BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP

Minsik Cho
ECE Dept. Univ. of Texas at Austin
Austin, TX 78712
thyeros@cerc.utexas.edu

David Z. Pan
ECE Dept. Univ. of Texas at Austin
Austin, TX 78712
dpan@ece.utexas.edu

ABSTRACT

In this paper, we propose a new global router, BoxRouter, powered by the concept of box expansion and progressive integer linear programming (ILP). BoxRouter first uses a simple PreRouting strategy which can predict and capture the most congested regions with high fidelity compared to the final routing. Based on progressive box expansion initiated from the most congested region, BoxRouting is performed with progressive ILP and adaptive maze routing. It is followed by an effective PostRouting step which reroutes without rip-up to obtain smooth tradeoff between wirelength and routability. Our experimental results show that BoxRouter significantly outperforms the state-of-the-art published global routers, e.g., 79% better routability than [1] (with similar wirelength and 2x speedup), 4.2% less wirelength and 16x speedup than [2] (with similar routability). Given the fundamental importance of routing, such dramatic improvement shall sparkle renewed interests in routing which plays a key role in nanometer design and manufacturing closure.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

General Terms

Algorithms, Design, Performance

Keywords

VLSI, Global Routing, Congestion

1. INTRODUCTION

Routing is a key stage for VLSI physical design. Aggressive technology scaling has led to much smaller/faster devices, but more resistive interconnects and larger coupling capacitance. Since routing directly determines the interconnects (wirelength, routability/congestion, and so on), thus the overall VLSI system performance [5, 15, 25], it plays a critical role in the deep-submicron *design closure*. For nanometer interconnects, the manufacturability and variability issues such as antenna effect, copper chemical mechanical polishing (CMP) and subwavelength printability have also emerged [6, 13, 18, 20, 25, 26]. Again, routing plays a major role in terms of the *manufacturing closure*.

The global routing, as its name implies, is the routing stage that plans the approximate routing path of each net to reduce the complexity of routing task and guide the detailed router [12]. Thus, it has significant impact on wirelength, routability and timing [12] [11]. Optimizing the wire density distribution during the global routing can also help to improve the overall manufacturability (e.g., less post-CMP topography variation, less copper erosion/dishing, and less optical interference for better printability [6, 13, 18, 20]). Furthermore, fast global routing can feed more accurate interconnect information (such as wirelength and congestion) back to placement or other early physical synthesis engines for better design convergence [24].

The importance of global routing in VLSI design flow has led to many works in predicting and estimating routing congestion, and designing global routers. Probability based congestion prediction for global routing is studied in [14, 17, 19, 22], and global router based congestion estimation is researched in [21] [23] for early wirelength estimation. Within the scope of over-the-cell global routing model [1], Burstein et al. [3] proposed a hierarchical approach to speed up integer programming formulation for global routing, and Kastner [16] proposed a pattern-based global routing. Raja et al. [1] presented the *Chi* dispersion router based on linear cost function, and showed better results than [16]. The multicommodity flow-based global router by Albrecht [2] showed good results and was used in industry, but at the expense of computational effort. No comparison, however, is available between [1] and [2], which are two state-of-the-art published global routers.

In this paper, we propose BoxRouter, a new global router based on the idea of box expansion and progressive ILP. Essentially, BoxRouter progressively expands the routing box and performs routing within each expanded box (BoxRouting), until the expanded box covers the whole circuit (all the wires are routed). Efficient ILP is formulated to perform BoxRouting, with the aid of effective PreRouting and PostRouting. The major contributions of this paper include the following.

- We observe that a simple PreRouting step can capture the most congested regions with reasonable accuracy, and improve runtime by reducing the number of wires to be routed later.

This work is supported in part by SRC, IBM Faculty Award, Fujitsu, Sun and equipment donations from Intel.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

- We propose the key BoxRouting idea which efficiently utilizes limited routing resources based on progressive box expansion initiated from the most congested region estimated by PreRouting. Using different routing strategies inside and outside the box, routability can be maximized with minimum wirelength increase.
- We propose an efficient progressive integer linear programming (ILP) for BoxRouting. In our ILP, only wires between two successive boxes are considered with L-shape patterns. Thus even with ILP, our runtime is still much faster than existing global routers [1] [2] [16].
- We propose an effective PostRouting step which *reroutes* wires from the most congested region *without rip-up*. It is more efficient than the conventional rip-up & route. It also provides smooth trade-off between wirelength and routability with only a simple parameter.

BoxRouter achieves impressively better results on the standard ISPD98 IBM benchmarks than [1] and [2], thus pushes the state-of-the-art considerably. Due to the fundamental importance of global routing, we believe it shall have many applications/implications for nanometer designs.

The rest of the paper is organized as follows. In Section 2, preliminaries are described. In Section 3, BoxRouter is proposed. Experimental results are discussed in Section 4, followed by the conclusion in Section 5.

2. PRELIMINARIES

2.1 Notations

Table 1 lists the notations used throughout this paper.

Table 1: The notations in this paper.

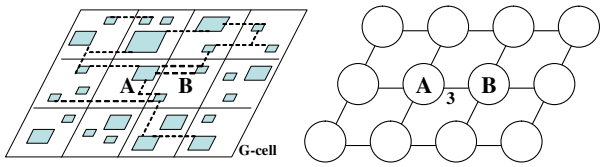
v_i	vertex / global routing cell i
e_{ij}	edge between v_i and v_j
m_{ij}	maximum routing resource of e_{ij}
c_{ij}	available routing resource of e_{ij}

2.2 Global Routing Model

The global routing problem can be modelled as a grid graph $G(V, E)$, where each vertex v_i represents a rectangular region of the chip, so called a global routing cell (G-cell), and an edge e_{ij} represents the boundary between v_i and v_j with a given maximum routing resource m_{ij}. Fig. 1 shows how the chip can be abstracted into a grid graph where $m_{AB} = 3$. A global routing is to find paths that connect the pins inside the G-cells through $G(V, E)$ for every net.

2.3 Global Routing Metrics

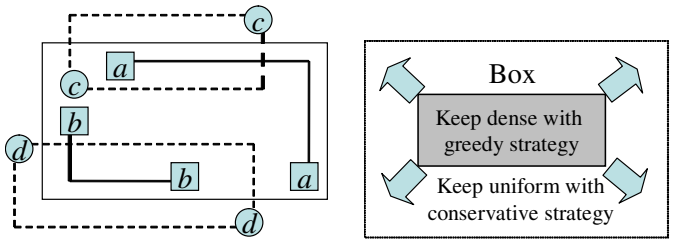
The key task of global router is to maximize the routability for successful detailed routing [24]. In addition, wirelength and runtime are other important metrics of global router.



(a) real circuit with G-cells (b) grid graph for routing

Figure 1: Grid graph model for global routing

- **Routability** can be estimated by the number of *overflow*s which indicates that routing demand locally exceeds the available routing capacity [24] [16]. In Fig. 1, the number of overflow between v_A and v_B is one, as there are four routed nets, but $m_{AB} = 3$. Formal definition of overflow is in [16].
- **Wirelength** is an important metric for placement as well as routing. But, it is less concern for global routing, as routing all wires with shortest path algorithms will result in minimum or near-minimum wirelength [24]. However, there can be huge difference between solutions of the same wirelength in terms of routability.
- **Runtime** is fairly significant, as global routing links placement and detailed routing, and needs to feed parasitic information to higher level of design flow for design convergence.



(a) motivation for BoxRouting (b) strategies of BoxRouting

Figure 2: The basic concept of BoxRouter

3. BOXROUTER

In this section, we present a new global router, BoxRouter, which is based on congestion-initiated box expansion. Our BoxRouter progressively expands a box which initially covers the most congested region only, but finally covers the whole circuit. After every expansion, a circuit is divided into two sections, inside the box and outside the box. BoxRouter uses different routing strategies for each section to maximize routability and minimize wirelength. Consider Fig. 2 (a), where two wires (a and b) are inside the box, but the other wires (c and d) are *not* inside the box. The routing resource inside the box is more precious to a and b than c and d for two reasons:

- If a or b are not routed within the box, wirelength will be increased due to detour.
- c and d may have another viable routing path outside box which does not waste the routing resource inside the box.

Therefore, BoxRouter first routes as many wires inside the box as possible with progressive integer linear programming (ILP) routing, by maximally utilizing the routing resource inside the box. Then, for the wires which cannot be routed by progressive ILP within the box (due to insufficient routing resources), BoxRouter detours them by adaptive maze routing with the following two strategies:

- **Inside the box**, use the routing resources as much as possible (greedily), as the wires inside the box have priority over those outside the box.

- **Outside the box**, use the routing resources conservatively, as the wires outside the box may need them later for their routing paths.

Those two strategies make the wire density of the circuit as in Fig. 2 (b), and help the wires detour the more congested region to maximize the routability with minimum detour.

The overall flow of BoxRouter is in Fig. 3, which will be explained in detail in the rest of this section. Section 3.1 describes the preprocessing for BoxRouter. Section 3.2 illustrates PreRouting for congestion estimation and routing speedup. Section 3.3 explains BoxRouting, the main idea of BoxRouter. Finally, Section 3.4 shows how to control the trade-off between wirelength and routability in PostRouting.

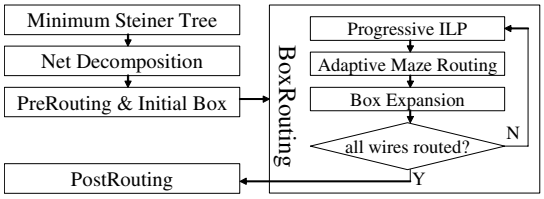
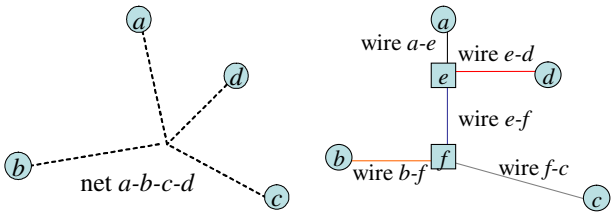


Figure 3: BoxRouter overall flow

3.1 Steiner Tree and Net Decomposition

A net can be decomposed into two pin *wires* with Rectilinear Minimum Seiner Tree as shown in Fig. 4. A fast and accurate steiner tree algorithm, Flute [4] is adopted in BoxRouter. But, any other steiner tree algorithm can be used for BoxRouter without significantly impacting the result . A special wire which does not need a bend is called a *flat* wire [17]. For example, wire *a*-*e*, *e*-*d*, *e*-*f* and *b*-*f* in Fig. 4 (b) are flat wires.

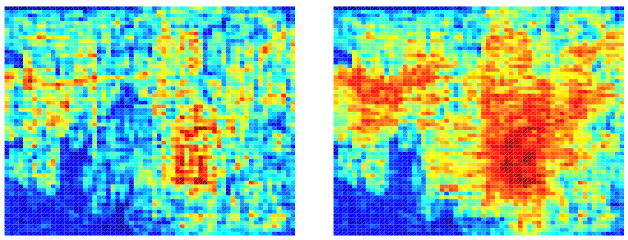


(a) hypergraph for a net (b) wires after decomposition

Figure 4: Net decomposition into two pin wires

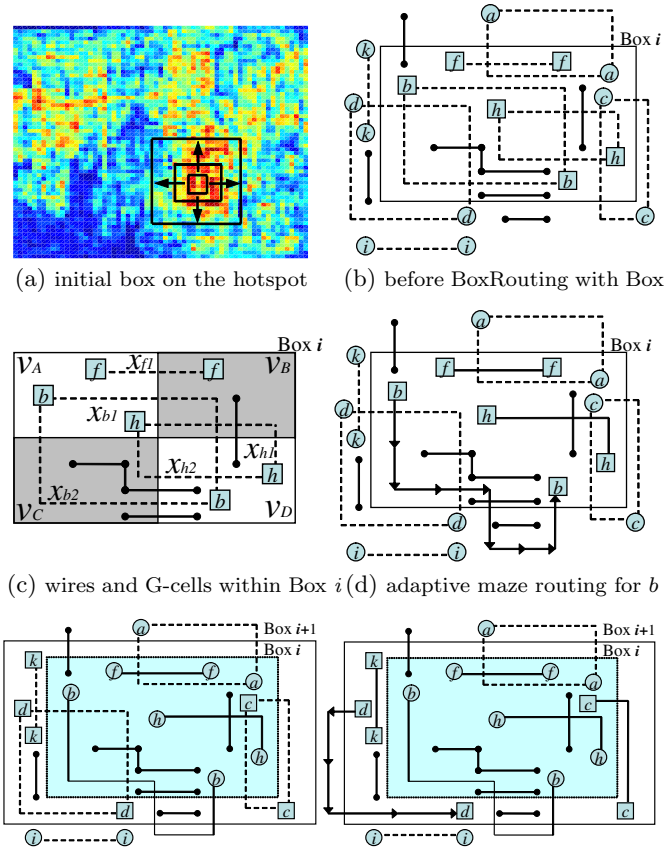
3.2 PreRouting and Initial Box

PreRouting simply routes as many flat wires as possible via the shortest path *without* creating any overflow. As



(a) congestion after PreRouting (b) congestion after BoxRouting

Figure 5: Congestion estimation from PreRouting



(a) initial box on the hotspot (b) before BoxRouting with Box i

(c) wires and G-cells within Box i (d) adaptive maze routing for b

(e) box expansion to Box i+1 (f) BoxRouting with Box i+1

Figure 6: BoxRouting example

bulk of nets can be routed in simple patterns (L-shape or Z-shape) [22] [23] [16], PreRouting can improve the runtime without degrading the final solution. More importantly, if enough number of wires can be routed by PreRouting, the global congestion view can be obtained. According to our experiments, about 60% of the final wirelength on average can be routed with tiny computational overhead by PreRouting. Fig. 5 shows an example of congestion maps after PreRouting and BoxRouting. The most congested regions (hotspots) can be predicted very well by PreRouting. A box which encompasses the four G-cells in the most congested area will be created as shown in Fig. 6 (a) as a starting point of BoxRouting.

3.3 BoxRouting

In this subsection, BoxRouting will be explained with Fig. 6. BoxRouting consists of three steps, progressive integer linear programming (ILP) routing, adaptive maze routing and box expansion as in Fig. 3. Those three steps are repeated until the expanded box covers the whole circuit.

Assuming a box is expanded from the most congested region as in Fig. 6 (a), consider Fig. 6 (b), where wires within the box after i-th expansion (Box i) are shown with squares (b, f and h), and the other wires are shown with circles. The already routed wires by either PreRouting or previous BoxRouting are simply shown as solid lines. Note that some flat wires like f, i and k remain unrouted until BoxRouting, if PreRouting gives up routing a wire due to any overflow or new steiner points introduced by adaptive maze routing

$$
\begin{aligned}
\textbf{max :} \quad & x_{b1} + x_{b2} + x_{f1} + x_{f2} + x_{h1} + x_{h2} \\
\textbf{s.t :} \quad & x_{b1}, x_{b2}, x_{f1}, x_{f2}, x_{h1}, x_{h2} \in \{0, 1\} \\
& x_{b1} + x_{b2} \leq 1 \\
& x_{f1} \leq 1, x_{f2} = 0 \\
& x_{h1} + x_{h2} \leq 1 \\
& x_{b1} + x_{f1} + x_{h1} \leq c_{AB} \\
& x_{b1} + x_{h1} \leq c_{BD} \\
& x_{b2} + x_{h2} \leq c_{AC} \\
& x_{b2} + x_{h2} \leq c_{CD}
\end{aligned}
$$

Figure 7: Progressive ILP formulation of Fig. 6 (c)

$$
\begin{aligned}
\textbf{max :} \quad & \textstyle\sum \{x_{i1} + x_{i2}\} && \forall i \in W_{box} \\
\textbf{s.t :} \quad & x_{i1}, x_{i2} \in \{0, 1\} && \forall i \in W_{box} \\
& x_{i1} + x_{i2} \leq 1 && \forall i \in W_{box} \\
& x_{i2} = 0 && \forall i \in W_{box} \cap W_{flat} \\
& \textstyle\sum_{e \in x_{i,j}} x_{ij} \leq c_e && \forall e \in W_{box}
\end{aligned}
$$

Figure 8: General progressive ILP formulation

(explained later in this section) convert a non-flat wire into a flat wire. For efficient routing as mentioned in the beginning of this section, only wires within the box will be routed by progressive ILP and adaptive maze routing.

In Fig. 6 (c), the wires within the box are shown with G-cells (v_A, v_B, v_C and v_D), and the corresponding progressive ILP formulation for maximum routability is shown in Fig. 7. To minimize the number of vias, two L-shape routings (x_{b1}, x_{b2} and x_{h1}, x_{h2}) are considered for each wire in our ILP formulation, but only one routing (x_{f1} and x_{f2}=0) is considered for flat wires. General progressive ILP formulation is shown in Fig. 8, where W_{box} is a set of unrouted wires within the current box and W_{flat} is a set of flat wires.

Differently from the hierarchical ILP [3], our ILP approach progressively routes a part of the circuit, which is covered by each expanding box. Also, as the solution from Box i is reflected in the next routing problem of Box i+1 (Box i+1 always encompasses Box i), our progressive ILP approach provides a seamless and incremental routing. Even though, the last box can cover the whole circuit, the ILP size remains tractable, as ILP is performed on the wires between two boxes like between Box i and Box i+1 in Fig. 6 (e).

However, due to the limited routing resource of each edge, some wires may not be routed with ILP. For example, assuming m_{CD}=2, the wire b cannot be routed with the ILP (x_{b1}=x_{b2}=0), as two prerouted wires on e_{CD} consume all the routing resources. For this case, b is routed by adaptive maze routing as in Fig. 6 (d) with the cost from Alg. 1.

Alg. 1 returns a unit cost as long as e_{XY} is inside box and still has available routing resource (line 2, 3). Otherwise, it returns a cost inversely proportional to the available routing resources (line 1). This cost function makes adaptive maze routing find the shortest path inside the box for wirelength minimization, but the most idle path outside the box for routability maximization. Note that the resource outside the box should be used conservatively, as the wires outside the current box may need them later. If too big detour is required to avoid small overflows, adaptive maze routing which looks for the minimum cost path may return a routing path with overflows.

After all the wires inside the box are routed either by progressive ILP or adaptive maze routing, the Box i will be expanded to Box i+1, and new wires (c, d and k) are encompassed by Box i+1 as shown in Fig. 6 (e). The result after applying BoxRouting (progressive ILP and adaptive maze routing) again is shown in Fig. 6 (f). The amount of increment during box expansion significantly affects the routing solution. As the box grows larger for every expansion with bigger increment, the runtime increases exponentially due to larger ILP problem size (more wires are added into the formulation due to larger expansion). But, the smaller overflow can be obtained, as the routing is performed more globally. More discussion is presented in Section 4. After all wires are routed (the box becomes big enough to cover the whole circuit), PostRouting of Section 3.4 will follow BoxRouting.

The intuition of BoxRouting is that it mimics the diffusion effect. By each BoxRouting, all the wires in the more congested region (within the box) are routed first by progressive ILP, then by adaptive maze routing. This makes the wires outside the box detour the box, as there is low chance of being routed through the box. Such detouring wires works like diffusing wires to the low congested area, improving routability at a cost of wirelength.

3.4 PostRouting (Reroute without Rip-up)

As the adaptive maze routing of BoxRouting uses conservative strategy outside the box in Alg. 1 (finding the most idle routing path outside the box), it may create unnecessary detour and overflow. Thus, PostRouting simply reroutes wires to remove unnecessary overhead with box expansion initiated from the most congested region, as done in BoxRouting. In detail, a wire in the more congested region will be rerouted first, and such rerouted wire can release the routing resource, as it may find the better routing path. Then, the surrounding wires can be rerouted with the released routing resource, potentially reducing detour and overflow again. This chain reaction propagates from the most congested region to less congested regions along the box expansion.

Maze routing is used for PostRouting, but with a different routing cost function as in Alg. 2, where parameter K is introduced. The parameter K controls the trade-off between wirelength and routability (overflow), by setting the cost of each overflow as K. Thus, higher K will discourage overflow

Algorithm 1 Adaptive Maze Routing Cost for BoxRouting

Input: G-Cell X, Y, Box B
1: Cost $C = m_{XY} - c_{XY}$
2: **if** e_{XY} is inside B and $c_{XY} > 0$ **then**
3: $C = 1$
4: **end if**
Output: C

Algorithm 2 Maze Routing Cost for PostRouting

Input: G-Cell X, Y, Param K
1: Cost $C = K$
2: **if** $c_{XY} > 0$ **then**
3: $C = 1$
4: **end if**
Output: C

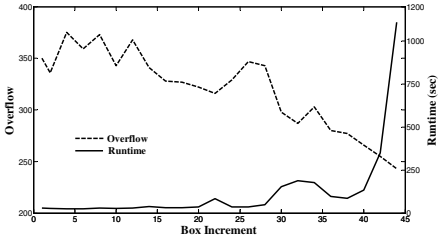


Figure 9: Overflow and runtime change by box increment for ibm04

at a cost of wirelength increase (more detours), but lower K will suppress detour at a cost of overflows. The effectiveness of parameter K is discussed in Section 4.

Our PostRouting is more efficient than the widely used Rip-up&Reroute (R&R), as PostRouting makes a wire *voluntarily* release a routing resource (this happens, only when the solution improves) during its rerouting, while R&R deprives it from a wire in the congested region without guaranteeing any improvement.

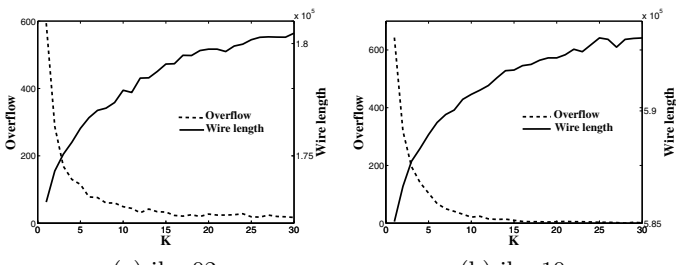
4. EXPERIMENTAL RESULTS

We implement BoxRouter in C++, and the ISPD98 IBM benchmarks are taken from [9]. All the experiments are performed on a 2.8 GHz Pentinum-4 Linux machine. Flute [4] is used for steiner tree construction, and GNU Linear Programming Kit (GLPK) 4.8 [10] is used as ILP solver.

Fig. 9 shows the overflow and runtime by the amount of box increment (See Section 3.3) for one benchmark. It clearly shows that with larger box increment, the overflow decreases, but the runtime increases exponentially. While the wirelength varies only by 0.11%, the overflow decreases by 30%, but the runtime increases by 500%. It indicates that with larger box increment during box expansion of BoxRouting, the solution quality can be improved at a cost of runtime (Note that ILP takes the majority of the runtime). However, the amount of box increment is kept sufficiently small for all the experiments in this section, as GLPK 4.8 is unstable for large ILP problems.

The effectiveness of parameter K (See Section 3.4) is shown in Fig. 10. It shows that with larger K, overflow decreases exponentially, but wirelength increases. We constantly find that overflow saturates faster than wirelength, and the best trade-off occurs around K=15 for all the tested benchmarks.

Table 2 shows the detail about benchmarks and the routing result by BoxRouter with K=15. The lower bound wirelength of each circuit is computed by the most accurate GeoSteiner 3.1 [4] [8]. It shows that BoxRouter has 3.7% wirelength overhead on average, and provides high quality solution for larger circuits with small overflows. Also, we observe that BoxRouter has linear memory complexity for all the tested benchmarks.



(a) ibm02 (b) ibm10

Figure 10: Routability and wirelength trade-off by Parameter K

Table 2: Routing Results from BoxRouter for ISPD98 IBM benchmarks.

circuit					BoxRouter (K=15)		
name	nets	wires	grids	lb.wlen[a]	wlen[b]	ovfl[c]	w.o(%)[d]
ibm01	11507	28232	64x64	60142	65193	126	8.4
ibm02	18429	55649	80x64	165863	179086	33	8.0
ibm03	21621	45727	80x64	145678	149879	9	2.9
ibm04	26163	53487	96x64	162734	171756	342	5.5
ibm05	27777	94304	128x64	409709	409747	0	0
ibm06	33354	82541	128x64	275868	282002	5	2.2
ibm07	44394	109365	192x64	363537	376247	81	3.5
ibm08	47944	133353	192x64	402412	409584	31	1.8
ibm09	50393	128708	256x64	411260	418023	4	1.6
ibm10	64227	182010	256x64	574407	591820	10	3.0

[a] asymptotic lower bound of wirelength by GeoSteiner 3.1
[b] wirelength hereafter in this section.
[c] overflow hereafter in this section.
[d] wirelength overhead.

For thorough comparison, we download two available global routers, Labyrinth 1.1 [16] [9] and Fengshui 5.1 (which has the newest implementation of the *Chi* dispersion router) [1] [7], and implement multicommodity flow-based global router in C++ (the binary is not available from the author) [2]. Although the results of Labyrinth and Fengshui are reported in [1], we reproduce the results due to the recent update in the benchmarks [9].

Table 3 shows the experimental results and comparison for Labyrinth, Fengshui, and Table 4 shows for multicommodity flow-based router. As there is a trade-off between wirelength and routability, we choose the parameter K of BoxRouter of Table 3 *with* wirelength constraint such that wirelength from BoxRouter is as small or smaller than those from Labyrinth and Fengshui for fair comparison. Regarding Table 4, we carefully choose the parameters of multicommodity router for each benchmark such the best results are yielded within 25 phases (the maximum phase in [2]), and simulate ibm01, ibm02, ibm04 and ibm07 (circuits with non-zero overflow in Table 3) again for BoxRouter *without* any constraint.

As in Table 3, BoxRouter outperforms Labyrinth and Fengshui by wide margin. In terms of wirelength and overflow, BoxRouter can reduce the wirelength by 14.3%, the overflow by 91.7% compared with Labyrinth, and improve the overflow by 79% with similar wirelength (actually 0.8% better) compared with Fengshui. Also, BoxRouter is 3.3x and 2.0x faster than Labyrinth and Fengshui respectively. Mutlicommodity flow-based router and BoxRouter show very comparable overflow as shown in Table 4. However, BoxRouter is 15.7x faster, and produces 4.2% shorter wirelength on average than multicommodity flow-based router. It shows that BoxRouter can provide high quality global routing solution with significantly less design turn around time.

5. CONCLUSION

In order to cope with the increasing impact of interconnect on system performance, we present an efficient global router, BoxRouter to maximize the routability with minimum wirelength. Experimental results show that BoxRouter outper-

Table 3: Comparison with Labyrinth 1.1 and Fengshui 5.1 (*Chi* dispersion) for ISPD98 IBM benchmarks.

circuit	Labyrinth 1.1			Fengshui 5.1			BoxRouter			Imprv. on Labyrinth			Imprv. on Fengshui		
name	wlen	ovfl	cpu(s)	wlen	ovfl	cpu(s)	wlen	ovfl	cpu(s)	wlen(%)	ovfl(%)	spd(x)[a]	wlen(%)	ovfl(%)	spd(x)[a]
ibm01	76517	398	21.2	66006	189	15.1	65588	102	8.3	14.3	74.4	2.5	0.6	46.0	1.8
ibm02	204734	492	34.5	178892	64	47.9	178759	33	34.1	12.7	93.3	1.0	0.1	48.4	1.4
ibm03	185116	209	36.3	152392	10	35.2	151299	0	16.9	18.3	100	2.1	0.7	100	2.1
ibm04	196920	882	83.5	173241	465	54.1	173289	309	23.9	12.0	65.0	3.5	0.0	33.5	2.3
ibm05[b]	420583	0	59.2	412197	0	104.8	409747	0	49.5	-	-	-	-	-	-
ibm06	346137	834	104.3	289276	35	80.1	282325	0	33.0	18.4	100	3.2	2.4	100	2.4
ibm07	449213	697	228.1	378994	309	122.2	378876	53	50.9	15.7	92.4	4.5	0.0	82.8	2.4
ibm08	469666	665	238.7	415285	74	113.8	415025	0	93.2	11.6	100	2.6	0.1	100	1.2
ibm09	481176	505	359.3	427556	52	125.1	418615	0	63.9	13.0	100	5.6	2.1	100	2.0
ibm10	679606	588	435.7	599937	51	212.9	593186	0	95.1	12.7	100	4.6	1.1	100	2.2
							average			14.3	91.7	3.3	0.8	79.0	2.0

[a] speedup hereafter in this section.
[b] ibm05 is dropped from comparison hereafter in this section, as it is a trivial case.

Table 4: Comparison with multicommodity flow-based router for ISPD98 IBM benchmarks.

circuit	Multicommodity			BoxRouter			Imprv.[a]	
name	wlen	ovfl	cpu(s)	wlen	ovfl	cpu(s)	wlen(%)	spd(x)
ibm01	68981	43	151.2	67674	41	11.8	1.9	12.8
ibm02	190418	3	494.5	182268	2	35.7	4.3	13.9
ibm03	160755	0	329.8	151299	0	16.9	5.9	19.5
ibm04	176610	225	326.6	173778	249	31.4	1.6	10.4
ibm05	410954	0	28.2[b]	409747	0	49.5	-	-
ibm06	296981	0	951.8	282325	0	33.0	4.9	28.9
ibm07	408510	0	1229.0	394170	0	50.8	3.5	24.2
ibm08	429913	0	865.7	415025	0	93.2	3.5	9.3
ibm09	442514	0	726.7	418615	0	63.9	5.4	11.4
ibm10	634247	0	1068.4	593186	0	95.1	6.5	11.2
				average			4.2	15.7

[a] overflow is not shown, as both are highly comparable.
[b] only one phase is required for ibm05, a trivial case.

forms the state-of-the-art publicly available global routers in terms of wirelength, routability and runtime. As the BoxRouter is still in beta version, we believe that further improvement can be achieved with multiple box-expansions, faster ILP solver, and so on. We plan to make BoxRouter code with OpenAccess interface [27] available [28] to sparkle more research on this fundamental topic for nanometer design and manufacturing closure.

6. ACKNOWLEDGMENT

The authors would like to thank Prof. Patrick Madden from SUNY Binghamton and Dr. Christoph Albrecht from Cadence Berkeley Lab for helpful discussions.

7. REFERENCES

[1] R. T. Hadsell and P. H. Madden. Improved Global Routing through Congestion Estimation. In *Proc. Design Automation Conf.*, June 2003.
[2] C. Albrecht. Global Routing by New Approximation Algorithms for Multicommodity Flow. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20, May 2001.
[3] M. Burstein and R. Pelavin. Hierarchial Global Wiring for Custom Chip Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2(4), Oct 1983.
[4] C. C. N. Chu. FLUTE: Fast Lookup Table Based Wirelength Estimation Technique. In *Proc. Int. Conf. on Computer Aided Design*, 2004.
[5] J. Cong. Challenges and opportunities for design innovations in nanometer technologies. In *SRC Design Science Concept Papers*, 1997.
[6] T. E. Gbondo-Tugbawa. Chip-Scale Modeling of Pattern Dependencies in Copper Chemical Mechanical Polishing Process. In *Ph.D. Thesis, Massachusetts Institute of Technology*, 2002.
[7] http://vlsicad.cs.binghamton.edu/.
[8] http://www.diku.dk/geosteiner/.
[9] http://www.ece.ucsb.edu/∼kastner/labyrinth/.
[10] http://www.gnu.org/software/glpk/glpk.html/.
[11] J. Hu and S. Sapatnekar. A Timing-Constrained Algorithm for Simultaneous Global Routing of Multimple Nets. In *Proc. Int. Conf. on Computer Aided Design*, 2000.
[12] J. Hu and S. Sapatnekar. A Survey On Multi-net Global Routing for Integrated Circuits. *Integration, the VLSI Journal*, 31, 2002.
[13] L. Huang and D. F. Wong. Optical Proximity Correction (OPC)-Friendly Maze Routing. In *Proc. Design Automation Conf.*, June 2004.
[14] A. B. Kahng and X.Xu. Accurate Pseudo-Constructive Wirelength and Congestion Estimation. In *Proc. System-Level Interconnect Prediction*, April 2003.
[15] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. An Exact Algorithm for Coupling-Free Routing. In *Proc. Int. Symp. on Physical Design*, April 2001.
[16] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21, July 2002.
[17] J. Lou, S. Krishnamoorthy, and H. S. Sheng. Estimating Routing Congestion using Probabilistic Analysis. In *Proc. Int. Symp. on Physical Design*, 2001.
[18] J. Mitra, P. Yu, and D. Z. Pan. RADAR: RET-Aware Detailed Routing Using Fast Lithography Simulations. In *Proc. Design Automation Conf.*, June 2005.
[19] C. Sham and E. F. Y. Young. Congestion Prediction in Early Stages. In *Proc. System-Level Interconnect Prediction*, April 2005.
[20] R. Tian, D. F. Wong, and R. Boone. Model-Based Dummy Feature Placement for Oxide Chemical-Mechanical Polishing Manufacturability. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20, 2001.
[21] M. Wang, , and M. Sarrafzadeh. Modeling and Minimization of Routing Congestion. In *Proc. Design Automation Conf.*, 2000.
[22] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic Congestion Prediction. In *Proc. Int. Symp. on Physical Design*, April 2004.
[23] J. Westra, C. Bartels, and P. Groeneveld. Is Probabilistic Congestion Estimation Worthwhile? In *Proc. System-Level Interconnect Prediction*, April 2005.
[24] J. Westra, P. Groeneveld, T. Yan, and P. H. Madden. Global Routing: Metrics, Benchmarks, and Tools. In *IEEE DATC Electronic Design Process*, April 2005.
[25] D. Wu, J. Hu, and R. Mahapatra. Coupling Aware Timing Optimization and Antenna Avoidance in Layer Assignment. In *Proc. Int. Symp. on Physical Design*, April 2005.
[26] G. Xu, L. Huang, D. Z. Pan, and D. F. Wong. Redundant-Via Enhanced Maze Routing for Yield Improvement. In *Proc. Asia and South Pacific Design Automation Conf.*, Jan 2005.
[27] http://openeda.si2.org.
[28] http://www.cerc.utexas.edu/utda.