Processing Rate Optimization by Sequential System Floorplanning *

Jia Wang† Ping-Chih Wu‡ Hai Zhou†

†EECS Department
Northwestern University
Evanston, IL 60208, U.S.A.
{jwa112, haizhou}@ece.northwestern.edu

‡Cadence Design Systems Inc.
555 River Oaks Parkway
San Jose, CA 95134, U.S.A.
pcwu@cadence.com

Abstract

The performance of a sequential system is usually measured by its frequency. However, with the appearance of global interconnects that require multiple clock periods to communicate, the throughput is usually traded-off for higher frequency (for example, through wire pipelining or latency insensitive design). Therefore, we propose to use the processing rate, defined as the amount of processed inputs per unit time, as the performance measure. We show that the minimal ratio of the flip-flop number over the delay on any cycle is an upper bound of the processing rate. Since the processing rate of a sequential system is mainly decided by its floorplan when interconnect delays are dominant, the problem of floorplanning for processing rate optimization is formulated and solved. We optimize the processing rate bound directly in a floorplanner by applying Howard's algorithm incrementally. Experimental results confirm the effectiveness of our approach.

1. Introduction

The performance of a sequential system is usually measured by its frequency, or equivalently, its clock period. Sequential optimizations such as minimal period retiming [1] and clock skew scheduling [2] can be used to optimize the clock period by balancing the combinational path delay between consecutive flip-flops. When interconnect delays begin to dominate the performance because of the aggressive scaling down of geometries in Deep Sub-Micron (DSM) VLSI technology, things become much more complicated. Unlike gate delays, interconnect delays are only available until very late in the current design flow and always after floorplanning and placement. Different floorplans or placements give different minimal clock periods after sequential optimizations. Ignoring such optimization possibilities in the design flow may result in sub-optimal solutions. The work [9] addressed this problem by considering the optimization potential in physical placement. In their work, the maximum ratio of the delay over the flip-flop number along any cycle in the circuit is known to bound the minimal clock period that can be achieved through sequential optimizations. To optimize the ratio in the placement, they identified the cycle with the maximum ratio in the current placement and used it as a constraint to find a new placement with a smaller maximum ratio.

When the operating frequency of the sequential system is given, it is possible that one clock period is too small to propagate a signal from one end of a long wire to the other end in one clock period. In this case, wire pipelining is vital to allow multi-cycle communication over a long wire. As suggested in [4], retiming can be used to pipeline the long wires. However, if the frequency is higher than the best one that could be achieved by retiming, more wire-pipelining units like flip-flops must be introduced on those long wires to pipeline them. The side effect is that these additional units may change the latency of some parts of the circuit so that the functionality is different from the original one. Both the latency insensitive design (LIS) [6, 7] and the wire-pipelining correcting method [5] addressed this problem. In these two approaches, the throughput, which is defined as the amount of the processed inputs per clock cycle, is traded for higher frequency and is no longer one as in retiming or clock skew scheduling. In both cases, the throughput is bounded by the minimum ratio of the flip-flop number to the number of the required wire-pipelining unit along any cycle. The work [8] optimized the throughput bound in LIS with floorplanning. They used a heuristic throughput evaluation method in the simulated annealing (SA) floorplanner Parquet [13] based on an assumption that an exact throughput evaluation is too time-consuming in SA. The heuristic method estimates the throughout by assigning different weights to different wires according to their contributions to the throughput.

We find that by looking at the bound of the processing rate, we can unify the situations where the throughput is fixed and the frequency is optimized and where the frequency is fixed and the throughput is optimized. In any case, the processing rate is bounded by the minimum ratio of the flip-flop number over the delay along any cycle. Since this bound is independent of the operating frequencies and the afterward optimization/wire pipelining methodologies, it is more general than the clock period bound or the throughput bound.

We optimize the processing rate bound directly in a SA

*This work is supported in part by the NSF under award CCR-0238484 and a gift from Cadence.

based floorplanner. Unlike the previous approaches where the bounds were handled indirectly, Howard's algorithm [15, 14] is applied to compute the bound exactly inside the inner loop. The resulted floorplans are evaluated under different frequencies to obtain the throughputs and the processing rates. To show that our approach achieves better solution quality and running time, we apply our algorithm to GSRC benchmarks as in [8] and compare our results with theirs. We need to point out that the results in [8] are optimized under different frequencies separately so our one-floorplan-fit-all approach is more universal and less time consuming.

The *Adjacent Constraint Graph* (ACG) [10, 11] is used as our floorplan representation. It preserves the geometrical adjacency information and its operations map to local perturbations in physical space. To exploit those local perturbations, we apply Howard's algorithm incrementally, which speeds up the floorplanner by 29% on average. In addition, we show that addressing the fixed-outline constraint explicitly in the cost function is possible and effective in the ACG representation.

The rest of this paper is organized as follows. In Section 2, we show how the minimal cycle ratio bounds the processing rate of a sequential system and formulate the *Floorplanning for Processing Rate* (FPR) problem. The SA based floorplanner incorporating processing rate optimization is presented in Section 3. Experimental results are shown in Section 4. Finally, Section 5 concludes the paper.

2. Processing Rate and Floorplan Problem

2.1. Processing Rate Bound

We define the processing rate of a sequential system as follows.

Definition 1 (Processing Rate) *In a sequential system, the processing rate is defined as the length of processed input sequence per unit time.*

Considering the frequency and the throughput in a synchronous system, we have the following observation.

Observation 1 *For a synchronous system, the processing rate is equal to the product of the frequency and the throughput.*

A synchronous system is modeled by a directed graph $G = (V, E)$. In the simplest case, each vertex is a combinational gate and each edge is an interconnect wire with signal direction. When the system contains modules that could not be treated as gates, we will follow the method in [4], where vertices are pins of modules and edges represent either the interconnects or the fan-in fan-out timing constraints inside modules. Let the number of the flip-flops along a wire $e \in E$ be $w(e)$ and the interconnect delay along the wire be $d(e)$. By optimal buffer insertion [3], the interconnect delay is linear to its wire length. For any cycle C in the graph, we define $w(C) = \sum_{e \in C} w(e)$ and $d(C) = \sum_{e \in C} d(e)$.

Minimal period retiming optimizes a system by relocating the flip-flops so that the clock period of the system, which is equal to the longest combinational path delay between two consecutive flip-flops, is minimized. On the other hand, clock skew scheduling assigns non-zero skews to the clocks driving the flip-flops. The flip-flops act like that they are moved around to balance the combinational path delay so that the minimal clock period can be achieved. If there is a cycle C in the system, when applying minimal period retiming or clock skew scheduling, the flip-flops along C divide C into $w(C)$ combinational parts. So there must be a combinational path between two consecutive flip-flops C whose delay is at least $\frac{d(C)}{w(C)}$. Therefore, the minimal clock period is bounded by

$$\lambda_G = \max_{cycle\ C\ in\ G} \frac{d(C)}{w(C)} \tag{1}$$

As the throughput is always one here, the processing rate is the same as the frequency. So it is upper bounded by $\frac{1}{\lambda_G}$.

When the operating frequency is fixed, either latency insensitive design (LIS) [6] or wire-pipelining with correction [5] can be applied to pipeline the long wires.

LIS makes the system functionally insensitive to the latency of the long wires by a latency insensitive communication protocol and additional control logics around computational blocks in the original system. In LIS, signals propagating in the system are divided into two categories: informative events corresponding to the signals in the original systems and stalling events asking the receiving entities, called pearls, to stall a cycle waiting for the informative ones. To pipeline long wires, wire pipelining units, called relay stations, are placed on those wires. Relay stations act like flip-flops but can handle the communication protocol. In addition to the area overhead introduced by the additional control logics, the throughput is affected by the stalling events introduced to the system. As discussed in [7], the throughput is bounded by the minimum cycle mean among all cycles in the system where the cycle mean is defined as the pearl number plus the relay station number divided by the pearl number along a cycle.

According to [6, 7, 8], we use G to model a system such that every vertex corresponds to a computational block which could be treated as gates and every edge corresponds to a communication channel. $w(e)$s are all 1 now because initially there is no flip-flops on the communication channels and every computational block latches its outputs in flip-flops every clock cycle. Let the clock period be ϕ under the target frequency. We need at least $w^*(e) = \lceil \frac{d(e)}{\phi} \rceil$ wire-pipelining units to pipeline the wire e. Among them, one is the original flip-flop latching the output and all the others are relay stations. Therefore, the throughput is bounded by

$$\min_{cycle\ C\ in\ G} \frac{w(C)}{\sum_{e \in C} w^*(e)} \leq \min_{cycle\ C\ in\ G} \phi \frac{w(C)}{d(C)} = \frac{\phi}{\lambda_G}$$

So processing rate is bounded by $\frac{1}{\lambda_G}$.

Another approach [5] applies the wire-pipelining correcting method after inserting extra flip-flops to pipeline the long wires. We still use the ϕ to represent the clock period. The minimal number of flip-flops needed to pipeline a wire is

$w_p(e) = \lceil \frac{d(e)}{\phi} \rceil$. We cannot feed a new input every clock cycle without affecting the functionality if there is the additional latency caused by extra flip-flops in a cycle. With ρ-slow transformation [5], we feed a new input every ρ cycles where

$$\rho = \max_{cycle\ C\ in\ G} \lceil \frac{\sum_{e \in C} w_p(e)}{w(C)} \rceil$$

The throughput of the system is $\frac{1}{\rho}$. The processing rate is bounded by $\frac{1}{\lambda_G}$ again since

$$\begin{aligned} \frac{1}{\rho\phi} &\leq \min_{cycle\ C\ in\ G} \frac{w(C)}{\sum_{e \in C} w_p(e)\phi} \\ &\leq \min_{cycle\ C\ in\ G} \frac{w(C)}{\sum_{e \in C} d(e)} = \frac{1}{\lambda_G} \end{aligned}$$

From the above discussions, we have the following theorem.

Theorem 1 *$\frac{1}{\lambda_G}$ is the upper bound of the processing rate of a synchronous system no matter what technique is used for wire pipelining.*

This bound is independent of the operating frequency and afterward optimization methodologies and affected only by the interconnect configurations. Intuitively, designs with larger bounds are superior to the ones with smaller bounds since the afterward optimization methodologies could possibly achieve those bounds.

2.2. Problem Definition

We formulate the *Floorplanning for Processing Rate* (FPR) problem as follows.

Problem 1 (Floorplanning for Processing Rate) *In a directed graph $G = (V, E)$, every vertex represents a pin in a module with given width and height; every edge e represents a wire connecting two pins. Two weights are assigned to each wire e: $w(e)$ is the number of the flip-flops on e; $d(e)$ is the delay of the wire e. It is asked to find a floorplan to maximize the processing rate bound,*

$$\frac{1}{\lambda_G} = \min_{cycle\ C\ in\ G} \frac{\sum_{e \in C} w(e)}{\sum_{e \in C} d(e)} \quad (2)$$

3. Floorplanning for Processing Rate Optimization

3.1. ACG Floorplanning

Adjacent Constraint Graph (ACG) [10] is a representation for general floorplans. It is a constraint graph containing horizontal and vertical constraint edges. ACG simplifies the classical horizontal and vertical constraint graph by removing redundancies through three conditions: first, there is exactly one relation between any pair of modules; second, no transitive edge is allowed; third, there is no cross which is an edge configuration as shown in Figure 1. Allowing crosses in the graph may introduce quadratic number of edges as shown in Figure 2.

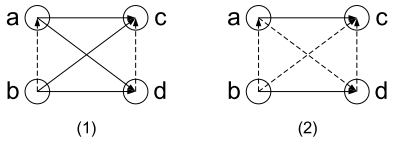

Figure 1. (1) Horizontal cross; (2) vertical cross.

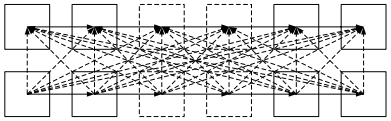

Figure 2. A constraint graph with quadratic number of edges.

To maintain those conditions, *Reduced ACG* is proposed in [11] according to the following property of ACG. As shown in Figure 3, the edges starting from a vertex are divided into four classes: class 1, edge to the adjacent vertex; class 2, edges in the same group (horizontal or vertical) as the class 1 one to the following vertices; class 3, the first edge in the group different from the class 1 one; class 4, the remaining edges, where every edge must be in the group different from the previous one. Reduced ACG is obtained by removing all the class 4 edges from ACG and there is an one-to-one mapping between ACG and Reduced ACG. Operations on Reduced ACG make only local changes to the graph and map to local perturbations in physical space. Figure 4 shows an example for the floorplan, ACG, and Reduced ACG.

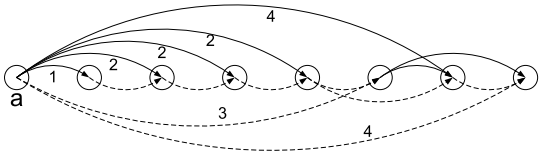

Figure 3. Edges starting from a are divided into 4 classes in ACG.

The SA based floorplanner used in [11] is extended to optimize the processing rate by combining the processing rate bound into the cost function. When a floorplan is obtained during SA, the physical locations of modules are computed and the delays of the interconnects are calculated as the Manhattan distance between modules. The bound $\frac{1}{\lambda_G}$ is computed directly as described in the following sections. We

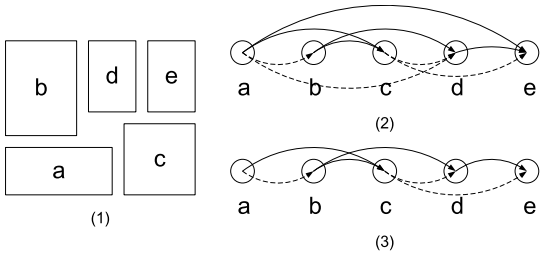

Figure 4. An example shows (1) floorplan; (2) ACG; (3) Reduced ACG.

also develop a method to address the fixed-outline constraint in the cost function.

3.2. Direct Bound Evaluation

Computing $\frac{1}{\lambda_G}$ is actually solving the minimum cycle ratio problem.

Given a strongly connected directed graph $G = (V, E)$ with two edge weight $w_1(e)$ and $w_2(e) > 0$ for each $e \in E$, the minimum cycle ratio problem is to compute the following minimum cycle ratio

$$\phi_{min} = \min_{cycle\ C\ in\ G} \frac{\sum_{e \in C} w_1(e)}{\sum_{e \in C} w_2(e)} \tag{3}$$

When $w_2(e) = 1$ for any edge e, it becomes the minimum cycle mean problem. According to [14], Howard's algorithm is the fastest one in practice to solve the minimum cycle mean problem. The version presented in their work is a simplified one from [15]. Based on the discussions in [15], we modify the implementation of Howard's algorithm in Section 2.5 of [14] to solve the minimum cycle ratio problem.

The intuition behind Howard's algorithm is to maintain a policy graph G_π through the computation. A policy graph is a sub-graph of G where there is exactly one edge starting from any vertex. The cycles in a policy graph can be enumerated since there is exactly one cycle in each weakly connected component of G_π. The minimum cycle ratio ϕ of G_π is obtained then, which is obviously an upper bound of ϕ_{min}. It can be asserted that $\phi = \phi_{min}$ if there is no negative cycle in G regarding to the edge weights $w_1(e) - w_2(e)\phi$. If there are negative cycles, one of them is identified in a newly constructed policy graph; this cycle has a cycle ratio less than ϕ. A vertex labeling is maintained to interleave the above two steps, i.e., to check for the negative cycles and to construct a new G_π.

In order to evaluate $\frac{1}{\lambda_G}$ of a system G given its floorplan, the graph G is first decomposed into strongly connected components (SCC). Only one such computation is needed per benchmark and the *Strongly–Connected–Components* algorithm in [16] works well here. We assign $d(e)$ to be the $w_2(e)$ and $w(e)$ to be the $w_1(e)$. By applying Howard's algorithm for minimum cycle ratio in each SCC and picking up the minimum among them, we obtain $\frac{1}{\lambda_G}$.

3.3. Incremental Bound Evaluation

In Howard's algorithm, different initial G_π's affect the number of the iterations and thus the running time. Intuitively, an initial G_π with a smaller ϕ tends to converge quicker than the one with a larger ϕ.

In our floorplanners, the floorplan does not change too much between successive steps in SA because of those local Reduced ACG perturbations. The final G_π's of the previous floorplan obtained by Howard's algorithm give near-optimal ϕ's. We reuse those final G_π's as our initial policy graphs for each SCC. For the first floorplan when those G_π's are not available, we follow [14] to construct it by choosing the edge with the minimum w_1 weight starting from each vertex.

As shown in Section 4, this incremental technique speeds up the floorplanner by 29% on average.

3.4. Handle the Fixed-outline Constraint

Following [13], the fixed-outline constraint is modeled with two constants. One is the maximum white-space fraction γ and the other is the aspect ratio $\alpha \geq 1$. Suppose the total area of all the modules are A. The desired width and height of the fixed-outline floorplan are computed as

$$H_* = \sqrt{(1+\gamma)A\alpha}, W_* = \sqrt{(1+\gamma)A/\alpha} \tag{4}$$

In [13], several approaches were proposed to handle the fixed-outline constraint. Addressing the constraint directly in the cost function was shown to be not successful for a classical SA based sequence-pair floorplanner. Adding slack-based moves was proposed to solve this problem. These moves change the aspect ratio of the floorplan toward the desired one. By applying these moves when the aspect ratio is not the desired one, a floorplan satisfying the fixed-outline constraint may be obtained during SA.

Instead of designing a new move and mixing it to the existing ones, we propose a way to handle the fixed-outline constraint directly in the cost function. For a floorplan with width W and height H where $H \geq W$, the cost associated with the fixed-outline constraint is defined as

$$outline\_cost = e^{\max(\frac{W}{W_*},1)+\max(\frac{H}{H_*},1)-2} \tag{5}$$

This cost is larger than 1 if the constraint is not met; when the constraint is satisfied, the cost becomes 1. The total cost of a floorplan is calculated as the product of the $outline\_cost$ and the other cost. The intuition is that the exponential function is very sensitive when the outline difference is large but less sensitive when the outline difference is small. So the floorplanner will push the floorplan hard toward the desired outline when there is enough whitespace. When the outline constraint is almost or totally satisfied, optimizing other cost becomes the priority.

It is possible during SA some floorplans meet the fixed-outline constraint but the final floorplan do not meet the constraint. So the best floorplan meeting the fixed-outline constraint is recorded during SA and is reported at the end.

4. Experimental Results

4.1. Experimental Setup

The experiments are conducted on a Windows machine with 1.4GHz Pentium M processor and 512M memory where the floorplanner is implemented using C++ and compiled with GCC 3.4.2.

Following the experimental setup in [8], we pick up GSRC benchmarks including n10, n30, n50, n100, n200, and n300. The signal directions and flip-flops are derived as follows according to [8]. The last pin of a net is treated as the source of the net and then the net is broken into two-pin nets. All the pins belong to one module are treated as one pin located at the center of the module so that the modules could be treated as gates. There is exactly one flip-flop on each edge to represent the situation where each module latches its outputs.

Classical MCNC benchmarks were used in [8] as well. However, as the sources of the nets were determined randomly in their work, we cannot generate the same configuration and thus we only experiment on GSRC benchmarks.

4.2. Results for Floorplanning for Processing Rate

We first test the floorplanner without fixed-outline constraint. The cost function used is $4\sqrt{area} + \lambda_G$ and the incremental bound evaluation method is employed. We run each benchmark for ten times and the best one is reported in Table 1 where the "ws (%)" column shows the white space in percentage.

For each floorplan, we record the random seed used and use it to perform another SA floorplanning without the incremental bound evaluation. The resulting floorplan is the same but the running time is different. The experimental results are reported in Table 2. For the incremental one and the non-incremental one (under "non-incre."), we report the running time as well as the total number of iterations (in "#iter.") of Howard's algorithm. Improvements in running time are reported in "impr." column and the average improvement is 29%.

circuit	area	ws (%)	λ_G	time (sec)
n10	240.6K	7.9	292.8	47.9
n30	220.9K	5.6	296.0	46.5
n50	217.1K	8.5	239.5	45.7
n100	196.0K	8.4	185.0	56.2
n200	195.6K	10.2	373.9	475.9
n300	312.7K	12.7	497.3	540.4

Table 1. Floorplanning for processing rate.

To compare our results with those in [8], the throughput for our final floorplans under different frequencies are computed. The frequencies are modeled by the *critical length*, which is the distance that a signal travels in one clock cycle. The corresponding critical lengths are 30%, 50%, 70%, and 100% of the square root of the total area of all the modules. According to Section 2, we calculate $w^*(e)$ first. Then by applying Howard's algorithm the same way as computing $\frac{1}{\lambda_G}$, the throughput is obtained.

circuit	incremental		non-incre.		impr. (%)
	time	#iter.	time	#iter.	
n10	47.9	8.9M	56.9	9.6M	15.8
n30	46.5	4.1M	65.0	5.3M	28.5
n50	45.7	3.8M	69.2	4.9M	34.0
n100	56.2	4.4M	75.4	5.5M	25.5
n200	475.9	5.1M	740.6	7.0M	35.7
n300	540.4	4.5M	827.6	6.7M	34.7
average improvement					29.0

Table 2. Incremental vs. non-incremental bound evaluation.

circuit	u.b.	100%	70%	50%	30%
n10	3.42 *	1.70	2.43	2.83	2.65
n30	3.38	1.82	1.95	2.43	2.55
n50	4.18	1.87	2.40	2.69	3.08
n100	5.41	2.36	2.74	3.15	3.93
n200	2.67	1.52	1.66	1.91	2.21
n300	2.01	1.15	1.37	1.45	1.59

* All the numbers shown are in the unit $\times 10^{-3}$.

Table 4. Processing rate vs. upper bound.

In [8], the experiments were conducted on a 1.4GHz Pentium III machine. For each circuit among n10, n30, n50, n100 and each of those critical lengths, they ran the experiment for 60 seconds ten times and we copy the best ones. They did not report the results for n200 and n300 but we report our results for these two benchmarks. The results are compared in Table 3 with the format $1 - throughput/white\ space\ (\%)$, which means that the smaller the number, the better. The dominating solutions in throughput and white space are highlighted. It can be seen that our results dominate theirs for about half of the cases and are not dominated by theirs for the other half. Moreover, since the machine configurations are comparable, our running times are much better considering [8] spent $60 \times 4 = 240$ seconds for each benchmark.

In Table 4, we report the processing rates for each critical lengths as well as the bounds $\frac{1}{\lambda_G}$ (in "u.b."). Although the bounds tend to be looser for larger critical lengths, we believe that the fidelity of the bound to the throughput makes our approach effective.

4.3. Results for Fixed-outline Floorplanning for Processing Rate

The fixed-outline constraint is handled explicitly by the cost function. The cost function used is $outline\_cost \times (w\sqrt{area} + \lambda_G)$. The area weight w is set to 0.5 for n10, n30, n50, and n100 and 2 for n200 and n300. Small w gives

circuit	method	100%	70%	50%	30%	time (sec)
n10	FPR	0.200/7.9	0.200/7.9	0.333/7.9	0.625/7.9	47.9
	[8]	0.166/8.9	0.250/7.2	0.500/4.0	0.636/6.2	60.0
n30	FPR	**0.167/5.6**	**0.375/5.6**	**0.444/5.6**	0.650/5.6	46.5
	[8]	0.200/6.9	0.375/8.3	0.500/6.8	0.636/8.5	60.0
n50	FPR	0.167/8.5	0.250/8.5	0.400/8.5	0.588/8.5	45.7
	[8]	**0.133/7.2**	0.375/6.9	0.473/6.7	0.636/7.2	60.0
n100	FPR	**0.000/8.4**	**0.188/8.4**	**0.333/8.4**	**0.500/8.4**	56.2
	[8]	0.000/9.1	0.200/9.1	0.375/8.6	0.500/9.7	60.0
n200	FPR	0.364/10.2	0.514/10.2	0.600/10.2	0.722/10.2	475.9
	[8]	-	-	-	-	-
n300	FPR	0.400/12.7	0.500/12.7	0.620/12.7	0.750/12.7	540.4
	[8]	-	-	-	-	-

Table 3. Throughput, white space, and running time: FPR (PM 1.4GHz) vs. [8] (PIII 1.4GHz).

better processing rate bound but if meeting the fixed-outline constraint is a problem, w is increased to emphasize better area. We run our floorplanner with the maximum white-space of $\gamma = 15\%$ and the aspect ratio of $\alpha = 1$, $\alpha = 1.5$, and $\alpha = 2$ respectively. The best results from ten runnings are reported in Table 5. The numbers are in the format $\lambda_G/running\ time\ (sec)$.

circuit	$\alpha = 1.0$	$\alpha = 1.5$	$\alpha = 2.0$
n10	282.0/51.9	282.5/51.2	303.8/65.5
n30	248.3/60.5	254.4/55.0	269.0/55.0
n50	209.0/62.7	210.7/59.2	221.1/60.1
n100	144.2/121.9	138.0/119.4	140.7/123.1
n200	358.2/637.8	378.8/620.6	382.5/863.9
n300	494.0/1091	548.4/840.2	539.8/830.9

Table 5. Fixed-outline floorplanning for processing rate.

5. Conclusion

We showed that optimizing the processing rate bound, which is the minimum ratio of the flip-flop number to the delay in any cycle, is more general than either optimizing the clock period or the throughput for a sequential system because the bound is independent of the operating frequencies or the available design methodologies. We built a SA based floorplanner optimizing for the processing rate bound by evaluating it directly in the inner loop of SA without introducing much overhead in running time. Moreover, exploiting the incremental structure of the evaluating algorithm sped up the evaluating process. Experimental results confirmed the effectiveness of our approach.

References

[1] C. E. Leiserson and J. B. Saxe. *Optimization Synchronous Systems.* Journal of VLSI and Computer Systems, 1:41-67, 1983

[2] J. P. Fishburn. *Clock Skew Optimization.* IEEE Trans. on Computers, 39(7):945-951, July 1990.

[3] R. H. J. M. Otten and R. Brayton. *Planning for Performance.* In DAC, pages 122-127, 1998.

[4] C. Lin and H. Zhou. *Retiming for Wire Pipelining in System-On-Chip.* In ICCAD, pages 215-220, 2003.

[5] V. Nookala and S. S. Sapatnekar. *A Method for Correcting the Functionality of a Wire-Pipelined Circuit.* In DAC, pages 570-575, 2004.

[6] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. *A Methodology for Correct-by-Construction Latency Insensitive Design.* In ICCAD, pages 309-315, 1999.

[7] L. P. Carloni and A. L. Sangiovanni-Vincentelli. *Performance Analysis and Optimization of Latency Insensitive Systems.* In DAC, pages 361-367, 2000.

[8] M. R. Casu and L. Macchiarulo. *Floorplanning for Throughput.* In ISPD, pages 62-69, 2004.

[9] A. P. Hurst, P. Chong, and A. Kuehlmann. *Physical Placement Driven by Sequential Timing Analysis.* In ICCAD, pages 379-386, 2004.

[10] H. Zhou and J. Wang. *ACG–Adjacent Constraint Graph for General Floorplans.* In ICCD, pages 572-575, 2004.

[11] J. Wang and H. Zhou. *Interconnect Estimation without Packing via ACG Floorplans.* In ASPDAC, pages 1152-1155, 2005.

[12] A. B. Kahng. *Classical Floorplanning Harmful?* In ISPD, pages 207-213, 2000.

[13] S. N. Adya, and I. L. Markov. *Fixed-outline Floorplanning: Enabling Hierarchical Design.* IEEE Trans. On VLSI Systems, 11(6):1120-1135, December 2003.

[14] A. Dasdan, S. S. Irani, and R. K. Gupta. *Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems.* In DAC, pages 37-42, 1999.

[15] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. Mc Gettrick, and J-P Quadrat. *Numerical Computation of Spectral Elements in Max-plus Algebra.* In Proc. IFAC Conf. on Syst. Structure and Control, 1998.

[16] T. H. Cormen, C. E. Leiserson, R. H. Rivest, and C. Stein. *Introduction to Algorithms.* 2nd ed., MIT Press, 2001.