An Interconnect-Centric Design Flow for Nanometer Technologies

JASON CONG, FELLOW, IEEE

Invited Paper

As the integrated circuits (ICs) are scaled into nanometer dimensions and operate in gigahertz frequencies, interconnects have become critical in determining system performance and reliability. This paper presents the ongoing research effort at UCLA in developing an interconnect-centric design flow, including interconnect planning, interconnect synthesis, and interconnect layout, which allows interconnect design and optimization to be properly considered at every level of the design process. Efficient interconnect performance estimation models and tools at various levels are also developed to support such an interconnect-centric design flow.

Keywords*—*Buffer block planning, buffer insertion, circuit partitioning, computer-aided design, delay minimization, design automation, gridless routing, integrated circuits, interconnections, interconnect modeling, interconnect optimization, interconnect planning, noise control, performance-driven routing, physical hierarchy generation, pin assignment, wire sizing.

I. INTRODUCTION

The driving force behind the spectacular advancement of the integrated circuit technology in the past thirty years has been the *exponential scaling* of the transistor feature size, i.e., the minimum dimension of a transistor. It has been following the Moore's Law [1] at the rate of a factor of 0.7 reduction every three years. It is expected that such exponential scaling will continue for at least another 10 to 12 years as projected in the 1997 National Technology Roadmap for Semiconductors (NTRS'97) [2] shown in Table 1.[1] This will lead to over half a billion transistors integrated on a single chip with an operating frequency of 2–3 GHz in the 70-nm technology by the year 2009.

Table 1
Overall Technology Roadmap from NTRS'97 [2]

Technology (nm)	250	180	150	130	100	70
Year	1997	1999	2001	2003	2006	2009
# transistors	11M	21M	40M	76M	200M	520M
Across chip clock (MHz)	750	1200	1400	1600	2000	2500
Area (mm^2)	300	340	385	430	520	620
Wiring Levels	6	6-7	7	7	7-8	8-9

With rapid feature size scaling, the circuit performance is increasingly determined by the interconnects instead of devices. In order to better understand the significance of interconnects in future technology generations, we collected basic interconnect parameters provided in NTRS'97 (shown in boldface in Table 2), and set up a proper 3-D interconnect model to extract various components of interconnect capacitance (shown in the remaining rows in Table 2) using the 3-D field-solver FastCap [4]. We also collected the basic device parameters provided in NTRS'97 (shown in boldface in Table 3) and derived the driver/buffer input capacitance, effective resistance, and intrinsic delay in each technology generation (shown in the remaining rows in Table 3) using HSPICE simulation. These data are used for quantitative analysis of device and interconnect performance in each technology generation.

Table 4 shows the delays of a minimum size transistor, an average length interconnect (1 mm), an unoptimized 2-cm global interconnect, and an optimized 2-cm global interconnect in each technology generation. It shows that although the intrinsic device delay of a minimum-size transistor will decrease from 70 ps in the 250-nm technology down to about 20 ps in the 70-nm technology, the delay of an average interconnect (1-mm metal line) will decrease only from about 60 to 40 ps, while the delays of a 2-cm unoptimized global interconnect (with driver sizing only) will actually increase from

Manuscript received October 2, 2000; revised January 15, 2001. This work was supported in part by Semiconductor Research Corporation under Contract 98-DJ-605, by the National Science Foundation Young Investigator Award MIP-9357582, by the MARCO/Gigascale Research Center, and by a grant from Intel Corporation.

The author is with the Department of Computer Science, University of California, Los Angeles, CA 90095 USA.

Publisher Item Identifier S 0018-9219(01)03201-7.

[1]NTRS'97 has been updated recently and the new version is called the 1999 International Technology Roadmap for Semiconductors (ITRS'99) [3]. The basic trend in ITRS'99 is the same as that in NTRS'97, although technology advancement is accelerated in ITRS'99 in certain areas. All the experimental results reported in this paper are still based on NTRS'97.

Table 2
Interconnect Parameters used in this Paper. The Basic Parameters in the First Six Rows (in Boldface) are Taken from NTRS'97. The Breakdown Capacitance Values in Remaining Rows Under Different Width and Spacing Assumptions are Obtained Using a 3-D Field-Solver (FastCap) [4]. C_a, C_f, and C_x are the Unit-Length Area, Fringing, and Same-Layer Line-to-Line Coupling Capacitance for the Given Width and Spacing Under the Assumption that the Wires are Located Between Two Ground Planes

Technology (nm)		**250**	**180**	**150**	**130**	**100**	**70**
Metal resistivity $\rho(\mu\Omega$ -cm)		**3.3**	**2.2**	**2.2**	**2.2**	**2.2**	**1.8**
Dielectric constant		**3.55**	**2.75**	**2.25**	**1.75**	**1.75**	**1.5**
Min. wire width (nm)		**250**	**180**	**150**	**130**	**100**	**70**
Min. wire spacing (nm)		**340**	**240**	**210**	**170**	**140**	**100**
Metal aspect ratio		**1.8:1**	**1.8:1**	**2.0:1**	**2.1:1**	**2.4:1**	**2.7:1**
Via aspect ratio		**2.2:1**	**2.2:1**	**2.4:1**	**2.5:1**	**2.7:1**	**2.9:1**
2X min. width & spacing	C_a (aF/μm)	29.0	21.2	16.2	12.0	14.4	8.56
	C_f (aF/μm)	41.8	30.2	24.8	18.3	14.1	14.8
	C_x (aF/μm)	71.0	58.3	49.4	42.8	45.3	41.6
5X min. width & spacing	C_a (aF/μ m)	73.5	53.6	40.6	30.0	26.6	19.5
	C_f (aF/μm)	63.5	47.3	38.4	28.5	28.2	23.6
	C_x (aF/μm)	18.3	16.9	15.4	14.8	16.5	16.7

about 2 to 3.5 ns. The optimized 2-cm global interconnect is obtained after the simultaneous driver sizing, buffer insertion, buffer sizing (to be discussed in Section III) using the TRIO package [5]. Although such aggressive optimization reduces the 2-cm global interconnect delay by 2× to 5× across different technology generations, it still does not reverse the trend of a growing gap between device and interconnect performance. It is still about 20× and 30× that of a minimum-size transistor in 100-nm and 70-nm technologies, respectively. Moreover, it also implies that multiple clock cycles are needed for signals to travel over such optimized global interconnects for gigahertz designs in nanometer technologies. For example, even for a moderate clock frequency of 3 GHz in the 70-nm technology generation, 2–3 clock cycles are needed to travel through the 2-cm optimized global interconnect. Note that the interconnect parameters shown in Table 2 from NTRS'97 have already considered the advances in the new interconnect materials, with the use of copper at the 180-nm generation and the use of low dielectric constant materials (the dielectric constant decreases from 3.55 in the 250-nm technology to 1.5 in the 70-nm technology). Although the use of these new interconnect materials is helpful in reducing interconnect delay, they do not provide the ultimate solution to the increasing performance mismatch between devices and interconnects. At best, they improve the interconnect performance by one or two technology generations. Even with these projected improvements, the global interconnects remain the performance bottleneck as shown in Table 4.[2] These results show clearly that the interconnect

[2]The interconnect process parameters provided in NTRS'97 are for a generic metal layer. It is likely that global interconnects will be put on higher metal layers, which have more aggressive reverse scaling. This may help to reduce the global interconnect somewhat. But again, it will not change the conclusion of our analysis.

Table 3
Device Parameters used in this Paper. The Values of Voltage and Transistor On-Current are Taken from NTRS'97. The Remaining Values are Obtained Using HSPICE Simulation. A Buffer is a Pair of Cascaded Inverters with the Size of the Second One Being Five Times that of the First One

Technology (nm)	**250**	**180**	**150**	**130**	**100**	**70**
Vdd (V)	**2.15**	**1.65**	**1.35**	**1.35**	**1.05**	**0.75**
Ion[NMOS/PMOS] ($\mu A/\mu$m)	**600/280**	**600/280**	**600/280**	**600/280**	**600/280**	**600/280**
Buffer input cap. (fF)	0.85	0.60	0.55	0.425	0.35	0.21
Buffer R_d (xKΩ)	3.42	3.72	4.52	4.50	4.78	4.84
Buffer intrinsic delay (ps)	70.5	51.1	48.7	45.8	39.2	21.9

Table 4
Intrinsic Gate Delays[3] and Delay Values for an Average Interconnect (1 mm), a 2-cm Unoptimized Global Interconnect [4] (Both with 2× Minimum Width and 2× Minimum Spacing), and a 2-cm Optimized Global Interconnect After Simultaneous Driver Sizing Buffer Insertion, Buffer Sizing, and Wire Sizing, Using the TRIO Package [5] (to be Presented in Section III) in Different Technology Generations [5] The Last Row Shows the Clock Period Based on the Clock Frequencies Projected in NTRS'97 as Shown in Table 1

Technology (nm)	250	180	150	130	100	70
Device intrinsic delay(ps)	70.5	51.1	48.7	45.8	39.2	21.9
1mm (ps)	59	49	51	44	52	42
2cm un-optimized (ps)	2080	1970	2060	2070	2890	3520
2cm optimized (ps)	890	790	770	700	770	670
Projected clock period (ps)	1333	833	714	625	500	400

delay far exceeds the device delay and is the dominating factor in determining the system performance in current and future technology generations.

Signal reliability due to the coupling noise between interconnects is another serious problem in nanometer designs. In order to limit the increase of interconnect resistance, the wire aspect ratio (height over width) will increase considerably, from its current value of 1.8 : 1 in the 250-nm technology to 2.7 : 1 in the 70-nm technology as predicted in NTRS'97 (shown in Table 2). The increase of wire aspect ratio together with the decrease of line-to-line spacing results in a rapid increase of coupling capacitance. Using FastCap, we computed the percentage of coupling capacitance in terms of the total capacitance in each technology generation and show the results in Fig. 1. We can see that the coupling capacitance contributes to over 70% of the total capacitance under the minimum spacing and over 50% under two times (2×) the minimum spacing in all technology generations!

Fig. 2 shows the peak values of capacitive crosstalk noise in each technology generation for a 1-mm line with 2× the minimum width and spacing to its two neighbors. It reaches over 30% of V_{dd} in the 70-nm generation even for such a moderate length (1 mm). The value of crosstalk noise depends on not only the coupling capacitance of adjacent wires,

[3]Same as the intrinsic buffer delays shown in Table 3.

[4]For both the 1-mm length average interconnect and 2-cm unoptimized global interconnect, the drivers are optimally sized to match the interconnect loads.

[5]The capacitance values used in the optimization are based on the set of 5× minimum width and spacing as shown in Table 2.

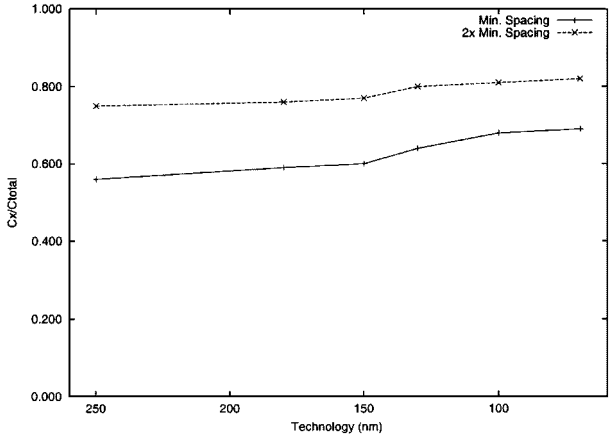


Fig. 1. The percentages of coupling capacitance over the total capacitance under the minimum and 2× minimum spacings in each technology generation.

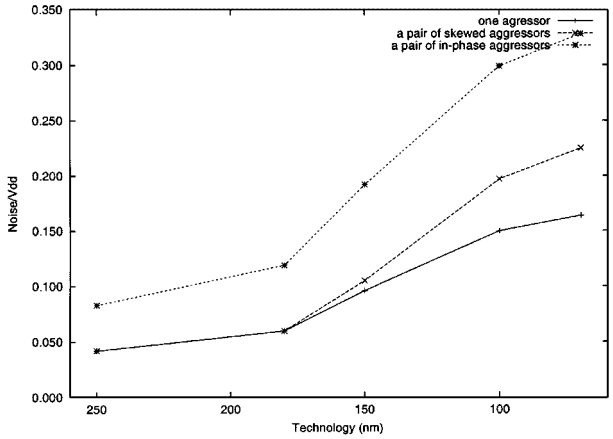


Fig. 2. The ratios of peak capacitive crosstalk noise to V_{dd} for a 1-mm line with 2× the minimum width and spacing to its neighbors under three different temporal relations: i) only one neighbor is switching, ii) both neighbors are switching simultaneously, and iii) both neighbors are switching but one after the other. The rise time of the switching signal is 10% of the projected clock period in NTRS'97.

but also the patterns and relative timing of the signals on neighboring wires. For example, under different switching patterns of neighboring wires, the noise value may differ by a factor of 2× to 3× as shown in Fig. 2. For high-speed circuits, global interconnects may also be subject to inductive noise due to the coupling inductance of the interconnects. Both the capacitive and inductive noises due to the coupling of interconnects present serious threats to signal reliability in nanometer designs if they are not controlled properly.

Given the dominating importance of interconnects in current and future generations of IC designs, we have been developing a new design methodology, named *the interconnect-centric design methodology*. In conventional very large scale integration (VLSI) designs, much emphasis has been given to design and optimization of logic and devices. The interconnection was done by either layout designers or automatic place-and-route tools as an afterthought. In interconnect-centric designs, we suggest that interconnect design and optimization be considered and emphasized

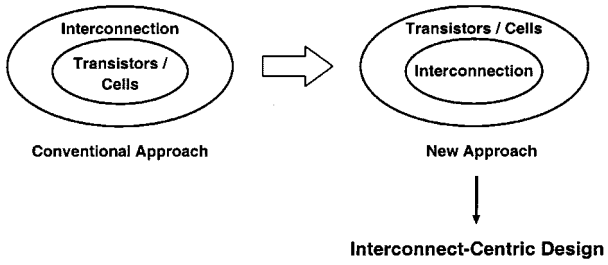


Fig. 3. Proposed paradigm shift for interconnect-centric VLSI design.

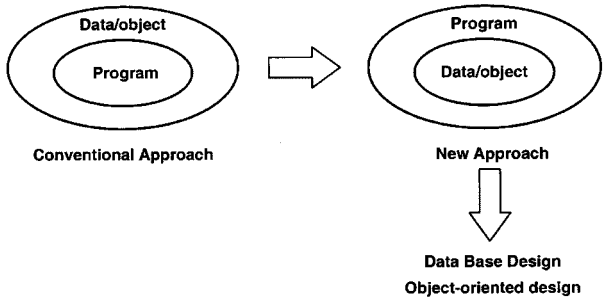


Fig. 4. An analogous methodology change in software design.

throughout the design process (see Fig. 3). Such a paradigm shift is analogous to the one happened in the software design domain of 1970s. In the early days of computer science, much emphasis was placed on algorithm design and optimization, while data organization was considered to be a secondary issue. It was recognized later on, however, that the data complexity is the dominating factor in many applications. This fact gradually led to a data-centric and object-centric software design methodology, including development of the database systems and the recent object-oriented design methodology (see Fig. 4). Although algorithms and data representation/management are integral parts of any software system, the shift in viewpoint from algorithm-centric to data/object-centric designs allows us to effectively manage the design complexity in many large applications. We believe that development of the interconnect-centric design techniques and methodologies will impact the VLSI system design, similar to the way that database design and object-oriented design methodologies have benefited software development.

II. Overview of Our Interconnect-Centric Design Flow

In the past several years, our research group at UCLA has been developing a novel interconnect-centric design flow and methodology that emphasizes interconnect planning and optimization throughout the entire design process. Such a flow goes through the following three major design phases: 1) interconnect planning, which includes physical hierarchy generation, floorplanning/coarse placement with interconnect planning, and interconnect architecture planning; 2) interconnect synthesis, which determines the optimal or near-optimal interconnect topology, wire ordering, buffer locations and sizes, wire width and spacing, etc., to meet the

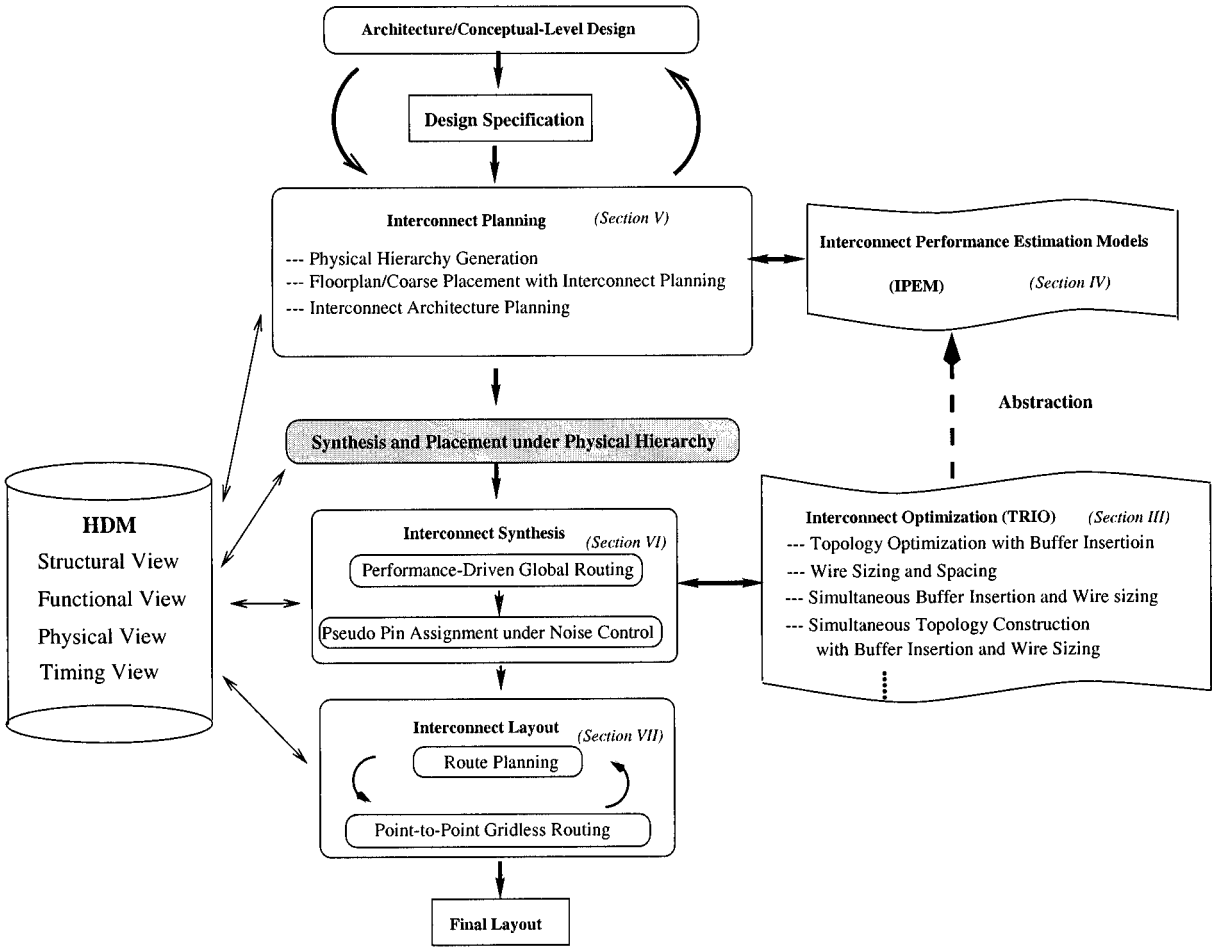


Fig. 5. Overview of our interconnect-centric IC design flow.

performance and signal reliability requirements of all nets under the area and routability constraints; and 3) interconnect layout, which carries out detailed routing to implement the complex width and spacing requirements of all wires using a flexible and efficient multilayer general-area gridless routing system. This paper highlights the key results we have achieved in these areas.

Fig. 5 shows an overview of our proposed interconnect-centric design flow. Each building block in this flow will be discussed in the remainder of this paper. Section III presents a set of interconnect optimization techniques, which form a key building block in our interconnect-centric design flow. Section IV presents a set of very efficient interconnect performance estimation models for optimized interconnects, which are needed for interconnect planning. With these two building blocks, Sections V–VII present the three major steps in our interconnect-centric design flow: interconnect planning, interconnect synthesis, and interconnect layout. Section V presents our results on interconnect planning; this is the centerpiece of the interconnect-centric design flow, including physical hierarchy generation, floorplanning/coarse placement with interconnect planning, and interconnect architecture optimization. After interconnect planning, we have roughly determined the global interconnect structures for achieving the given performance target when it is possible. Otherwise, an early feedback is given to the circuit/system designer to revise the architecture design or/and the performance target. Section VI presents our work on interconnect synthesis, which has successfully incorporated various interconnect optimization techniques in multilayer global routing and pseudo pin assignment for delay and noise control and optimization. After interconnect synthesis, we have basically completed the interconnect designs to meet both the performance and signal reliability (noised related) requirements. Section VII presents our work on developing an efficient multilayer gridless routing system, which is needed to support interconnect layout of optimized interconnects with possibly complex geometries. Finally, Section VIII summarizes our proposed interconnect-centric design flow. We shall revisit Fig. 5 at that time to show how various design steps and optimization techniques presented in this paper interact and interleave with other design steps in the traditional design flow to form a complete design system. Note that the shaded box labeled “synthesis and placement under physical hierarchy” is not discussed in this paper, as we are using off-the-shelf logic synthesis and placement tools for this step. More detailed discussions are given in Section VIII.

III. Interconnect Optimization

Interconnect optimization determines the optimal interconnect structure of each net in terms of interconnect topology, wire width and spacing, buffer locations and sizes, etc., to meet the performance and signal reliability requirements. Interconnect optimization is a key building block in the interconnect-centric design flow. Our group started a systematic study of the interconnect optimization problems since 1991 and has developed a number of efficient optimal or near-optimal algorithms for various interconnect optimization problems, including:

- interconnect topology optimization;
- wire sizing optimization;
- global interconnect sizing and spacing;
- simultaneous driver, buffer, and interconnect sizing;
- simultaneous interconnect topology construction with buffer insertion and/or wire sizing;

and other possible combinations of these optimization techniques. In this section, we first highlight the set of interconnect optimization algorithms developed and used in the UCLA TRIO package (Tree, Repeater, and Interconnect Optimization) [5] for various interconnect optimization problems and briefly mention other related work. Then, we discuss the impact of interconnect optimization on interconnect delay minimization.

A. Interconnect Topology Optimization

We first showed in [6] that when the *resistance ratio*, defined to be the driver effective resistance over the unit wire resistance, is small enough, not only the total wirelength (i.e., the total interconnect capacitance) but also the interconnect topology will impact the interconnect delay. The first step in interconnect topology optimization is to minimize or control the pathlengths from the driver to the timing-critical sinks to reduce the interconnect RC delays. A number of algorithms have been developed by my group and other groups to minimize both the pathlengths and the total wirelength in a routing tree. For example, the *bounded-radius bounded-cost* (BRBC) algorithm [7] bounds the radius (i.e., the maximum pathlength between the driver and a sink) in the routing tree while minimizing its total wirelength. It first constructs a minimum spanning tree (MST), then eliminates the long paths from the source to sinks by adding "short-cuts" into the MST, and finally computes a shortest path tree of the resulting graph. Other algorithms in this class include the AHHK tree construction and the "performance oriented spanning tree" construction, which are discussed in [8] and [9]. A significant progress was the development of the A-tree algorithm [6] that computes a minimal-length shortest-path Steiner tree (called the A-tree) in the Manhattan plane very efficiently using a bottom-up merging heuristic, and reports sizable delay reduction with only a small wirelength overhead compared to the optimal Steiner tree. The A-tree construction method has been extended to signal nets with multiple drivers (as in signal busses) [10]. A graph-version of the A-tree algorithm was also developed [11], which can model routing obstacles and routing congestion in the design. The family of A-tree algorithms has been implemented in the TRIO package.

Further optimization of interconnect topology involves using more accurate delay models during routing tree topology construction. For example, the Elmore delay model was used in [12] and the 2-pole delay model was used in [13] to evaluate which node or edge should be added to the routing tree for delay minimization during iterative tree construction. Incremental moment computation and bottom-up tree construction techniques were nicely combined in the RATS-tree algorithm for topology and wire sizing optimization under higher order moment-based delay models [14]. These results were summarized in [9]. All the topology optimization algorithms discussed so far construct a performance-driven Steiner tree in the Hanan-grid induced by the terminals of the net under optimization. The result in [15] shows that under the Elmore delay model, it is sometimes beneficial to use a Steiner tree that is not completely on the Hanan-grid for further delay minimization.

B. Wire Sizing Optimization

We showed in [6] and [16] that when wire resistance becomes significant, as in deep submicrometer or nanometer designs, proper wire sizing can effectively reduce the interconnect delay. Assuming each wire has a set of discrete wire widths, an optimal wire sizing algorithm was developed in [6] and [16] for a single-source RC interconnect tree to minimize the sum of weighted delays from the source to timing-critical sinks under the Elmore delay model. The study showed that an optimal wire sizing solution satisfies the monotone property, the separability, and the dominance property. In particular, the dominance property is important for efficient computation of an optimal wire sizing solution. Given two wire sizing solutions $\mathcal{W}$ and $\mathcal{W}'$, we say that $\mathcal{W}$ *dominates* $\mathcal{W}'$ if $w_e \geq w'_e$ for every segment e (where w_e is the width of segment e in $\mathcal{W}$). Given a wire sizing solution $\mathcal{W}$ for the routing tree, and any segment e in the tree, a *local refinement* on e is defined to be the operation to optimize the width of e while keeping the wire width assignment of $\mathcal{W}$ on other segments unchanged. Then, the dominance property can be stated as follows [17]:

Dominance Property: Suppose that $\mathcal{W}^*$ is an optimal wire sizing solution. If a wire sizing solution $\mathcal{W}$ dominates $\mathcal{W}^*$, then any *local refinement* of $\mathcal{W}$ still dominates $\mathcal{W}^*$. Similarly, if $\mathcal{W}$ is dominated by $\mathcal{W}^*$, then any *local refinement* of $\mathcal{W}$ is still dominated by $\mathcal{W}^*$.

Based on the dominance property, the lower (or upper) bounds of the optimal wire widths can be computed efficiently by iterative local refinement, starting from a minimum-width solution (or maximum-width solution for computing upper bounds). It was shown in [17] that the lower and upper bounds usually meet, which leads to an optimal wire sizing solution. Otherwise, for those segments that the lower and upper bounds do not meet, a simple enumeration-based approach or a more clever dynamic programming-based method can be used to compute the optimal widths of those segments within their lower and upper bounds. This method is very efficient, capable of handling

large interconnect structures, and leads to substantial delay reduction. It has been extended to optimize the routing trees with multiple drivers [18], routing trees without *a priori* segmentation of long wires [18], and to meet the target delays using Lagrangian relaxation [19]. The reader may refer to [9] for a more detailed summary.

An alternative approach to wire sizing optimization computes an optimal wire sizing solution using bottom-up merging and top-down selection [20] in a way very similar to the buffer insertion algorithm to be presented in the Section III-D. At each node v, a set of irredundant wire sizing solutions of the subtree rooted at v is generated by merging and pruning the irredundant wire sizing solutions of the subtrees rooted at the child nodes of v. Eventually, a set of irredundant wire sizing solutions is formed at the driver for the entire routing tree, and an optimal wire sizing solution is chosen by a top-down selection process. The approach has the advantages that it can handle different signal required times at sinks directly without going through iterative weighting by the local refinement based approaches. It can also be easily extended to be combined with routing tree construction and buffer insertion as shown in Section III-E. Both the local refinement-based approach and the dynamic programming-based approach have been implemented in the TRIO package.

Further studies on wire sizing optimization include using more accurate delay models, such as higher order RC delay models [21] and lossy transmission-line models [22], and optimal wire shaping under the assumption that continuous wire sizing is allowed to each wire segment [23], [24]. These results are discussed in more details in [9]. The impact of wire sizing is discussed in Section III-F.

C. Global Wire Sizing and Spacing

The coupling capacitance between adjacent wires has become a significant portion of the total wire capacitance in nanometer designs. All the wire-sizing algorithms presented in the preceding subsection either ignore the coupling capacitance or assume a fixed coupling capacitance and lump it into the fringing capacitance. The assumption of a fixed coupling capacitance during wire sizing is not realistic if we maintain a fixed pitch spacing between wires. In this case, if the width of one wire is changed, its spacings to adjacent wires will also change, usually resulting in different coupling capacitance. The *global interconnect sizing and spacing (GISS)* problem optimizes the widths and spacings for multiple nets simultaneously with consideration of coupling capacitance for delay minimization. We developed two methods (in [25] and [26]) and implemented them in the TRIO package. Both methods have developed the theory to establish some kinds of dominance property, similar to the one presented in the preceding subsection for wire sizing, which enable the efficient iterative computation of the lower and upper bounds of the optimal GISS solution (for multiple nets). These bounds usually meet for most of wire segments, which give the optimal widths and spacings for these segments. Bounded enumeration or dynamic programming can be used on wire segments whose lower and upper bounds do not meet in order to search for an optimal solution (see [25] and [26] for details). Both methods can handle fairly general capacitance models, such as a table-lookup-based capacitance model, and do not rely on closed-form formula for capacitance computation. Substantial delay improvements were reported in [25] and [26] when compared to the net-by-net wire sizing approaches.

D. Buffer Insertion

For long interconnects, wire sizing (and spacing) alone is not sufficient to limit the quadratic growth of the interconnect delay with respect to its length. In this case, *buffer insertion* (also called *repeater insertion*) is widely used to tradeoff the active device area for reduction of interconnect delays. With optimal buffer insertion, the growth of interconnect delay becomes linear to its wirelength.

A polynomial-time optimal algorithm was presented in [27] to find the optimal buffer placement and sizing for RC trees under the Elmore delay model. The algorithm assumes that the possible buffer positions (called legal positions), possible buffer sizes, and the required times at sinks are given and uses a dynamic programming approach to maximize the required arrival time at the source. The algorithm works in two phases: a bottom-up synthesis phase for computing possible buffer assignment solutions at each node and a top-down selection phase for generating the optimal buffer insertion solution. In the bottom-up synthesis phase, at each legal position i for buffer insertion, a set of possible buffer assignments, called *options*, in the subtree T_i rooted at i is computed. For a node k who is the parent of two subtrees T_i and T_j, the list of options for T_k is generated from the option lists for T_i and T_j based on a merging rule and a pruning rule, so that the number of options for T_k is no more than the sum of the numbers of options for T_i and T_j plus the number of possible buffer assignments on the edge coming to k. As a result, if the total number of legal positions is N and there is one type of buffer, the total number of options at the root of the entire routing tree is no larger than $N+1$ even though the number of possible buffer assignments is 2^N. In the top-down selection phase, the optimal option which maximizes the required arrival time at the source is first selected. Then, a top-down backtracing procedure is carried out to select the buffer assignment solution that led to the optimal option at the source. This algorithm has been implemented in the TRIO package. The impact of buffer insertion is discussed in Section III-F.

E. Simultaneous Device and Interconnect Optimization

The most effective approach to interconnect performance optimization is to consider the interaction between devices and interconnects, and optimize both of them at the same time. Two approaches are discussed in this subsection.

1) Simultaneous Device and Wire Sizing: We first developed a solution to the simultaneous driver and wire sizing (SDWS) problem in [28] and later generalized it to solve the simultaneous buffer and wire sizing (SBWS) problem for a buffered routing tree [29]. In both cases, the switch-resistor model is used for the driver and the Elmore delay model is

used for the interconnects modeled as RC trees. The objective function is to minimize the sum of weighted delays from the first stage of the cascaded drivers through the buffered routing tree to timing-critical sinks. It was shown that the dominance property still holds for the SDWS and SBWS problems, and the local refinement operation, as used for wire sizing, can be used iteratively to compute tight lower and upper bounds of the optimal widths of the driver, buffers, and wires efficiently. Again, the lower and upper bounds often meet, which leads to an optimal solution. Dynamic programming or bounded enumeration can be used to compute the optimal solution within the lower and upper bounds when they do not meet. This approach has demonstrated substantial reduction on both delay and power compared to manual designs when applied to large buffered clock trees.

It was recently shown in [26] and [30] that the dominance property holds for a large class of objective functions called *general CH-posynomials*. Based on this general result, we developed an algorithm capable of performing *simultaneous transistor and wire sizing* (STWS) efficiently for a general netlist (not limited to buffered trees). A significant advantage of the CH-posynomial formulation is that it can handle more accurate transistor models, including both simple analytical models or more accurate table-lookup-based models obtained from detailed simulation. In this case, the equivalent resistance of a transistor is no longer a fixed constant. It considers the effect of the input waveform slope and the output load, which leads to better optimization results. Both SDWS/SBWS and STWS algorithms have been implemented in the TRIO package.

Other studies on simultaneous device and wire sizing include a polynomial-time optimal solution to the SBWS problem under the continuous buffer and wire sizing assumptions [31], a method of using higher order RC delay models for the interconnect by either matching to the target moments [32] and a method of using a q-pole transfer function [33] for sensitivity analysis. A related problem to the SBWS problem is simultaneous buffer insertion with wire sizing, which can be solved optimally by a convex quadratic programming approach [34]. The reader may refer to [9] for a more detailed survey.

2) Simultaneous Topology Construction with Buffer and Wire Sizing: The *wire-sized buffered A-tree* (WBA-tree) algorithm was proposed [35] for simultaneous routing tree topology construction, buffer insertion, and wire sizing. It naturally combines the A-tree construction algorithm [6] and the simultaneous buffer insertion and wire sizing algorithm [20], as both use bottom-up construction techniques. The WBA-tree algorithm includes a bottom-up synthesis phase and a top-down selection phase. During the bottom-up synthesis phase, it iteratively selects two subtrees for merging with consideration of both minimization of wirelength and maximization of the estimated arrival time at the source. As a result, it is able to achieve both *critical path isolation* and a *balanced load decomposition*, as often used for fanout optimization in logic synthesis. The top-down selection phase selects the best buffered A-tree topology with wire sizing recursively starting from the source. It was shown that the WBA-tree algorithm produces solution with smaller delay than the two-step approach of A-tree construction followed by optimal buffer insertion and wire sizing. The WBA-tree algorithm has been implemented in the TRIO package. It has also been extended recently to explore multiple interconnect topologies at each subtree and use higher order RLC delay models based on efficient incremental moment computation in partially constructed routing trees [14].

Other methods, such as iterative routing tree construction with wire sizing [36], [37] and combining P-tree topology with buffer insertion and wire sizing [38] have also been proposed. These algorithms are summarized in [9]. We would like to point out that it was shown in [39] and [40] that, with sufficient buffer insertion, the wire sizing solution can be considerably simplified so that it uses only a few wire widths.

F. Impact of Optimal Interconnect Optimization

The TRIO package has integrated all of the interconnect optimization algorithms highlighted in preceding subsections and was used to evaluate the impact of various interconnect optimization techniques. Two types of device models are considered in the TRIO package—a simple switch-level RC model, and a table-based device delay model that models the device delay as a function of input waveform slope, device size, and output load. Two types of interconnect capacitance models are considered in the TRIO package—a simple model that assumes constant unit area and unit fringing capacitance, and a table-based interconnect capacitance model that considers area, fringe, and coupling capacitance as a function of wire width and spacing [41]. Most of the algorithms in TRIO use the Elmore delay to guide the optimization process. Some of them also use high-order moments-based delay models. The optimization engines of the algorithms in TRIO use either bottom-up dynamic programming or efficient iterative local refinement based on the dominance property. Both approaches produce optimal or near-optimal results with polynomial time complexity in most cases.

Our experimental results show that interconnect optimization can reduce the interconnect delay significantly. Fig. 6 shows the impact of various interconnect optimizations on a 2-cm global interconnect in each technology generation as projected in NTRS'97 [2]. The three delay curves shown in the figure correspond to a 2-cm unoptimized interconnect with driver sizing only to match the load (labeled as DS), a 2-cm interconnect with optimal buffer insertion and sizing (labeled as BIS), and a 2-cm interconnect with optimal buffer insertion, buffer sizing, and wire sizing (labeled as BISWS), all computed by the TRIO package. As we move from the 250-nm technology to the 70-nm technology, with driver sizing only, the 2-cm global interconnect delay increases roughly from 2 to 3.5 ns. With optimal buffer insertion and sizing, the 2-cm global interconnect delay is controlled to be under 2 ns, decreasing slightly from 1.8 ns in the 250-nm technology to 1.5 ns in the 70-nm technology. When optimal wire sizing is also applied, the delay is further reduced, from 900 ps in the 250-nm technology to about

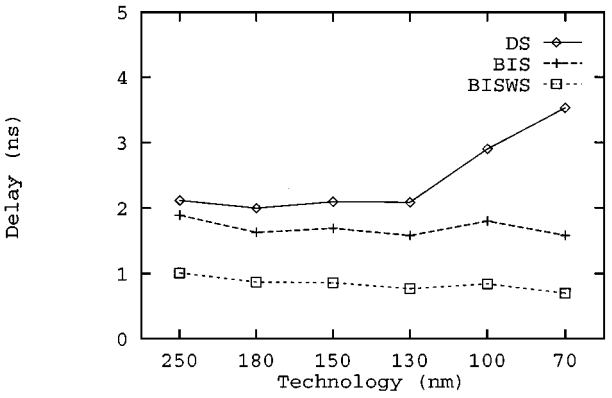


Fig. 6. Impact of optimal interconnect optimization for a 2-cm global interconnect in each technology generation in NTRS'97. The maximum buffer during the optimization is limited to 40× minimum feature size, and the maximum wire width is set to be 10× minimum wire width.

700 ps in the 70-nm technology. As can be seen from the figure, a factor of up to 5× can be achieved with proper interconnect optimization. All the experiments are based on the interconnect parameters and device parameters derived from NTRS'97 as presented in Tables 2 and 3. Currently, the TRIO package is being extended for optimizing multiple physically related and temporal-related interconnect structures for both delay and noise optimization in nanometer designs. The TRIO package is available for download from the World Wide Web [5].

IV. Interconnect Performance Estimation

Given the fact that interconnect optimization may lead to a factor of 5× reduction on global interconnect delays as shown in the preceding section, it is important to fully consider the impact of interconnect optimization during design planning. However, a brute-force integration by running existing interconnect optimization algorithms directly during the synthesis and design planning stages will not be practical for the following reasons:

- Inefficiency: Although most of the interconnect optimization algorithms discussed in the preceding section have polynomial time complexity and are efficient to use during layout synthesis (For example, TRIO can optimize roughly 1 to 100 nets per second depending on the optimization algorithm being used), they are not efficient enough to be used *repeatedly* during interconnect planning where one would like to explore tens of thousands of floorplan configurations. For each configuration, the performance of tens of thousands of global and semi-global interconnects needs to be evaluated very quickly.
- Lack of abstraction: To make use of those optimization programs, a lot of detailed information is needed, such as the granularity of wire segmentation, number of wire widths, and buffer sizes, etc. However, such information is usually not available during design planning.

Given these difficulties, existing design planning tools simply use wirelength-based interconnect delay models and ignore the impact of various possible interconnect optimization operations. This may lead to very inaccurate results. To overcome this problem, we have developed a set of fast and accurate *interconnect performance estimation models* (IPEMs) with consideration of various optimization techniques, including optimal wire sizing (OWS), simultaneous driver and wire sizing (SDWS), and simultaneous buffer insertion, buffer sizing, and wire sizing (BISWS) [42], [39]. These IPEMs are very efficient (constant run time in practice), and provide high-level abstraction. In addition, our IPEMs provide explicit relations between the interconnect performance and layout design parameters under various kinds of optimization; this helps to make design decisions at high levels. These models have been tested on a wide range of parameters and exhibit about 90% accuracy on average compared with those running complex optimization algorithms in TRIO directly (in terms of the delay measured by HSPICE simulations).

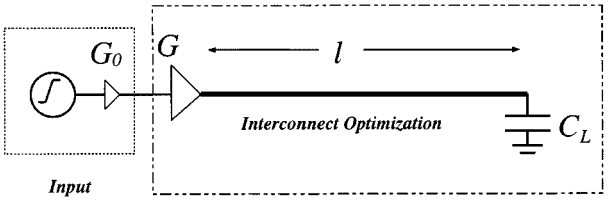


Fig. 7. IPEM problem formulation.

The interconnect performance estimation problem can be formulated as follows. Given an interconnect wire of length l driven by a gate G and with loading capacitance C_L as shown in Fig. 7, we assume that G's input waveform is generated by a nominal gate G_0 connected with a ramp voltage input. The delay to be minimized is the overall delay from the input of G_0 to the load C_L, while the delay to be estimated is the stage delay from the input of G to C_L, denoted as $T(G, l, C_L)$. The input stage delay is included so that it acts as a constraint to avoid oversizing of G during the interconnect optimization. Our goal is to develop simple closed-form formula and/or procedures to efficiently estimate $T(G, l, C_L)$ with consideration of various interconnect optimization techniques such as OWS, SDWS, and BISWS. In the rest of this section, we highlight two commonly used IPEMs, one for optimal wire sizing (OWS), and the other for simultaneous buffer insertion and wire sizing (BIWS).

A. IPEM Under OWS

We showed in [42] that the delay a wire of length l driven by a driver of effective resistance R_d with loading capacitance C_L under OWS can be estimated using the following formula:

$$T_{\text{ows}}(R_d, l, C_L) = \Big(\alpha_1 l/W^2(\alpha_2 l) + 2\alpha_1 l/W(\alpha_2 l) + R_d c_f + \sqrt{R_d r c_a c_f l}\Big) \cdot l \qquad (1)$$

where

$$\alpha_1 = \frac{1}{4} r c_a, \qquad \alpha_2 = \frac{1}{2}\sqrt{\frac{r c_a}{R_d C_L}}$$

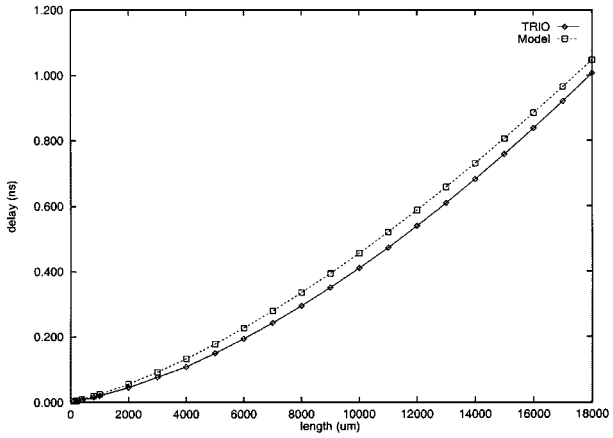


Fig. 8. Comparison of our delay estimation model with running TRIO for OWS under the 0.18-μm technology, with R_d and C_L from a 100× min gate. TRIO uses wire width set $\{W_{\min}, 2W_{\min}, \ldots, 20W_{\min}\}$ and 10-μm-long segments.

r is the sheet resistance, c_a and c_f are the unit area and fringing capacitance coefficients, respectively, and $W(x)$ is Lambert's W function [43] defined as the value of w that satisfies $we^w = x$. Since the value of the W function can be obtained easily through either table-lookup or simple numerical computation, the formula shown in (1) can be computed in constant time, which is at least an order of 10 000 times faster than running the best available OWS algorithm directly. Fig. 8 shows the comparison of the results obtained by our delay estimation model versus those by running the TRIO package under OWS optimization. It shows that our model is highly accurate (about 90% accuracy on average).

In fact, the area of the OWS solution can also be estimated accurately using the following formula

$$A_{\text{ows}}(R_d, l, C_L) = \sqrt{\frac{r(c_f l + 2C_L)}{2R_d c_a}} \cdot l. \tag{2}$$

Wiring area estimation is useful for routing congestion analysis and estimation of power dissipation on interconnects.

B. IPEM Under BIWS

As shown in Section III-F, simultaneous buffer insertion and wire sizing (BIWS) is most effective in reducing long interconnect delays. Such optimization is usually based on the dynamic programming technique and is not efficient enough for fast delay estimation. In this section, we will first introduce the concept of critical length for buffer insertion under OWS and give an analytical formula for it. Then, we shall present the IPEM under BIWS using the concept of critical length.

1) Critical Length for BIWS: We define the *critical length* under BIWS to be the longest length that a wire can run without benefiting from buffer insertion. Let $T_{\text{1buf}}(\alpha, R_d, l, C_L)$ denote the delay by inserting a buffer b at the position of αl from the source ($0 \le \alpha \le 1$). Then

$$\begin{aligned} &T_{\text{1buf}}(\alpha, R_d, l, C_L) \\ &= T_{\text{ows}}(R_d, \alpha l, C_b) + T_b + T_{\text{ows}}(R_b, (1-\alpha)l, C_L) \end{aligned} \tag{3}$$

Table 5
Critical Length $l_c(b)$ (in mm) for BIWS in Each Technology Generation for Some Typical Buffer Sizes from 2× to 100× of a Minimum Size Buffer. The Corresponding Values Under Uniform min Wire Width Based on [44] are Shown in the Last Row for Comparison

Tech. (μm)	0.25	0.18	0.15	0.13	0.10	0.07
2×	4.12	3.80	3.97	3.61	2.92	2.08
10×	6.40	5.81	6.01	5.51	4.45	3.30
20×	7.47	6.83	7.04	6.39	5.30	3.91
40×	8.65	7.92	8.14	7.43	6.35	4.49
100×	9.98	9.10	9.30	8.57	7.13	5.21
[44]	2.52	2.23	2.14	1.94	1.50	1.43

is the delay after inserting the buffer and applying OWS to the two resulting wires separated by the buffer b with intrinsic delay of T_b, input capacitance of C_b, and output resistance of R_b.

We can find the α that minimizes $T_{\text{1buf}}(\alpha, R_d, l, C_L)$ by solving the root of $dT_{\text{1buf}}/d\alpha = 0$ under $0 \le \alpha \le 1$ and denote it as $\alpha^*(l)$. Then, it is beneficial to insert a buffer b if and only if the resulting delay is smaller than the delay by OWS only, i.e.,

$$T_{\text{1buf}}(\alpha^*(l), R_d, l, C_L) < T_{\text{ows}}(R_d, l, C_L). \tag{4}$$

We define the *critical length* for inserting buffer b to be the minimum l that satisfies (4) and denote it as $l_{\text{crit}}(b, R_d, C_L)$.

Intuitively, when the wirelength l is small, optimal wire sizing will achieve the best delay, whereas when the interconnect is long enough, the buffer insertion becomes beneficial. Thus, the root of l^* for the following equation:

$$f(l) = T_{\text{1buf}}(\alpha^*(l), R_d, l, C_L) - T_{\text{ows}}(R_d, l, C_L) = 0 \tag{5}$$

gives the critical length for buffer insertion, i.e., $l_{\text{crit}}(b, R_d, C_L)$. We can use a fast two-level binary search to compute the root l^* for (5). Let ϵ_{l0} and ϵ_l be the initial range and the error tolerance for l^*, respectively, and $\epsilon_{\alpha 0}$ and ϵ_α be the initial range and the error tolerance for α^*, respectively. Then, the root can be computed in $\log_2(\epsilon_{l0}/\epsilon_l)$ iterations of l. For each l, we need another binary search for $\alpha^*(l)$, which takes $\log_2(\epsilon_{\alpha 0}/\epsilon_\alpha)$ steps. In practice, $\epsilon_{l0} = 2$ cm, $\epsilon_l = 10$ μm, $\epsilon_{\alpha 0} = 1$, and $\epsilon_\alpha = 0.01$ are usually sufficient for our delay estimation purpose, which leads to at most $\log_2 2000 \times \log_2 100 = 77$ steps for computing $l_{\text{crit}}(b, R_d, C_L)$. So, in practice, $l_{\text{crit}}(b, R_d, C_L)$ can be computed in constant time.

For simplicity, let us denote $l_{\text{crit}}(b, R_b, C_b)$ as $l_c(b)$. Table 5 shows the critical lengths $l_c(b)$ computed by our method using some typical buffer sizes in each technology generation based on the interconnect and device parameters presented in Tables 2 and 3. We also compared these values with the critical lengths computed using the formula in [44] that does not consider optimal wire sizing (in their case, the critical length is independent of the buffer size). It is

Table 6
Logic Volumes ($\times 10^6$) in Terms of the Number of 2-Input Minimum-Size NAND Gates (Area Estimated Based on NTRS'97) that Can be Packed in the Square Area of $(1/2)\,l_c(b) \times (1/2)\,l_c(b)$ for Different Buffer Sizes in Each Technology Generation

Tech. (μm)	0.25	0.18	0.15	0.13	0.10	0.07
2-NAND (μm^2)	7.80	4.04	3.00	2.18	1.28	0.64
2×	0.55	0.89	1.31	1.49	1.66	1.69
10×	1.31	2.09	3.01	3.48	3.87	4.25
20×	1.79	2.88	4.13	4.68	5.48	5.97
40×	2.40	3.88	5.52	6.33	7.87	7.88
100×	3.19	5.12	7.21	8.42	9.93	10.6

not surprising to see that, with OWS, the critical lengths computed by our method are longer than those from [44] without OWS.

We would like to point out that, although $l_c(b)$ decreases as the feature size scales down, the number of logic cells that can be reached within $l_c(b)$ is actually increasing. We define the *logic volume* covered by $l_c(b)$ to be the number of two-input minimum-size NAND gates that can be packed in the region spanned by $(1/2)\,l_c(b) \times (1/2)\,l_c(b)$. Table 6 shows that the logic volume actually increases due to the scaling down of devices. This implies that more and more devices can be packed into a region without buffer insertion.

2) IPEM Under BIWS: Using the concept of critical length, we showed in [42] that in an optimal BIWS solution, the distances between adjacent buffers are the same and can be approximated by $l_{\text{crit}}(b, R_b, C_b)$. Therefore, the delay of an interconnect of length l can be approximated by the following simple linear model with respect to l:

$$T_{\text{biws}} = \tau_{\text{biws}} \cdot l + t_g \tag{6}$$

where t_g is the gate delay and τ_{biws} is given by the delay of critical length l_c under OWS (estimated by IPEM in Section IV-A) divided by length l_c. That is,

$$\tau_{\text{biws}} = t_g/l_c + \alpha_1 l_c/W^2(\alpha_2 l_c) + 2\alpha_1 l_c/W(\alpha_2 l_c) + R_b c_f + \sqrt{R_b r c_a c_f l_c}. \tag{7}$$

In practice, $l_c(b) = l_{\text{crit}}(b, R_b, C_b)$ can be computed in constant time (in fact, they can be precomputed for each technology and each buffer size and stored in a look-up table, if necessary), so can (7). Given these values, the IPEM under BIWS shown in (6) can clearly be computed in constant time. Fig. 9 shows the comparison of the delay values predicted by the IPEM under BIWS with those obtained after actual optimization using TRIO. Again, a close match is observed.

IPEMs for other interconnect optimization procedures, such as simultaneous driver and wire sizing (SDWS), and buffer insertion, buffer sizing, and wire sizing (BISWS) have also been developed and discussed in [42]. The interconnect performance estimation models discussed in this section can be used in a wide spectrum of applications, such as the following.

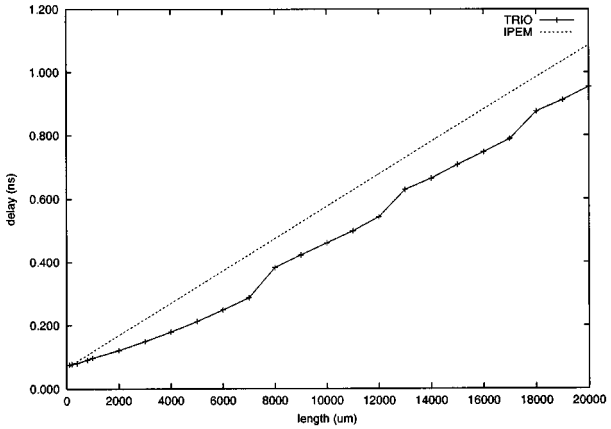


Fig. 9. Comparison of IPEM under BIWS with actual BIWS optimization using TRIO under the 0.18-μm technology. G_0 and C_L are from 10× min. Buffer size is 20× min.

- RTL and physical level floorplan: During the sizing and placement of functional blocks, our models can be used to accurately predict the impact on the performance of global interconnects. Such applications will be shown in Sections V-A and V-B.
- Placement-driven synthesis and mapping: A companion placement may be kept during synthesis and technology mapping as suggested in [45]. For every logic synthesis operation, the companion placement will be updated. Once the cell positions are known, our IPEMs can be used to accurately predict interconnect delays for the synthesis engine.
- Interconnect process parameter optimization: Interconnect parameters (e.g., metal aspect ratio, minimum spacing, etc.) may be tuned to optimize the delays predicted by our models for global, average, and local interconnects under certain wirelength distributions. One such application will be shown in Section V-C.

The UCLA IPEM package includes a set of library routines that implement the IPEMs for various interconnection optimization procedures. It can be downloaded from the World Wide Web at http://cadlab.cs.ucla.edu/software_release/ipem/htdocs/. The use of IPEM is demonstrated in interconnect planning as shown in the next section.

V. Interconnect Planning

Interconnect planning is the first step and also the centerpiece of our interconnect-centric design flow. It is applied very early on in the design process and has tremendous impact on the final result. In this paper, we discuss interconnect planning after interconnect optimization and interconnect performance estimation because interconnect planning makes uses of various interconnect performance estimation models to consider the impact of interconnect optimization during the planning process.

We further divide the interconnect planning process into three steps: physical hierarchy generation, floorplanning/coarse placement with interconnect planning, and interconnect architecture planning. These are defined in the following paragraphs.

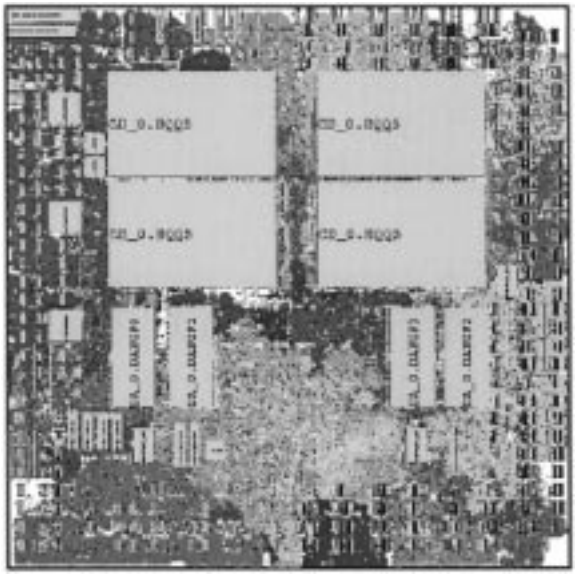


Fig. 10. An example of logic hierarchy in the final layout (Courtesy of IBM). It is a large ASIC design with over 600 000 placeable objects, designed using IBM's SA27E technology (a 0.18-μm technology with $L_{\text{eff}} = 0.11\,\mu$m and using copper wires).

- Physical hierarchy generation: Designs in the nanometer technologies are inevitably hierarchical given their high complexity. However, the HDL description provided by the architecture and/or circuit designers usually follows the *logical hierarchy* of the design which reflects the logic dependency and relationship of various functions and components in the design. Such logical hierarchy may not map well to a two-dimensional layout solution as it is usually conceived with little or no consideration of the layout information. This is further evident from the suboptimal results produced by many existing hierarchical design tools which use the logic hierarchy for floorplanning and recursive synthesis, placement, and routing. Their results can be considerably worse than those by (good) flat design tools (when the design complexity is still tractable). Fig. 10 shows an example of the logic hierarchy in the final layout (obtained by optimizing directly on the flat design). Modules in the same block in the logic hierarchy have the same gray shading in the layout. As can be seen, the logic hierarchy does not map directly into the physical hierarchy. This suggests that enforcing floorplanning or placement algorithms to follow the logic hierarchy boundary can be harmful to the final layout. Therefore, the first step of our interconnect planning process is to generate a good *physical hierarchy* that is most suitable for being embedded on a two-dimensional silicon surface for performance optimization. Such physical hierarchy generation in fact defines the global, semi-global, and local interconnects (based on their levels in the physical hierarchy) and has significant impact on the final design quality. In Section V-A, we present our recent work on multilevel, multiway, performance-driven partitioning with retiming as a possible approach to generating a good physical hierarchy. Retiming is considered during partitioning so that flip-flops can be repositioned onto the global interconnects to hide some global interconnect latency.
- Floorplanning/coarse placement with interconnect planning: After the physical hierarchy is generated, the second step is floorplanning with interconnect planning, which is also called *physical-level interconnect planning*. It interacts closely with the interconnect synthesis tools (to be presented in Section VI) and plans for the best interconnect topology, wire ordering, wire width and spacing, layer assignment, etc., for all global and semi-global interconnects to meet the required performance. For example, it is estimated that there will be a large number of buffers to be inserted for high-performance designs in future technology generations (close to 800 000 in 70-nm technology [46]). If these buffers are distributed over the entire chip in an unstructured way, it will definitely complicate the layout design and verification. Section V-B presents a method to automatically plan for buffer blocks during floorplan to achieve performance, area, and routability optimization.
- Interconnect architecture planning: Due to the advance in VLSI fabrication technology, such as the use of chemical–mechanical polishing (CMP) for global and local planarization of insulator and metal levels, the design rules are no longer completely dictated by the manufacturing capability and leave large room for optimization. The goal of *interconnect architecture planning* is to take advantage of the degree of freedom in the process technology and determine various interconnect parameters for overall system-level performance, reliability, and power optimization, subject to the manufacturing constraints. These parameters include the number of routing layers, the thickness of each interconnect and isolation layer, the metal resistivity and dielectric constant of each layer (assuming different material/process may be used for different layers for performance, yield, and cost considerations), the nominal width and spacing in each layer, vertical interconnection schemes (e.g., via dimensions and structures), and so on. Such interconnect architecture planning should consider a given design characterization (specified in terms of the target clock rate, interconnect distribution, depth of the logic network, etc.) obtained after physical hierarchy generation and floorplanning with interconnect planning. In some cases, such optimization requires adjustments in the fabrication process, which is more suitable and economical for high-volume designs (such as microprocessor designs) or a class of designs with similar design characterizations. We present in Section V-C our work on wire-width planning as an example of interconnect architecture planning, whose objective is to predetermine a small number of common wire widths in each layer so that they can be used for optimizing interconnects of a wide range of lengths in that layer.

The next three subsections illustrate our progress on physical hierarchy generation, floorplanning with interconnect planning, and interconnect architecture planning. We would like to emphasize that although significant progress has been made on interconnect planning as reported in this section, we are still actively working in this area to gain deeper understanding and search for better solutions.

A. Performance-Driven Partitioning with Retiming

As we explained at the beginning of this section, the first, and probably the most important, step of the interconnect planning process is to transform the logic hierarchy implied in the design specification into a good physical hierarchy so that it is most suitable for being embedded onto a two-dimensional silicon surface for performance optimization. We believe that this step can be achieved using (possibly recursive) partitioning and floorplanning/coarse placement with careful consideration of the impact on interconnect performance. Traditionally, partitioning is viewed and used as a means to enable the divide-and-conquer methodology to tackle the design complexity (as used, for example, in the min-cut-based placement approach). In our interconnect-centric design flow, however, we view top-down partitioning as a step that *defines* the interconnects—the connections between different blocks resulted from top-level partitioning become global interconnects, and the connections within the same block after several steps of partitioning become local interconnects. After applying partitioning recursively (sometimes together with coarse placement), we can define a hierarchy of interconnects, which in turn defines the *physical hierarchy* of the given design. In order to achieve this objective, we have developed a *performance-driven* partitioning algorithm with consideration of retiming. Our algorithm, named HPM, is different from the conventional partitioning algorithms in two ways.

- The HPM algorithm is targeted for performance optimization. Most conventional partitioning algorithms consider only cutsize minimization. Although this tends to minimize the total number of global interconnects, it does not consider the impact of the partitioning result on the overall circuit performance. For example, it is not desirable to have multiple global interconnects in a timing-critical path (recall that Table 4 shows that the delay of 2-cm global interconnect is about 15× larger than that of a 1-mm local interconnect). Yet the conventional partitioning algorithms make no effort to avoid such configuration.
- The HPM algorithm considers retiming during partitioning to hide (some) global interconnect latency. The benefit of considering retiming during partitioning can be illustrated by the simple motivational example shown in Fig. 11. The two partitioning solutions (a) and (b) both have the same cutsize of 1 and delay of 4, assuming that each node delay is 1, the intrablock connection delay is 0, and the interblock connection delay is 2. Let us apply optimal retiming to both solutions.

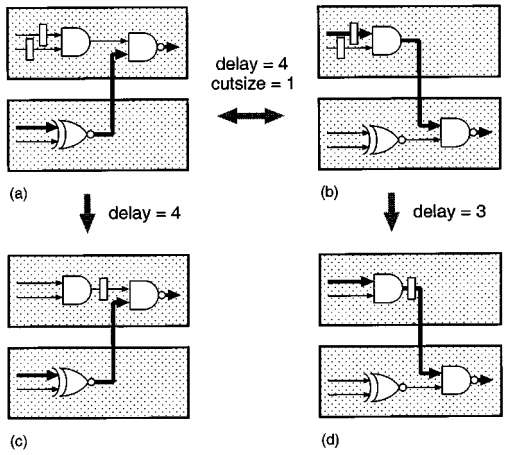


Fig. 11. Advantage of simultaneous partitioning and retiming for delay minimization. Critical paths are shown in thick lines.

For solution (a), retiming cannot help to reduce the delay [see Fig. 11(c)]. However, retiming can reduce the delay of solution (b) from 4 to 3 by repositioning of the flip-flops to hide part of the large interblock delay [see Fig. 11(d)]. This example suggests clearly that we can hide some global interconnect delay latency with proper consideration of retiming as we define the global interconnects during partitioning. We would like to emphasize that retiming over global interconnects is especially important to multigigahertz designs. Table 4 shows that the delay of a 2-cm global interconnect stays above 500 ps in each technology generation, even with use of new interconnect materials (as predicted in NTRS'97) and aggressive interconnect optimization. This implies that, for multigigahertz designs, we need multiple clock cycles to cross a global interconnect. This can only be achieved with retiming and pipelining on global interconnects in synchronous designs.[6]

Given a sequential circuit, the HPM algorithm computes a partitioning solution with the minimum clock period under retiming with possible node replication. The area of each block in the partitioning solution is bounded by a given number A. For delay computation, the HPM algorithm assumes each gate v has a delay of d_v, each global interconnect between blocks has a delay of D, and each local interconnect delay within each block is 0.[7] The computation of the minimum clock period is achieved by solving a sequence of the decision problem formulated as follows. For a sequential circuit with a given target clock period ϕ and a given area bound on each block, decide if there exists a partitioning solution with a clock period of no more than

[6]Asynchronous design has the potential not to be limited by the global interconnect delay. In particular, a globally asynchronous, locally synchronous (GALS) is a promising design method that is currently being studied by the researchers in the Gigascale Silicon Research Center (GSRC). The detailed discussion of this topic is beyond the scope of this paper. The reader may refer to related publications at http://www.gigascale.org for more details.

[7]This simplification is based on the fact that we lump the average local interconnect delay into the node delay d_v, assuming that local interconnect delays do not vary much.

ϕ under the given delay model after retiming and possible logic replication.

The HPM algorithm integrates several recent advances in circuit partitioning in developing a highly efficient performance-driven partitioning algorithm with retiming. The concepts and techniques used in HPM include:

- The iterative label computation technique to test the feasibility of a proposed clock period under simultaneous partitioning and retiming [47].
- The highly efficient label computation procedure based on the monotone property of the label computation and the efficient longest path computation [48].
- The multilevel partitioning paradigm, which has led the best cutsize minimization based partitioning package hMETIS [49].
- An efficient performance-driven clustering algorithm (PRIME) with retiming [48].
- An efficient multilevel clustering algorithm based on global edge separability for cutsize minimization [50].
- The multiway partitioning framework [51] to overcome the limitation of the recursive bipartitioning approach.

In particular, the HPM algorithm employs the multilevel partitioning framework as shown in Fig. 12. Over the past twenty years, multilevel methods have been studied extensively as a means of accelerating numerical algorithms for partial differential equations [52], [53]. Application area are quite diverse, including image processing, combinatorial optimization, control theory, statistical mechanics, quantum electrodynamics, and linear algebra. Multilevel techniques for VLSI physical designs are currently an area of intensive research activity. Much progress has been made in multilevel circuit partitioning and placement. hMETIS [54] produces the best cut size minimization in circuit partitioning, and mPL [55] achieves competitive circuit placement with over $10\times$ speed-up on designs with over 200K movable objects. The use of multilevel approach makes it feasible to extract the physical hierarchy from the flat design (which may consist of tens of millions of gates resulted from flattening the logic hierarchy).

The multilevel method is used in the HPM algorithm as follows. During the clustering phase of the HPM algorithm, a performance-driven clustering method with consideration of retiming [48] is used to build the base-level clustering structure to ensure the best possible subsequent retiming. Then it builds a multilevel clustering structure based on the global edge separability metric for cutsize minimization [50]. During the refinement phase of the HPM algorithm, simultaneous cutsize and performance-driven partitioning [56] is performed. It adopts a recently developed multiway partitioning framework [51] to overcome the limitation of the recursive bipartitioning approach. As a result, the HPM algorithm produces partitioning solutions that are: 1) 7% to 23% better in terms of delay compared to the best-known cutsize-driven hMETIS algorithm [49] with 19% increase in cutsize and 2) 81% better in terms of cutsize compared to the best-known delay-driven PRIME algorithm [48] with only a 6% increase in delay.

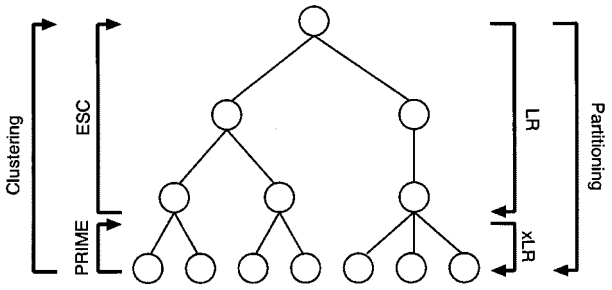


Fig. 12. Main flow of the HPM algorithm along with the illustration of its multilevel cluster hierarchy.

Table 7
The Number of Buffers Estimated for a High-Performance Design in Each Technology Generation

Technology(nm)	250	180	130	100	70
#buffers per chip	5K	25K	54K	230K	797K

Using the HPM algorithm, we generate the physical hierarchy as follows. Given a complex design specified in some HDL description (say either VHDL or Verilog language) in a hierarchical representation, we shall first perform a quick RTL synthesis and flatten the functional hierarchy as much as possible, down to a netlist of simple functional units (such as adders or decoders, but not necessarily gates) and predesigned IP blocks. Then, we shall apply either the two-way HPM algorithm recursively or the multiway HPM algorithm with coarse placement to generate a good physical hierarchy for subsequent interconnect planning and synthesis steps.

The delay model used in HPM is simplistic as it assumes that all global interconnects have a uniform delay D. This assumption is due to the lack of physical information of the blocks generated by the HPM algorithm. Currently, the HPM algorithm is being extended to perform coarse placement/floorplanning together with partitioning and retiming [57]. It shows that the coarse placement operation can be naturally integrated into the multilevel framework used in the HPM algorithm. The placement information provides much more accurate global interconnect delay estimation and allows the possibility of repositioning flip-flops to the *middle* of a long global interconnect (not just at its two ends). Experimental results show that combining partitioning, coarse placement, and retiming can provide an additional 23% delay reduction compared to the physical planning results obtained by separate performance-driven partitioning followed by floorplanning. Given the efficiency and quality produced by the HPM algorithm (especially when combined with coarse placement), we believe that it is capable of extracting good physical hierarchies for large-scale designs in nanometer technologies.

B. Buffer Block Planning

After the physical hierarchy is generated, the second step of the interconnect planning process is floorplanning with interconnect planning. Traditional floorplanning algorithms

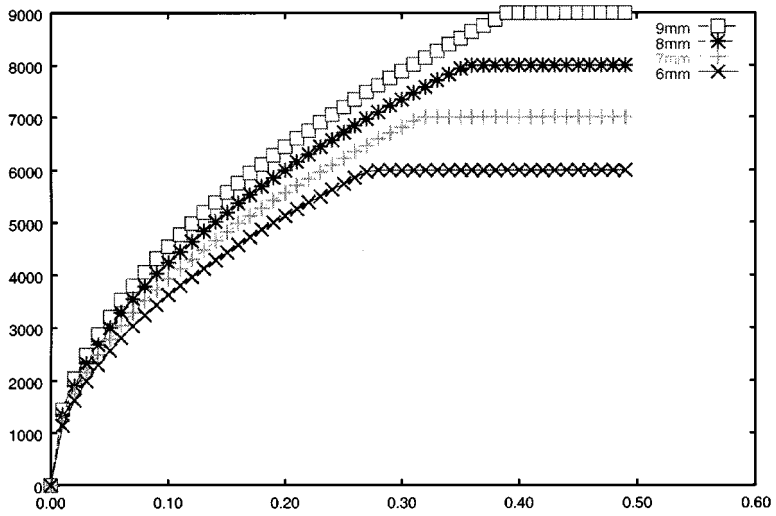


Fig. 13. The distance of feasible region for inserting a buffer under different delay constraints specified by δ for length 6-mm to 9-mm wires in the 0.18-μm technology. The x axis shows the value of δ and the y axis shows the length of the corresponding feasible region.

focus on dimensions and placement of functional blocks but ignore the interconnects associated with the design. We believe that floorplanning needs to interact closely with the interconnect synthesis tools to plan for the best interconnect topology, wire ordering and width, wire spacing, layer assignment, etc., for all global and semi-global interconnects in order to achieve the best possible circuit performance. As an example, we present here our recent work on automatic buffer block generation during floorplan design [58].

As shown in Section III-F, buffer insertion is a very effective technique to reduce the delay of long interconnects. Table 7 shows the number of buffers estimated for a high-performance design in each technology generation [46]. It is estimated that close to 800 000 buffers will be inserted in high-performance designs in the 70-nm technology [46]. If so many buffers are arbitrarily distributed over a chip, it may cause several problems: 1) it makes it difficult to use/reuse predesigned IP blocks; 2) it may complicate global/detailed routing and power/ground distribution; and 3) it may result in excessive area increase without proper planning. To overcome these problems, we propose to group buffers into buffer blocks. We have formulated the following *buffer block planning* (BBP) problem. Given an initial floorplan and the performance constraints for each net, we want to determine the optimal locations and dimensions of the buffer blocks such that the overall chip area and the number of buffer blocks after buffer insertion are minimized, while the performance constraint for each net is satisfied (assuming that it can be met by optimal buffer insertion). The output from our buffer block planning consists of the number of buffer blocks, each buffer block's area, location, and corresponding nets that use some buffer in this buffer block to meet the delay constraints.

Our study first shows that given a two-pin net, the *feasible region* (FR) for a buffer B, defined to be the maximum region where B can be located while still meeting the delay constraint, is quite large. Given the route from the source to the sink, for a given delay constraint T_{req}, the feasible region $[x_{\min}, x_{\max}]$ for inserting one buffer is

$$x_{\min} = \text{MAX}\left(0, \left(K_2 - \sqrt{K_2^2 - 4K_1K_3}\right)\Big/2K_1\right) \quad (8)$$

$$x_{\max} = \text{MIN}\left(l, \left(K_2 + \sqrt{K_2^2 - 4K_1K_3}\right)\Big/2K_1\right) \quad (9)$$

where

$$K_1 = rc$$
$$K_2 = (R_b - R_d)c + r(C_L - C_b) + rcl$$
$$K_3 = R_dC_b + T_b + R_b(C_L + cl) + 1/2rcl^2 + rlC_L - T_{\text{req}}$$

r is the unit length wire resistance, c is the unit length wire capacitance, T_b is the intrinsic delay for the buffer, C_b is the input capacitance of the buffer, and R_b is the output resistance of the buffer. Note that for (8) and (9) to be valid, $K_2^2 - 4K_1K_3 \geq 0$ shall hold. Otherwise, no feasible region exists, and the initial floorplanning/timing budget has to be modified. Fig. 13 shows the FR for inserting one buffer to an interconnect of length from 6 mm to 9 mm in the 0.18-μm technology specified in NTRS'97. We first compute the best delay T_{best} by inserting one buffer, then set the delay constraint to be $(1+\delta)T_{\text{best}}$, with δ varying from 0 to 50%. The x axis shows the value of δ and the y axis shows the length of the corresponding FR, i.e., $x_{\max} - x_{\min}$. It is interesting to see that even with a fairly small amount of slack, say 10% of T_{best}, the FR can be as much as 50% of the total wirelength!

When the route from the source to the sink is not specified, the feasible region is not just an interval, but a two-dimensional region which is the *union* of the one-dimensional feasible regions of *all* possible routes from source to sink. The *optimal* buffer locations, in this case, form a line segment of slope $+1$ or -1, for buffer insertion. Fig. 14 shows an example of *a two-dimensional feasible region* with some

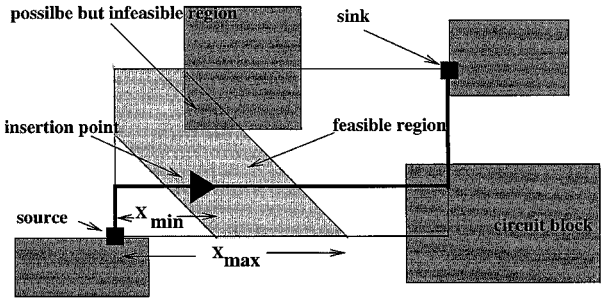


Fig. 14. Two-dimensional feasible region. The existing circuit blocks act as obstacles for buffer insertion.

routing obstacles. Obviously, routing obstacles need to be deducted from the feasible region computation.

When multiple buffers between the source and the sink of a net are needed to meet the delay constraint, we can compute the feasible region of each buffer using a simple analytical formula similar to that in (8) and (9), assuming that all other buffers are taking their optimal positions [58]. In this case, however, after a buffer is placed (i.e., "committed") to a position within its feasible region, we need to update the feasible regions of all other *unplaced* buffers of the same net to safely meet its delay constraint. Since we have an analytical formula, this update can be computed in constant time. A more recent study suggests a way to compute more conservative feasible regions to allow multiple buffers to be placed or moved simultaneously without violating the performance constraints [59].

Given the efficient procedures for computing feasible regions for buffer insertion, our buffer block planning algorithm works as follows. First, it builds the horizontal and vertical polar graphs of the given floorplan to keep track of available space for buffer insertion. The available space is divided into tiles. An *area slack* is computed for each tile, which measures the impact on the overall chip area if the tile area is increased. The algorithm iteratively chooses the tile with the maximum area slack and inserts a buffer with the least flexibility (i.e., the minimal feasible region) into this tile. The area slacks of tiles and the feasible regions of the affected buffers (for multiple-buffer nets) may need to be updated after each buffer assignment. It was shown in [58] that this simple buffer planning scheme works well, mainly due to the large degree of freedom from feasible regions for buffer insertion. Experimental results show that the proposed algorithm can reduce the number of buffer blocks by a factor of 2.4× with smaller chip area and a better chance of meeting timing constraints and smaller overall chip area.

C. Wire Width Planning

After physical hierarchy generation and floorplanning with interconnect planning, we know the wirelength distribution on each layer. In this case, it is possible to perform interconnect architecture planning for each layer for optimizing the performance and cost of the overall design as discussed in the beginning of Section V. This section presents our results on wire width planning, which is part of our overall effort on interconnect architecture planning. As stated in Section III-B, wire sizing is an effective technique for reducing interconnect delays. However, having many different wire widths will considerably complicate the layout design, especially the routing process. Therefore, it is interesting to investigate the possibility of using a small set of predetermined "fixed" widths in each layer to get close to optimal performance for all interconnects in a wide range of wirelengths in that layer (not just one length).

Given the wirelength distribution in each layer (which can be obtained accurately after floorplanning with interconnect planning), the *wire-width planning problem* is to find the best width vector $\vec{W}$ for that layer such that the following objective function:

$$\Phi\left(\vec{W}, l_{\min}, l_{\max}\right) = \int_{l_{\min}}^{l_{\max}} \lambda(l) \cdot f\left(\vec{W}, l\right) dl \qquad (10)$$

is minimized, where $\lambda(l)$ is the distribution function of wirelength l, $l_{\min}$, and $l_{\max}$ are the minimum and maximum wirelengths for this metal layer, and $f(\vec{W}, l)$ is the objective function to be minimized by the design. In this study, we choose $f(l)$ to be of the form $A^j(\vec{W}, l) \cdot T^k(\vec{W}, l)$, where $A(\vec{W}, l)$ and $T(\vec{W}, l)$ denote the area and delay using $\vec{W}$. For one-width design, $\vec{W}$ has only one component W. For two-width design, $\vec{W}$ has two components W_1 and W_2. If we set $j = 0$ and $k = 1$ in (10), the objective is to achieve the best delay. Insisting the minimum delay in wire sizing can be very costly in terms of wire width, as the delay/width curve is very flat while approaching the optimal delay. Our empirical study suggests that the AT^4 metric (i.e., $j = 1$ and $k = 4$) leads to area-efficient performance optimization in general. For example, it was shown in [39] that, under the 0.10-μm technology, optimal one-width solution for a 2-cm interconnect under the AT^4 metric uses over 60% smaller wiring area with only a 10% increase in delay compared to that obtained for delay optimization only. The wire width planning results presented in this section uses the AT^4 metric. But our solution technique is general for optimizing other metrics as well.

Our approach to the wire-width planning problem is fairly straightforward. We find the best one-width or two-width pair to minimize the objective function in (10) by exhaustive enumeration through all possible widths or all possible wire-width pairs, respectively. This method clearly cannot scale to find a wire-width planning solution with many widths. But this is not a problem, as we are only interested in finding a very small number of widths per layer as the planning solution. Using this approach, we in fact have achieved a rather surprising result which suggests that two *predetermined* wire widths per metal layer are sufficient to achieve near-optimal performance for *a wide range of nets* in that layer. For example, for layers 7 and 8 in the 0.10-μm technology, assuming that the wirelength in this layer pair distributed evenly from 7.57 to 24.9 mm (according to the interconnect length distribution model described in [39]), our wire-width planning tool suggests that the best one-width is 1.98 μm and that the best two-width design consists of wires of widths 1.0 μm and 2.0 μm. Table 8 shows the comparison of using the one-width, two-width, and many-width designs

Table 8
Comparison of Using One-Width Design, Two-Width Design, and Many-Width Design (up to 50× min Width) Using GISS for Wire Sizing and Spacing. Layers 7 and 8 of 0.10-μm Technology are Used, with Wirelength Ranging from 8.04 to 22.8 mm. Driver Size is Assumed to be 250× min Size

Scheme	pitch-sp=2.0 μm			pitch-sp=2.9 μm			pitch-sp=3.8 μm		
	T_{avg}	ΔT_{max}	avg-w	T_{avg}	ΔT_{max}	avg-w	T_{avg}	ΔT_{max}	avg-w
one-width	0.245	28.2%	1.98	0.177	15.7%	1.83	0.143	5.9%	1.63
two-width	0.215	7.0%	1.08	0.167	5.9%	1.23	0.140	3.9%	1.41
many-width	0.204	-	1.03	0.159	-	1.19	0.136	-	1.38

by running GISS (global interconnect sizing and spacing) algorithm discussed in Section III-C (also in [25]). Three different pitch spacings (pitch-sp) between adjacent wires in layers 7 and 8 of the 0.10-μm technology are used. For each pitch-sp, we compare the average delay, the maximum delay difference (in percentage) from GISS ($\Delta T_{\max}$) for all lengths, and the average width. For pitch-spacing of 2.0 μm, one-width design has an average delay about 14% and 20% larger than those from the two-width design and the many-width design, respectively. Moreover, it has an average wire width (thus area) about 1.83× and 1.92× of those from two-width design and many-width design, respectively. The two-width design, however, achieves close to the optimal delay as computed by the many-width design obtained by the GISS algorithm (just 3 to 5% larger) and uses only a slightly larger area (less than 5%) than that of the multiwidth design using GISS. When the pitch spacing becomes larger, the differences between one-width, two-width, and many-width designs get smaller.

In Table 8, we also list the maximum delay difference ($\Delta T_{\max}$) between the one-width and two-width designs compared to the many-width design. It is an important metric as it can bound the error of the corresponding wire-width planning solution under *any length distribution function* $\lambda(l)$ in (10). For the two-width design shown in the table (derived from uniform distribution $\lambda(l) \equiv 1$), since the maximum delay difference $\Delta T_{\max}$ is only 3.9% to 7%, one can conclude that this two-width design will differ from the optimal design (using possibly many widths) by at most 3.9% to 7% for *any distribution function* $\lambda(l)$. The reader may refer to [39] for more details.

The fact that two widths are sufficient for each layer greatly simplifies the detailed routing problem (a full-blown gridless router may not be necessary) and possibly other problems, such as RC extraction and layout verification.

VI. Interconnect Synthesis

The second major component in our interconnect-centric design flow is interconnect synthesis. Given a logic synthesis and placement solution, interconnect synthesis determines the optimal or near-optimal interconnect topology, wire ordering, buffer locations and sizes, wire width and spacing, etc., to meet the performance and signal reliability requirements of all nets under the area and routability constraints. This is similar to the traditional global routing step, but with much emphasis on interconnect performance and signal reliability optimization. In our system, interconnect synthesis

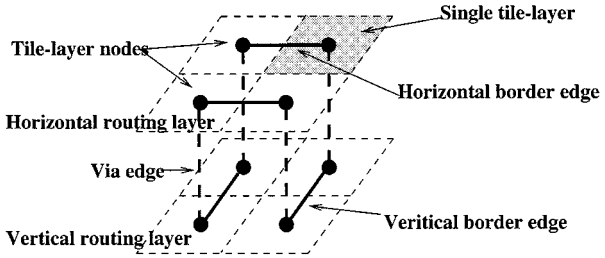


Fig. 15. A two-layer, four-tile routing problem.

is achieved in two steps: 1) multilayer general-area global routing for delay and congestion optimization and 2) wire ordering and spacing for noise and routability optimization. These two steps will be discussed in detail in the remainder of this section. The output of interconnect synthesis provides the topology, width, spacing, and ordering specifications of all the nets to the subsequent interconnect layout module (to be presented in Section VII) for detailed routing.

A. Performance-Driven Global Routing

It is predicted in NTRS'97 that there will be 8 to 9 routing layers in the 70-nm technology generation. Any routing system for nanometer technologies has to be capable of multilayer routing. Since the concept of routing channels is no longer well defined in multilayer routing, and over-the-cell or over-the-block routing is widely used, we use a tile-based structure to model the available routing resources. We divide the multilayer routing region into a set of "tiles," with each tile consisting of a number of layers. We use "tile-layer" to indicate a single layer within a tile. The entire routing region is modeled as a graph, with tile-layers as nodes, and both vias and the "borders" between tile-layers as edges. Preferred-direction routing is obtained by including (or excluding) appropriate border edges. Fig. 15 shows our routing model for a four-tile, two-layer routing example. Preferred-direction routing is used in this example, with the top layer for horizontal wires and the bottom layer for vertical wires. The size of a tile is chosen in such a way so that the global router can perform various interconnect optimization operations (as discussed in Section III) effectively on the underlying global routing grid. We suggest using one-half (or some other fraction) of the critical length for buffer insertion (defined in Section IV-B1) as the length for a tile, since it will provide sufficient details for the global router to decide the buffer locations and wire width changes.

Our global router employs two basic routing engines for routing congestion optimization: one is based on the traditional rip-up and reroute approach, and the other is based on iterative deletion. The rip-up and reroute portion of our global router is similar to that of [60], in that we iteratively change the cost of each node in the routing graph based on the current congestion in that tile-layer so that the router tends to converge to a low-congestion solution. The choice of routing cost functions and net routing orders was discussed in detail in [61]. The iterative deletion method was first proposed in [62] for standard cell global routing, and further refined in [63] and this work. It begins with multiple routing paths for

each two-pin net (a multipin net will be first decomposed into a set of two-pin nets), and iteratively removes a redundant routing path (for a net still having multiple paths associated with it) with the highest cost (measured in terms of congestion and/or other performance-based metrics) until each net has only one routing path. The iterative deletion method provides a more *global* view of the overall routing congestion and is shown to be very effective for congestion optimization [62], [63]. In fact, the two routing optimization strategies can be combined naturally: we can use rip-up and reroute to get multiple routing paths for some or all nets, and then use iterative deletion to select the best route for each net. This process may be repeated so that more candidate paths (computed by rip-up and reroute) can be used at the next round of iterative deletion.

The performance optimization capability is achieved by the *Required-Arrival-Time Steiner tree (RATS-tree)* construction algorithm proposed in [14]. It computes *a set of solutions* that meets the given required times at the sinks using a bottom-up dynamic programming approach. It can perform both routing topology optimization and wire sizing. The fact that the RATS-tree algorithm may produce *a set of* high-performance routing structures for each timing-critical net works very well with the iterative deletion method. Our global router starts with multiple high-performance routing structures for each timing-critical net and finally selects one of the structures for each net for overall routing congestion minimization. As a result, our global router can achieve performance optimization for a large percentage of nets with little increase in overall routing congestion [64].

Note that although the original RATS-tree algorithm does not consider buffer insertion, it is straightforward to extend it to consider buffer insertion in its bottom-up dynamic programming framework in the same way as the WBA-tree algorithm presented in Section III-E. Currently, we are extending our global router so that it can invoke any of the interconnect optimization routine available in the TRIO package (including the WBA-tree algorithm), enabling it to consider topology optimization, buffer insertion, wire sizing, or a combination of these optimization techniques during global routing.

B. Pseudo Pin Assignment for Noise Control

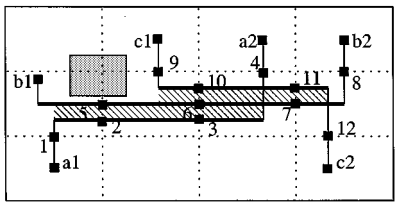
Equipped with topology optimization, buffer insertion, and wire sizing optimization techniques, the tile-based global router can effectively minimize the interconnect delay. However, it has little control of coupling noise due to the lack of information of wire ordering and spacing for estimating the coupling capacitance. Since the coupling noise is becoming a serious consideration in nanometer designs (as shown in Figs. 1 and 2 in Section I) the second step in our interconnect synthesis phase is to determine wire ordering and spacing for noise and routability optimization by solving the pseudo pin assignment problem. Given a set of routing tiles and a global routing solution associated with it, a *pseudo pin* of a net is a wire crossing point of the net at some tile boundary.[8] The *pseudo pin assignment problem* is to determine the locations of all pseudo pins of all the nets. Because pseudo pin assignment determines the wire ordering and spacing to a large extent, it can be used effectively for crosstalk noise control. Moreover, it provides an important bridge between global routing and detailed routing by specifying pseudo pin locations on the boundary of each tile. Otherwise, the detailed routing problem for each tile is not well defined. Therefore, the pseudo pin assignment problem also has a significant impact on the routability, wirelength, and via count of the final layout solution. Fig. 16 illustrates the impact that pseudo pin assignment. It shows two pseudo pin assignments of the same global routing solution on 3×4 tiles. The tile boundaries are shown as dotted lines. Pseudo pins are labeled 1 to 12; real pins are labeled $a1$, $a2$, $b1$, $b2$, $c1$, and $c2$; and the gray areas are obstacles. The possible detailed routing solutions according to the pseudo pin assignment are also shown in the figure. The narrower solid lines represent wires on layer 1 (vertical) and the wider solid lines represent the wires on layer 2 (horizontal). The shaded areas indicate the coupling between the wires of net $b1$–$b2$ to other wires under the minimum spacing. We can see that the total coupled length (length of shaded areas) is roughly 4 (tile widths) in Fig. 16(a), but decreases to 2 (tile widths) in Fig. 16(b). The detour on net $b1$–$b2$ is roughly 1 (tile height) in Fig. 16(a) and 1.5 (tile height) in Fig. 16(b). This example shows that different pseudo pin assignments can lead to considerably different via counts, wire lengths, and capacitance coupling among nets.

The objective of our pseudo pin assignment is to determine the locations of pseudo pins to minimize a weighted sum of the total wirelength TL and the estimated number of required vias VC under the crosstalk constraints. We choose this objective because crosstalk noise only needs to be controlled in a safe range (instead of being eliminated completely), while the wirelength and the number of vias usually need to be minimized as much as possible.

Given a global routing solution, our pseudo pin assignment algorithm, named the PPA algorithm, assigns pseudo pins layer by layer, as one can easily verify that the assignment of pseudo pins in one layer has little effect on pseudo pin assignments on different layers [65]. Moreover, since each pseudo pin is confined to a single tile boundary, we only need to assign pseudo pins for one row (or column) of tiles at a time so that we do not need to work on the entire layer all at once.

Given a row of tiles to be processed, the PPA algorithm first decomposes their vertical boundaries to a set of intervals induced by maximum horizontal strips. The *maximum horizontal strips*, which were first defined in [66], are strips (rectangles) that form a partition on the empty space in a routing region such that no strip is horizontally adjacent to any other strips. In our algorithm, the rectangle objects are

[8]In contrast, the original pins in the design are called the "*real pins*" in order to be distinguished from the pseudo pins.

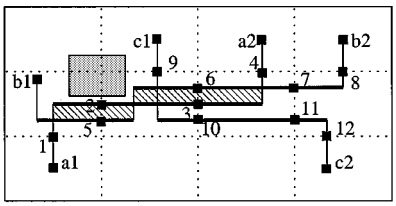


Vias = 6

Coupling capacitance on net b1-b2 ≈ 4

Detour on net b1-b2 ≈ 1

(a)



Vias = 8

Coupling capacitance on net b1-b2 ≈ 2

Detour on net b1-b2 ≈ 1.5

(b)

Fig. 16. Impacts of pseudo pin assignment. (a) Pseudo pin assignment with less vias and detours, but larger coupling on net $b1$-$b2$. (b) Pseudo pin assignment with smaller coupling on net $b1$-$b2$, but more vias and detours.

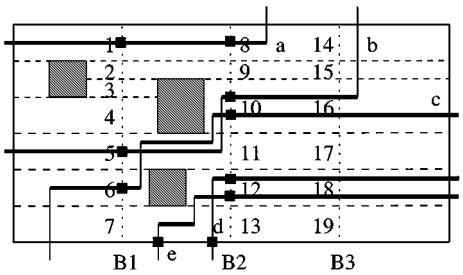


Fig. 17. Tile boundary decomposition.

obstacles, real pins, or the projections of real pins from adjacent layers.[9] Fig. 17 shows an example of the maximum strips formed on a row of four tiles. The gray areas are rectangle objects, which are obstacles or real pins. The dashed lines are the horizontal lines extended from the top and bottom boundaries of the rectangles. The decomposed intervals are labeled 1 to 19. After tile boundary decomposition, our PPA algorithm solves the pseudo pin assignment problem in two steps: coarse pseudo pin assignment (CPPA) and detailed pseudo pin assignment (DPPA). In CPPA, each pseudo pin is estimated with a crosstalk-safe spacing from its noise constraint and assigned to an interval. In DPPA, each pseudo pin is assigned to an exact location and crosstalk noise constraints must be satisfied. These two steps are briefly described in the next two paragraphs.

In the CPPA step, we first estimate the crosstalk-safe spacing for each pseudo pin by assuming that the pseudo pin is adjacent to a pair of pseudo pins which have the average capacitance, resistance, driver/receiver characteristics. We assume each pseudo pin has a noise budget that can be calculated from the noise constraints. From the noise budget for the pseudo pin, we can calculate the maximum allowed coupling capacitance using the noise estimation model in [67] (or any reasonable noise estimation model), which in turn allows us to find the minimum separation distance to its neighbor by interpolation in the capacitance lookup table. The resulting minimum separation distance is the estimated crosstalk-safe spacing for the pseudo pin. Next, a coarse routing graph is generated from the boundary decomposition. Each vertex represents either an interval or a connection point (a real pin or a pseudo pin). Each edge connects a pair of vertices which can reach each other without crossing a tile boundary, with proper cost associated to reflect the estimated wirelength and via count for this connection. Then, we compute the assignment of pseudo pins to the intervals by using one of the two following approaches. One is the *net-by-net* approach that uses a shortest path algorithm on the coarse routing graph to assign nets one by one. The other is the *iterative deletion* approach (similar to that used in global routing in the preceding section). It works on one boundary at a time and simultaneously assigns all the unassigned nets crossing the boundary. The algorithm iteratively picks the most crowded unassigned boundary to do the assignment. Experimental results show that the iterative deletion approach produces better results with longer computation time.

In the DPPA step, we assign pseudo pins to the exact locations in each interval. Pseudo pins of the same maximum horizontal strip (used in the tile boundary decomposition) are assigned at the same time in a way similar to channel routing. Each subnet in the strip is first assigned as a single wire segment. We determine the ordering of these wire segments using a simple packing algorithm and the spacing of these wire segments based on noise budget in a way similar to the CPPA step. We may have an assignment that exceeds the strip height if we insist that all the pseudo pins in every subnet in the strip must be aligned. If this happens, we apply a heuristic algorithm to break up some alignments and introduce jogs (dog-legs) to resolve the problem in a way similar to channel routing. The objective of this heuristic is to align as many pseudo pins as possible.

We have tested our PPA algorithm by the following. We first run our PPA algorithm on two sets of test cases under different configurations (with and without noise control). If the crosstalk noise constraints are not considered, the estimated average noise in PPA is 0.11–0.26 V_{DD} with up to 36% of nets that have noise larger than 0.3 V_{DD}. If the crosstalk noise constraints are considered, the average noise estimated in PPA is reduced to 0.10–0.16 V_{DD} and with no nets with noise larger than 0.3 V_{DD}.

We then use our multilayer gridless detailed router (will be described in Section VII) to obtain the detailed routing results. From the detailed routed results, we do a 2-D extraction to find out the line resistance, the line capacitance and coupling capacitance for all the nets and use the same formula as

[9]In fact, each rectangle is expanded by half of the minimum spacing on that layer.

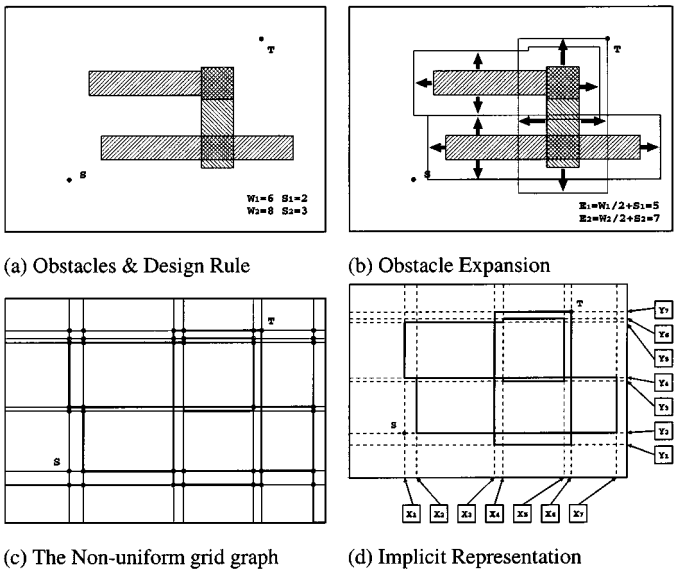


Fig. 18. Construction of implicit connection graph. (a) A multilayer variable width and variable spacing routing problem. (b) The problem is transferred into a zero-width routing problem using expansion. (c) A connection graph for finding the path from s to t. (d) An implicit representation of such a graph using two sorted arrays.

in PPA to calculate the crosstalk noise from these data. For the experiments without noise control, the average noise in each test case after detailed routing is 0.13 V_{DD}–0.22 V_{DD} with up to 11% of nets larger than 0.3 V_{DD}. For the experiments with noise control, the average noise after detailed routing in each case reduces to 0.11 V_{DD}–0.15 V_{DD} with no crosstalk noise violations. These data verified that the noise control performed in the pseudo pin assignment can be preserved after detailed routing with high fidelity. Even without rip-up and reroute, the detailed routing completion rate is 93%–99% and the average vias per net is only 0.7–1.4, which suggests that noise control in PPA does not sacrifice routability. A more detailed description of this work is available in [65].

VII. Interconnect Layout

The final step of our interconnect-centric design flow is interconnect layout. Aggressive interconnect synthesis and optimization often result in complex interconnect structures with many buffers, variable widths within the same net, or even variable widths within the same segment. Different spacing rules are also needed for crosstalk control and minimization. These requirements need to be supported by an efficient multilayer gridless detailed routing system. In the past few years, we have developed a novel gridless detailed routing system, named Dune, to support multilayer, variable-width, variable-spacing routing. Dune has two major components: a point-to-point gridless routing engine and a route planning engine. Both will be described in more detail in this section.

A. Point-to-Point Gridless Routing

In general, there are two types of approaches to the gridless routing problem. One approach uses the tile-based algorithms [68]–[70]. The routing region is partitioned into tiles induced by the boundaries of obstacles. The routing problem is reduced to searching a tile-to-tile path among these tiles, usually based on the corner-stitching data structure [66]. The other approach uses the connection graph-based algorithm [71]. A connection graph is built based on the obstacles in the routing region, and usually the special width and spacing requirements for the net to be routed are encoded in the graph. A maze-searching algorithm is applied on the graph to find the route. Since tiles are more complex to manage, and a tile-to-tile path needs postprocessing to obtain a final design-rule-correct route, we use a connection graph-based approach, with two novel contributions: use of a nonuniform grid graph and use of an implicit representation of the graph.

Given a routing region (usually a tile defined by the global router), we first construct a connection graph called *Non-Uniform Grid Graph*, denoted as G_S, based on the expansion of rectangular obstacles in the routing region according to wire/via width and spacing rules. In the routing region, the preexisting routings and objects can be most conveniently represented as a set of possibly overlapping rectangles $R = \{r_1, r_2, \ldots, r_{N_R}\}$ located in at different layers as illustrated in Fig. 18(a). The layout design rules create an obstruction zone [72] around each obstacle where the centerlines of wires and center of vias cannot be placed. That is, the centerline of a wire of width w must be at least $dw_i = (w/2 + ws_i)$ away from the edge of the obstacle r_i where ws_i is the wire spacing between the current net and the obstacle r_i. We let $\tilde{R}$ be the set of rectangles that are expanded from those in R by dw_i in each of the four rectilinear directions, as shown in Fig. 18(b). Note that ws_i does not have to be the minimum wire-to-wire spacing and may vary from net to net due to various kinds of interconnect optimizations for delay and/or noise minimization. Similarly, we can create the set of rectangles expanded according to via width and spacing rules,

denoted as $\tilde{R}^v$. Given a multilayer routing problem with the obstacle set R, a source s, and a sink t, our *Non-Uniform Grid (NUG) Graph* is defined to be an orthogonal grid graph where its x grid locations are the vertical boundary locations of $\tilde{R}$ and $\tilde{R}^v$ plus the x locations of s and t, and its y grid locations are the horizontal boundary locations of $\tilde{R}$ and $\tilde{R}^v$ plus the y locations of s and t. Any location defined by the intersection of these two x and y grid lines is a valid graph node if it is not contained[10] by any rectangle in $\tilde{R}$ or $\tilde{R}^v$, as shown in Fig. 18(c).

We have proved that such a graph guarantees to include a gridless connection of the minimum cost in multilayer variable width and variable spacing routing [73]. Compared to a uniform grid graph required for gridless routing, our nonuniform grid graph is much sparser, as a very fine grid must be used in a uniform routing graph to support a reasonably large set of widths and spacings in gridless routing. Although it is possible to construct a slightly smaller routing graph yet still ensure the existence of a shortest path from s to t as discussed in [73], we choose to use the NUG graph due to its simplicity and regularity resulting from its grid structure. As a result, we can come up with a simple, implicit representation of the graph to support maze routing so that the NUG graph is highly compressed in storage and efficient in query. The implicit representation used in our router is simply two sorted arrays, X_S and Y_S, to store the x coordinates and y coordinates of the NUG graph, respectively. The two-array data structure is linear in terms of the number of obstacles in the routing region (including existing routes), so it is highly memory efficient. Note that no precomputation is needed to obtain the implicit representation other than sorting the coordinates which can be maintained incrementally. We generate the graph nodes and edges on-the-fly during the routing process. Generation of a new graph node during routing involves a *query* that consists of two steps. First, compute the possible position of the neighbor of the current node in routing, and second, determine the feasibility of the position. The computation of the possible neighboring position is trivial: Suppose the current route ends at ($X_S[i]$, $Y_S[i]$) and the maze expansion direction is *right*, the next possible graph node position is simply ($X_S[i+1]$, $Y_S[i]$).

The feasibility test is slightly more complex. The position is feasible for placing a wire or via if it is not enclosed by the applicable expanded rectangles in $\tilde{R}$ or $\tilde{R}^v$, respectively. Therefore, finding the feasibility of a node requires a *point enclosure query* defined as follows: given a set of rectangles $R = \{r_i | i = 1, 2, \ldots, N_R\}$ and a point v, return the set of rectangles that contain v. We have developed a novel data structure using a combination of slit tree and interval tree, and cache structure to support efficient point enclosure queries for mazing routing with the implicit representation of the NUG graph. Our experiments show that this data structure is very efficient in memory usage while very fast in answering maze expansion related queries.

[10]A point p is contained by a rectangle r_k if the point falls within the *open* rectangle r_k.

The efficiency of our point-to-point gridless routing engine is validated when we apply it to the incremental routing problem in [74] (called the ECO problem in that paper). It was compared with the explicit uniform grid-based approach and Iroute [75], [76], a well-known tile-based router for gridless routing, The results show that the implicit representation of the NUG graph is very efficient in memory usage, 14× smaller than that of the explicit representation and 2–3× smaller than Iroute. The queries supported by our data structure are also very fast. The run time of our maze routing algorithm is 2–4× faster than Iroute.

B. Route Planning

To overcome the net ordering problem associated with the net-by-net routing approach and support efficient rip-up and reroute in gridless routing, we developed a coarse grid-based route planning algorithm. It uses a line-sweeping algorithm to find all routing obstacles in each grid cell and uses exact gridless design rules (variable width and variable spacing) to accurately estimate the available routing resources in each grid cell. It uses a multi-iteration planning method to overcome the net ordering problem and evenly distributes the nets into routing regions. It plays a similar role as the conventional congestion-driven global router, but models the available routing resources more accurately and interacts with the underlying point-to-point gridless routing engine much more closely. The route planning algorithm provides three capabilities.

- With efficient multi-iteration route planning on the coarse grid, it spreads out the nets to reduce the overall congestion and thus improves the routability.
- It constrains each net's searching space to a set of coarse grid cells identified by the planned route, and thus greatly speeds up the point-to-point gridless routing for the net. For example, given the route planning solution shown in Fig. 19(a), gridless maze routing for net (s_2, t_2) on the NUG graph can be constrained to the shaded cells.
- It provides an efficient framework to support rip-up and reroute, which is a difficult problem in gridless routing. During the reroute phase, we apply two methods to find the alternative route for the blocked net, based on the updated congestion information. One is *local refinement*. If following the originally planned route fails to complete routing, the gridless routing engine will search more cells around the blocked cell. For example, assuming net (s_3, t_3) is routed, following the planned route for net (s_2, t_2) as shown in Fig. 19(a) does not lead to a valid solution. In this case, the search region by the gridless routing engine is expanded as shown in Fig. 19(b). The other method is *rerouting*. It finds an alternative route on the coarse grid for the net. The cells along the previous path are given extra penalties to guide the new route to be away from it. The replanned

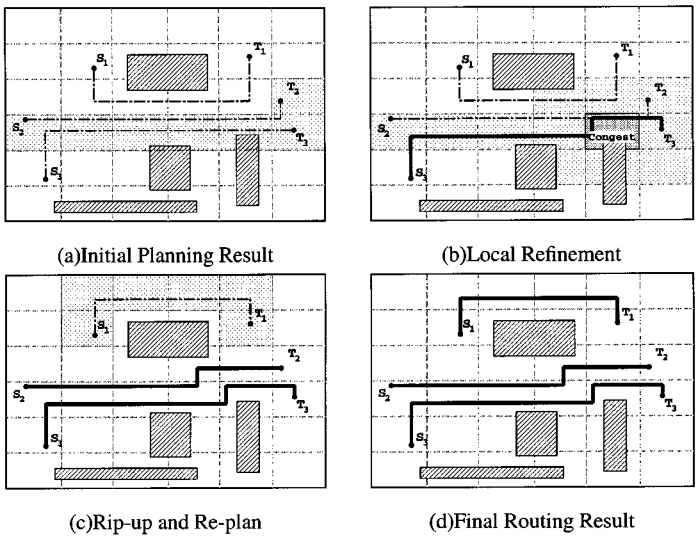


Fig. 19. Replanning strategies. (a) An initial planning result. However, due to the local congestion after routing net 3, a local refinement algorithm is used to reroute net 2, as shown in (b). (c) A rip-up and replan method is used to reroute net 1. (d) Final routing result.

result is then given back to the gridless routing engine again to search for the final connection. Fig. 19(c) and (d) show the rip-up and replan process when net $(s1, t1)$ failed to route.

Experimental results show that using the route planning algorithm in our gridless detailed routing can improve the routability and also speed up the run time significantly, by a factor of 3 to 17. A detailed description of our gridless detailed router can be found in [73] and [74].

VIII. Summary and Ongoing Work

In this paper, we presented our research efforts and results on developing an interconnect-centric design flow. The research involves three major design phases: interconnect planning, interconnect synthesis, and interconnect layout. An efficient interconnect optimization library (the TRIO package) and a set of efficient interconnect performance estimation models (the IPEM package) have also been developed to support efficient interconnect optimization and performance estimation of optimized interconnects during interconnect planning and synthesis.

With a good understanding of these building blocks, we can revisit the overall flow for interconnect-centric designs, as shown in Fig. 5. Our flow can be summarized as follows. In order to cope with the design complexity of gigascale integration in the nanometer technologies, we would like the designer (or the design team) focus on designs primarily at the architecture or conceptual level. Given a design specification (usually in a HDL specification such as Verilog or VHDL) as the output of the architecture or conceptual level design, our interconnect-centric flow first goes through the interconnect planning phase which transforms the functional hierarchy embedded in the HDL specification into a good physical hierarchy and performs coarse placement with global interconnect planning and interconnect architecture planning (when appropriate). It is possible for physical hierarchy generation to be performed together with coarse placement and global interconnect planning at the same time, as the global placement and interconnect planning usually influence the physical hierarchy generation. Interconnect performance estimation models are used extensively during interconnect planning for predicting the performance of the optimized interconnects. After physical hierarchy generation, coarse placement with global interconnect planning, we shall have a good first-order estimation of the overall circuit performance (which is determined primarily by global interconnects). We can quickly provide feedback to the designer to indicate if the proposed architectural or conceptual level design is feasible. Therefore, the designer can quickly iterate with the interconnect planning tool to evaluate multiple architecture or micro-architecture designs, and converge to the most promising one(s) for further refinement.

After interconnect planning, the next of phase of the design flow is synthesis and placement for each module under the physical hierarchy, as shown in the shaded box in Fig. 5. Currently, we are using off-the-shelf synthesis and placement techniques for this step (such as using the Design Compiler from Synopsys or the SIS/VIS package from UC Berkeley for logic synthesis and the TimberWolf or GordianL package for placement). We tend to believe once the physical hierarchy and global interconnects are defined, existing synthesis and placement algorithms can work well at the module level which contains mainly local interconnects, as argued in [77]. In particular, gain-based synthesis can be used to synthesize small to medium size logic blocks under the physical hierarchy [78], [79]. We are also starting a new project at UCLA on placement-driven synthesis to investigate if one can improve the result from this step significantly by combining synthesis and placement at the module level.

Once synthesis and placement for each module is determined, we perform interconnect synthesis, which includes performance-driven global routing with various interconnect optimizations for delay minimization followed by pseudo pin assignment with noise minimization.

Finally, a gridless routing system is used to complete interconnect layout to implement various kinds of optimized interconnect structures. It includes a coarse grid based route planning engine and an efficient point-to-point gridless routing engine working on the implicit representation of the underlying nonuniform grid graph.

All these modules have been implemented, and they interact through a common hierarchical data model (HDM). The HDM provides a complete functional and physical representation of the design, including the structural view, the functional view, the physical view, and the timing view so that logic transformation, interconnect planning/optimization, or layout design can be carried out at every phase of the design process.

Each module in this design flow has been fully verified and has shown very promising results, as presented in various sections throughout this paper. We are in the process of performing integrated test of the overall flow and design methodology. We are integrating all the modules into the proposed interconnect-centric design flow and running several complete designs through such flow. Our test suite includes the PicoJava processor from Sun Microsystems and a few designs from IBM. (IBM has installed IDM, the IBM Data Model, at UCLA and we are developing an interface to it.) We hope to report complete experimental results in the near future. We believe that such an interconnect-centric design flow will effectively bridge the gap between high-level design abstraction and physical-level implementation, reduce or eliminate the uncertainty due to interconnects on system performance and reliability, and assure design convergence between synthesis and layout.

ACKNOWLEDGMENT

The author would like to thank the students and researchers (current and former) at the UCLA VLSI CAD Laboratory who have contributed to the development of the interconnect-centric design flow presented in this paper. They include C.-C. Chang, L. He, K.-Y. Khoo, C.-K. Koh, T. Kong, H. Leung, H.-C. P. Li, S. Lim, P. Madden, T. Okamoto, D. Z. Pan, T. Shibuya, C. Wu, and X. Yuan. The author would also like to thank T. Drumm from IBM for stimulating discussions regarding hierarchical designs and for providing the design plot shown in Fig. 10. The author is very grateful to X. Yuan for her assistance in preparing this manuscript.

REFERENCES

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electron. Mag.*, vol. 38, pp. 114–117, Apr. 1965.

[2] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*, 1997.

[3] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 1999.

[4] K. Nabors and J. White, "FastCap: A multiple accelerated 3-D capacitance extraction program," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1447–1459, Nov. 1991.

[5] J. Cong, L. He, C.-K. Koh, D. Z. Pan, and X. Yuan. (1999) *UCLA Tree-Repeater-Interconnect-Optimization Package (TRIO)* [Online]. Available: http://cadlab.cs.ucla.edu/software_release/trio/htdocs/

[6] J. Cong, K. S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," in *Proc. Design Automation Conf.*, 1993, pp. 606–611.

[7] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably good performance-driven global routing," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 739–752, June 1992.

[8] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*. Boston, MA: Kluwer, 1994.

[9] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integr. VLSI J.*, vol. 21, pp. 1–94, 1996.

[10] J. Cong and P. H. Madden, "Performance driven routing with multiple sources," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1995, pp. 1203–1206.

[11] J. Cong, A. Kahng, and K. Leung, "Efficient algorithm for the minimum shortest path Steiner arborescence problem with application to VLSI physical design," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 24–38, 1999.

[12] K. D. Boese, A. B. Kahng, and G. Robins, "High-performance routing trees with identified critical sinks," in *Proc. Design Automation Conf.*, 1993, pp. 182–187.

[13] D. Zhou, F. Tsui, and D. S. Gao, "High performance multichip interconnection design," in *Proc. 4th ACM/SIGDA Physical Design Workshop*, Apr. 1993, pp. 32–43.

[14] J. Cong and C.-K. Koh, "Interconnect layout optimization under higher-order RLC model," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 713–720.

[15] J. Hu and S. S. Sapatnekar, "Simultaneous buffer insertion and non-Hanan optimization for VLSI interconnect under a higher order AWE model," in *Proc. Int. Symp. Physical Design*, 1999, pp. 133–138.

[16] J. Cong and K. S. Leung, "Optimal wiresizing under the distributed Elmore delay model," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 634–639.

[17] ——, "Optimal wiresizing under the distributed Elmore delay model," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 321–336, Mar. 1995.

[18] J. Cong and L. He, "Optimal wiresizing for interconnects with multiple sources," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 568–574.

[19] C. P. Chen, Y. W. Chang, and D. F. Wong, "Fast performance-driven optimization for buffered clock trees based on Lagrangian relaxation," in *Proc. Design Automation Conf.*, 1996, pp. 405–408.

[20] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 138–143.

[21] N. Menezes, S. Pullela, F. Dartu, and L. T. Pillage, "RC interconnect synthesis—A moment fitting approach," in *Proc. Int. Conf. Computer-Aided Design*, 1994, pp. 418–425.

[22] T. Xue, E. S. Kuh, and Q. Yu, "A sensitivity-based wiresizing approach to interconnect optimization of lossy transmission line topologies," in *Proc. IEEE Multi-Chip Module Conf.*, 1996, pp. 117–121.

[23] C.-P. Chen, H. Zhou, and D. F. Wong, "Optimal non-uniform wiresizing under the Elmore delay model," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 38–43.

[24] J. P. Fishburn, "Shaping a VLSI wire to minimize Elmore delay," in *Proc. Eur. Design Test Conf.*, 1997, pp. 244–251.

[25] J. Cong, L. He, C.-K. Koh, and Z. Pan, "Global interconnect sizing and spacing with consideration of coupling capacitance," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 628–633.

[26] J. Cong and L. He, "Theory and algorithm of local-refinement based optimization with application to device and interconnect sizing," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 406–420, Apr. 1999.

[27] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.
[28] J. Cong and C.-K. Koh, "Simultaneous driver and wire sizing for performance and power optimization," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 408–423, Dec. 1994.
[29] J. Cong, C.-K. Koh, and K.-S. Leung, "Simultaneous buffer and wire sizing for performance and power optimization," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1996, pp. 271–276.
[30] J. Cong and L. He, "An efficient approach to simultaneous transistor and interconnect sizing," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 181–186.
[31] C. C. N. Chu and D. F. Wong, "An efficient and optimal algorithm for simultaneous buffer and wire sizing," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1297–1304, Sept. 1999.
[32] N. Menezes, S. Pullela, and L. T. Pileggi, "Simultaneous gate and interconnect sizing for circuit-level delay optimization," in *Proc. Design Automation Conf.*, June 1995, pp. 690–695.
[33] N. Menezes, R. Baldick, and L. T. Pileggi, "A sequential quadratic programming approach to concurrent gate and wire sizing," in *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 144–151.
[34] C. C. N. Chu and D. F. Wong, "A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 787–798, June 1999.
[35] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 44–49.
[36] T. D. Hodes, B. A. McCoy, and G. Robins, "Dynamically-wiresized Elmore-based routing constructions," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1994, pp. 463–466.
[37] T. Xue and E. S. Kuh, "Post routing performance optimization via tapered link insertion and wiresizing," in *Proc. Eur. Design Automation Conf.*, 1995, pp. 74–79.
[38] J. Lillis, C. K. Cheng, T. T. Y. Lin, and C. Y. Ho, "New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing," in *Proc. Design Automation Conf.*, June 1996, pp. 395–400.
[39] J. Cong and D. Z. Pan, "Interconnect estimation and planning for deep submicron designs," in *Proc. Design Automation Conf.*, June 1999, pp. 507–510.
[40] C. J. Alpert, A. Devgan, and S. Quay, "Is wire tapering worthwhile?," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 430–435.
[41] J. Cong, L. He, A. B. Kahng, D. Noice, N. Shirali, and S. H.-C. Yen, "Analysis and justification of a simple, practical 2 1/2-D capacitance extraction methodology," in *Proc. ACM/IEEE Design Automation Conf.*, June 1997, pp. 40.1.1–40.1.6.
[42] J. Cong and D. Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 1999, pp. 97–100.
[43] C.-P. Chen and D. F. Wong, "Optimal wire sizing function with fringing capacitance consideration," in *Proc. Design Automation Conf.*, 1997, pp. 604–607.
[44] R. Otten, "Global wires harmful," in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 104–109.
[45] M. Pedram, N. Bhat, and E. Kuh, "Combining technology mapping and layout," *The VLSI Design: An Int. J. Custom-Chip Design, Simulation and Testing*, vol. 5, no. 2, pp. 111–124, 1997.
[46] J. Cong. (1997, Dec.) Challenges and opportunities for design innovations in nanometer technologies. *SRC Working Papers* [Online]. Available: http://www.src.org/prg_mgmt/frontier.dgw
[47] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Computer-Aided Design*, pp. 489–498, 1998.
[48] J. Cong, H. Li, and C. Wu, "Simultaneous circuit partitioning/clustering with retiming for performance optimization," in *Proc. Design Automation Conf.*, 1999, pp. 460–465.
[49] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. Design Automation Conf.*, 1997, pp. 526–529.
[50] J. Cong and S. K. Lim, "Edge separability based circuit clustering with application to circuit partitioning," in *Proc. Asia and South Pacific Design Automation Conf.*, 2000, pp. 429–434.
[51] ——, "Multiway partitioning with pairwise movement," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 512–516.
[52] A. Brandt, "Multi-level adaptive solution to boundary value problems," *Math. Comput.*, vol. 31, no. 138, pp. 333–390, 1977.
[53] W. Briggs, *A Multigrid Tutorial*: SIAM, 1987.
[54] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 69–79, Mar. 1999.
[55] T. Chan, J. Cong, T. Kong, and J. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 171–176.
[56] J. Cong and S. K. Lim, "Performance driven multiway partitioning," in *Proc. Asia and South Pacific Design Automation Conf.*, 2000, pp. 441–446.
[57] ——, "Physical planning with retiming," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 2–7.
[58] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 358–363.
[59] P. Sarkar, V. Sundararaman, and C.-K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," in *Proc. Int. Symp. Physical Design*, 2000.
[60] E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model," *Integr. VLSI J.*, vol. 5, pp. 3–16, 1987.
[61] J. Cong and P. H. Madden, "Performance driven multi-layer general area routing for PCB/MCM designs," in *Proc. Design Automation Conf.*, 1998, pp. 356–361.
[62] J. Cong and B. Preas, "A new algorithm for standard cell global routing," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1988, pp. 176–179.
[63] J. Cong and P. H. Madden, "High performance global routing for standard cell design," in *Proc. Int. Symp. Physical Design*, 1997, pp. 73–80.
[64] C.-K. Koh, "VLSI interconnect layout optimization," Ph.D. dissertation, 1998.
[65] C.-C. Chang and J. Cong, "Pseudo pin assignment with crosstalk noise control," in *Proc. Int. Symp. Physical Design*, 2000, pp. 41–47.
[66] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. 3, pp. 87–99, Jan. 1984.
[67] T. Stohr, M. Alt, A. Hetzel, and J. Koehl, "Analysis, reduction and avoidance of crosstalk on VLSI chips," in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 211–218.
[68] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *IEEE Int. Symp. Circuits Syst.*, May 1987, pp. 588–591.
[69] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 507–517, July 1987.
[70] L.-C. Liu, H.-P. Tseng, and C. Sechen, "Chip-level area routing," in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 197–204.
[71] T. Ohtsuki, "Gridless routers—New wire routing algorithms based on computational geometry," in *Proc. Int. Conf. Circuits and Systems*, 1985, pp. 802–809.
[72] W. Schiele, T. Kruger, K. Just, and F. Kirsch, "A gridless router for industrial design rules," in *Proc. Design Automation Conf.*, June 1990, pp. 626–631.
[73] J. Cong, J. Fang, and K. Khoo, "DUNE: A multi-layer gridless routing system with wire planning," in *Proc. Int. Symp. Physical Design*, Apr. 2000, pp. 12–18.
[74] ——, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 163–167.
[75] M. Arnold and W. Scott, "An interactive maze router with hints," in *Proc. 25th Design Automation Conf.*, June 1988, pp. 672–676.
[76] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "Magic: A VLSI layout system," in *Proc. 21st Design Automaton Conf.*, June 1984, pp. 152–159.
[77] D. Sylvester and K. Keutzer, "A global wiring paradigm for deep submicron design," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 242–252, Feb. 2000.
[78] I. E. Sutherland, R. F. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. New York: Academic, 1999.
[79] [Online]. Available: http://www.magma-da.com/c/@4sqgeyXVifzmg/Pages/Gain_based_overview.html

Jason Cong (Fellow, IEEE) received the B.S. degree in computer science from Peking University in 1985 and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign, in 1987 and 1990, respectively.

Currently, he is a Professor and Co-Director of the VLSI CAD Laboratory in the Computer Science Department of University of California, Los Angeles. His research interests include layout synthesis and logic synthesis for high-performance low-power VLSI circuits, design and optimization of high-speed VLSI interconnects, FPGA synthesis and reconfigurable architectures. He has published over 140 research papers and led over 20 research projects supported by DARPA, NSF, and a number of industrial sponsors in these areas. He served as the General Chair of the 1993 ACM/SIGDA Physical Design Workshop, the Program Chair and General Chair of the 1997 and 1998 International Symposium on FPGAs, respectively, Program Co-Chair of the 1999 International Symposium on Low-Power Electronics and Designs, and on program committees of many major conferences, including DAC, ICCAD, and ISCAS.

Dr. Cong received the Best Graduate Award from the Peking University in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois at Urbana-Champaign in 1989. He received the NSF Young Investigator Award in 1993, the Northrop Outstanding Junior Faculty Research Award from UCLA in 1993, the IEEE TRANSACTIONS ON CAD Best Paper Award in 1995 from IEEE CAS Society, the ACM SIGDA Meritorious Service Award in 1998, and an SRC Inventor Recognition Award in 2000. He was appointed as a Guest Professor of Peking University since 2000. He is an Associate Editor of IEEE TRANSACTIONS ON VLSI SYSTEMS and *ACM Transactions on Design Automation of Electronic Systems*.