

[30] G. Sigl, K. Doll, and F. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *28th ACM/IEEE Design Automation Conference*,1991, p.427-432.

[31] W. Swartz and C. Sechen. "Timing Driven Placement for Large Standard Cell Circuits", *32nd ACM/IEEE Design Automation Conference*, 1995, pp. 211–215.

[32] R. S. Tsay and E. Kuh, "A Unified Approach to Circuit Partitioning and Placement", *20th Proceeding of Princeton Conference on Information Sciences and Systems*, 1986, pp. 155-160.

[33] R. S. Tsay, E. Kuh, and C. P. Hsu, "Proud: A Sea-Of-Gate Placement Algorithm", *IEEE Design & Test of Computers*,1988, pp 44-56.

[34] R. S. Tsay, *Partitioning, Placement, amd Routing Algorithms for High Complexity Integrated Circuits*, Ph.D. Thesis, University of California at Berkeley, Computer Science Department, 1989.

[35] R. S. Tsay, E. Kuh, "A Unified Approach to Partitioning and Placement", *IEEE Transactions on Circuits and Systems*, Vol.38, No.5, May 1991. pp. 521-633.

[36] G. J. Wipfler, M. Wiesel, and D. A. Mlynski, "A Combined Force and Cut Algorithm for Hierarchical VLSI Layout, *Proceedings of Design Automation Conference*, 1983, pp. 124-125.