Clustering for Processing Rate Optimization[*]

Chuan Lin, Jia Wang, and Hai Zhou
Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208

Abstract

Clustering (or partitioning) is a crucial step between logic synthesis and physical design in the layout of a large scale design. A design verified at the logic synthesis level may have timing closure problems at post-layout stages due to the emergence of multiple-clock-period interconnects. Consequently, a trade-off between clock frequency and throughput may be needed to meet the design requirements. In this paper, we find that the processing rate, defined as the product of frequency and throughput, of a sequential system is upper bounded by the reciprocal of its maximum cycle ratio, which is only dependent on the clustering. We formulate the problem of processing rate optimization as seeking an optimal clustering with the minimal maximum-cycle-ratio in a general graph, and present an iterative algorithm to solve it. Since our algorithm avoids binary search and is essentially incremental, it has the potential of being combined with other optimization techniques. Experimental results validate the efficiency of our algorithm.

1 Introduction

Circuit clustering (or partitioning) is often employed between logic synthesis and physical design to decompose a large circuit into parts. Each part will be implemented as a separate cluster that satisfies certain design constraints, such as the size of a cluster. Clustering helps to provide the first order information about interconnect delays as it classifies interconnects into two categories: intra-cluster ones are local interconnects due to their spatial proximity while inter-cluster ones may become global interconnects after floorplan/placement and routing (also known as circuit layout).

Due to aggressive technology scaling and increasing operating frequencies, interconnect delay has become the main performance limiting factor in large scale designs. Industry data shows that even with interconnect optimization techniques such as buffer insertion, the delay of a global interconnect may still be longer than one clock period, and multiple clock periods are generally required to communicate such a global signal. Since global interconnects are not visible at logic synthesis when the functionality of the implementation is the major concern, a design that is correct at the logic synthesis level may have timing closure problems after layout due to the emergence of multiple-clock-period interconnects.

This gap has motivated recent research to tackle the problem from different aspects of view. Some of them resort to retiming [14], which is a traditional sequential optimization technique that moves flip-flops within a circuit without destroying its functionality. It was used in [21, 4, 17, 20, 16] to pipeline global interconnects so as to reduce the clock period. Although retiming helps to relieve the criticality of global interconnects, there is a lower bound of the clock periods that can be achieved because retiming cannot change the latency of either a (topological) cycle or an input-to-output path in the circuit. In case that the lower bound does not meet the frequency requirement, redesign and re-synthesis may have to be carried out.

One way to avoid redesign is to insert extra wire-pipelining units like flip-flops to pipeline long interconnects, as done within Intel [5] and IBM [13]. It can be shown that if the period lower bound is determined by an input-to-output path, pipelining can reduce the lower bound without affecting the functionality. However, if the period lower bound is given by a cycle, inserting extra flip-flops in it will change its functionality.

C-slow transformation [14] is a technique that slows down the input issue rate[1] of the circuit to accommodate higher frequencies. It was thus used in [18] to retain the functionality when extra flip-flops were inserted in cycles. In other words, throughput was sacrificed (became $1/C$) to meet the frequency requirement.

Instead of slowing down the throughput uniformly over the whole circuit, Latency Insensitive Design (LID) [2, 1], on the other hand, employs a protocol that slows down the throughput of a part of the circuit only when it is needed. As a result, LID can guarantee minimal throughput reduction while satisfying the frequency requirement.

We show in Section 2 that the aforementioned three approaches (retiming, pipelining with C-slow, and pipelining with LID) can be unified under the same objective function of maximizing the *processing rate*, defined as the product of frequency and throughput, as illustrated in Figure 1. In addition, the processing rate of a sequential system is upper bounded by the reciprocal of the maximum cycle ratio of the system, which is only dependent on the clustering. Therefore, we propose an optimal algorithm that finds a clustering with the minimal maximum-cycle-ratio.

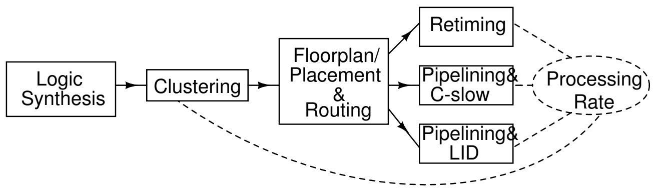

Figure 1: A logical and physical design flow.

The rest of this paper is organized as follows. Section 2 presents the problem formulation. Two previous works are reviewed in Section 3. Section 4 defines the notations and constraints used in this paper. Our algorithm is elaborated in Section 5, followed by the implementation details in Section 6. We present some experimental results in Section 7. Conclusions are given in Section 8. Due to space limit, all the proofs are omitted, which can be found in [15].

2 Problem formulation

We consider clustering subject to a size limit for clusters. More specifically, each gate has a specified size, as well as

[*]This work was supported by the NSF under CCR-0238484.

[1]The issue rate is defined as the number of clock periods between successive input changes. An issue rate of 1 indicates that the inputs can change every clock period.

each interconnect. We require that the size of each cluster, defined as the sum of the sizes of the gates and the interconnects in the cluster, should be no larger than a given constant A. Replication of gates is allowed, i.e., a gate may be assigned to more than one cluster in the layout. When a gate is replicated, its incident interconnects are also replicated so that the clustered circuit is logically equivalent to the original circuit. Figure 2 (taken from [19]) shows an example of gate replication in a clustering.

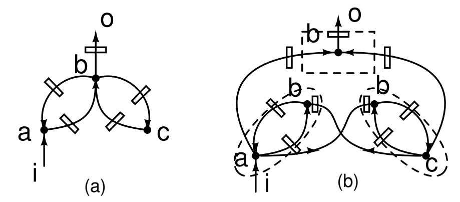

Figure 2: (a) An example circuit; (b) A clustering with 3 replicas of gate b.

Given a particular clustering c, we treat the replicas of gates and the original ones distinctly and denote them all as V_c. We use E_c to denote the set of interconnects among V_c. The clustered circuit is represented as $G_c = (V_c, E_c)$. In order for the circuit to operate at a specified clock period λ, additional wire-pipelining flip-flops are inserted. For all cycle o_c in G_c, let $d(o_c)$ denote the cycle delay, $w(o_c)$ and $w_\lambda(o_c)$ denote the number of flip-flops in o_c before and after additional pipelining flip-flops are inserted, respectively. Assuming $w(o_c) > 0$, the cycle ratio of o_c is defined as $\phi(o_c) = d(o_c)/w(o_c)$. Note that $\phi(o_c)$ is defined using $w(o_c)$, not $w_\lambda(o_c)$. The *maximum cycle ratio* over all the cycles in G_c is denoted as $\phi_c = \max_{o_c \in G_c} \phi(o_c)$.

We define *processing rate* as follows.

Definition 1 *For a sequential system, processing rate is defined as the length of processed input sequence per unit time. In particular, it is the product of frequency and throughput in a synchronous system.*

The larger the processing rate, the better the sequential system. Given the above definition, the approach of retiming actually maximizes the processing rate by minimizing the period while keeping the throughput. It is interesting to notice that the approach of pipelining with C-slow transformation also maximizes the processing rate for a specified period by computing the least slowdown of the issue rate, which is transformed into throughput reduction. As an alternative, Latency Insensitive Design (LID) helps the clustered circuit reach the maximum throughput for a specified period. Therefore, all the three approaches can be unified under the same objective function of maximizing the processing rate.

It was shown in [3] that the maximum throughput ρ_λ of an LID for a specified period λ can be computed as

$$\rho_\lambda = \min_{\text{cycle } o_c \in G_c} \frac{w(o_c)}{w_\lambda(o_c)}.$$

On the other hand, the fact that the circuit can operate at the specified period λ after the insertion of additional flip-flops implies that $w_\lambda(o_c)\lambda \geq d(o_c)$, i.e., $\frac{1}{w_\lambda(o_c)} \leq \frac{1}{d(o_c)/\lambda}$, $\forall$cycle $o_c \in G_c$. Substitute this into the formula of ρ_λ to get

$$\rho_\lambda \leq \min_{o_c \in G_c} \frac{w(o_c)}{d(o_c)/\lambda} = \min_{o_c \in G_c} \frac{\lambda}{\phi(o_c)} = \frac{\lambda}{\phi_c}.$$

It follows that the maximum processing rate of an LID is upper bounded by $\frac{1}{\phi_c}$ since

$$\textit{max processing rate} = \frac{1}{\lambda} \cdot \rho_\lambda \leq \frac{1}{\lambda} \cdot \frac{\lambda}{\phi_c} = \frac{1}{\phi_c}.$$

It is also an upper bound of the maximum processing rate obtained by the approach of retiming, as shown in [21]. In other words, all the three approaches share the same upper bound of their common objective.

To maximize the processing rate, one can either maximize the upper bound or try to achieve the upper bound. They are equally important. However, since achieving the upper bound requires further knowledge on physical design, such as buffer and flip-flop allowable regions [8, 21] while the upper bound itself is only dependent on the maximum cycle ratio of the clustered circuit, we will consider how to optimally cluster the circuit such that the upper bound is maximized, or equivalently, the maximum cycle ratio is minimized.

In order to compute the maximum cycle ratio, we need to know how to compute the delay of a cycle during clustering. Although local interconnect delays can be obtained using some delay models at synthesis, the delays of global interconnects are not available until layout. Therefore during clustering, we assume that each global interconnect induces an extra constant delay D, e.g., if interconnect (u, v) with delay $d(u, v)$ is assigned to be inter-cluster, then its delay becomes $d(u, v) + D$.

Since we want to minimize the maximum cycle ratio, the path delays from primary inputs (PIs) to primary outputs (POs) can be ignored as they can be mitigated by pipelining. This motivates us to formulate the problem in a strongly connected graph as follows.

Problem 1 (Optimal Clustering Problem) *Given a directed, strongly connected graph $G = (V, E)$, where each vertex $v \in V$ has a delay $d(v)$ and a specified size, and each edge $(u, v) \in E$ has a delay $d(u, v)$, a specified size and a weight $w(u, v)$ (representing the number of flip-flops on it), find a clustering of vertices with possible vertex replication such that: 1. the size of each cluster is no larger than a given constant A; 2. each global interconnect induces an extra constant delay D; 3. the maximum cycle ratio of the clustered circuit is minimized.*

For simplicity, we assume that each gate has unit size and the size of each interconnect is zero. Our proposed algorithm can be easily extended to handle various size scenarios.

3 Previous work

Pan *et al.* [19] proposed to optimally cluster a sequential circuit such that the lower bound of the period of the clustered circuit was minimized with retiming. However, the period lower bound may not come from a cycle ratio. In addition, their algorithm needs to start from PIs, thus cannot be used to solve our problem in a strongly connected graph. In this sense, they solved a different problem, even though it looks similar to ours.

Their problem was solved by binary search, using a test for feasibility as a subroutine. For each target period, they used a procedure called *labeling computation* to check the feasibility. The procedure starts with label assignment 0 for PIs and $-\infty$ for the other vertices, and repeatedly increases the label values until they all converge or the label value of some PO exceeds the target period, for which the target period is considered infeasible. For each vertex, the amount of increase in its label is computed using another binary search that basically selects the minimum from a candidate set. Because of the nested binary searches, their algorithm is relatively slow. In addition, the algorithm requires $O(|V|^2)$ space to store a pre-computed all-pair longest-path matrix, which is impractical for large designs. Cong *et al.* [9] improved the algorithm by tightening the candidate set to speed up the labeling computation, and by reducing the space complexity to linear dependency. But the improved algorithm still needs the nested binary searches.

Besides the difference in problem formulation, our algorithm differs from theirs in two algorithmic aspects. Firstly, our algorithm focuses on cycles, thus can work on any general graph. Secondly, no binary search is employed in our algorithm. As a result, our algorithm is efficient and essentially incremental. Like [9], our algorithm does not need pre-computed information on paths either.

Except for these differences, [19] revealed some important results on clustering, which we employ here to simplify our notations.

- Each cluster has only one output, which is called the *root* of the cluster. If there is a cluster with more than one output, we can replicate the cluster so that each copy of the cluster has only one output.
- For each vertex in V, there is at most one cluster rooted at it and its arrival time (defined in Section 4) is not larger than the arrival times of its replicas.
- If $u \in V$ is an input of the cluster rooted at $v \in V$, then the cluster rooted at v must not contain a replica of u.

4 Notations and constraints

For a particular clustering c and a path $p_c = u \rightsquigarrow v$ in G_c, we use $w(p_c)$ to represent the number of flip-flops on p_c, which is the sum of the weights of p_c's constituent edges. Similarly, $d(p_c)$ represents the delay along p_c, which is the sum of the delays of p_c's constituent edges and vertices, except for $d(u)$. Note that the delay of an inter-cluster edge $(u, v) \in G_c$ is $d(u, v) + D$. We use $\phi(o_c)$ to denote the cycle ratio of o_c, and ϕ_c to denote the maximum cycle ratio of G_c.

Since we only need to consider clusters rooted at the vertices in V, at most one for each vertex, we use c_v to refer to the set of vertices that are included in the cluster rooted at $v \in V$. Let $i_v \subset V$ be the set of inputs of c_v and $r_v \subseteq V_c - V$ be the set of replicas of $v \in V$. In the remainder of this paper, when we say $u \in c_v$ $(u \neq v)$, we mean that the cluster rooted at $v \in V$ contains a replica of $u \in V$. For example, Figure 3(a) shows a circuit before clustering. There are five vertices (a-e) and seven edges. Figure 3(b) illustrates a clustering of the circuit with size limit $A = 3$, where dashed circles represent clusters. For each cluster, the vertex whose index is outside the cluster indicates the root. For example, c_a contains replicas of vertices c and e with the input set $i_a = \{d\}$.

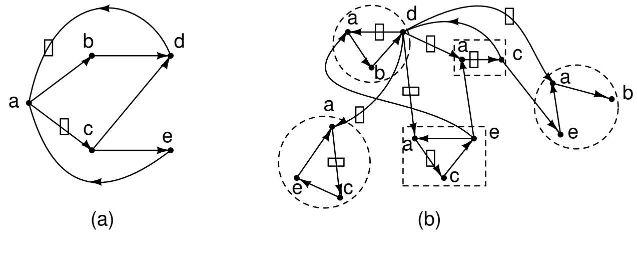

Figure 3: An example of clustering representation

We use a label $t : V \to \Re$ to denote the arrival time of the vertex. To ease the presentation, we will extend the domain of t to V_c to represent the arrival times of the replicas of the vertices. Based on this, a clustering that satisfies the cluster size requirement and has a maximum cycle ratio not larger than a given value ϕ can be characterized as follows.

$$t(v) \geq 0, \qquad \forall v \in V \tag{1}$$
$$t(v) \leq t(v'), \qquad \forall v' \in r_v, v \in V \tag{2}$$
$$t(v) \geq t(u) + d(u,v) + d(v) - w(u,v)\phi, \quad \forall (u,v) \in E_c, u, v \in c_x, x \in V \tag{3}$$
$$t(v) \geq t(u) + d(u,v) + D + d(v) - w(u,v)\phi, \quad \forall (u,v) \in E_c, u \in i_x, v \in c_x, x \in V \tag{4}$$
$$|c_v| \leq A, \qquad \forall v \in V \tag{5}$$

where (1)-(4) guarantee that the arrival times are all achievable, and (5) is the cluster size requirement. In particular, (2) ensures that the arrival time of $v \in V$ is no larger than the arrival times of its replicas.

Following the convention, $(u, v) \in E_c$ is a *critical edge* under ϕ iff it is intra-cluster with $t(v) = t(u)+d(u, v)+d(v)- w(u, v)\phi$, or it is inter-cluster with $t(v) = t(u) + d(u, v) + D + d(v) - w(u, v)\phi$. A *critical path* under ϕ refers to a path whose constituent edges are all critical under ϕ. Vertex u is a *critical input* of c_v under ϕ iff $u \in i_v$ and v can be reached by u through a critical path $p = u \to x \rightsquigarrow v$ under ϕ where the sub-path $x \rightsquigarrow v$ is in c_v. When a critical path actually forms a cycle, it is then called a *critical cycle*. Cycle o_c is critical under ϕ iff $d(o_c) = w(o_c)\phi$.

A *legal clustering* must satisfy (5). When the arrival times of a legal clustering satisfy (1)-(4) under ϕ, it is called a *feasible clustering* under ϕ. When a critical cycle is present in a feasible clustering under ϕ, it is called a *critical clustering* under ϕ. A given ϕ is *feasible* iff there exists a feasible clustering under ϕ. We must note that for a legal clustering, its maximum cycle ratio is feasible. In fact, any value larger than the maximum cycle ratio of a legal clustering is also feasible.

Consider a feasible clustering c under ϕ. For all $(u, v) \in E$, it is either in E_c with $u \in i_v$, or there is an edge (u', v) such that $u' \in r_u$. In either case, the following inequality is true by (2)-(4).

$$t(v) \geq t(u) + d(u,v) + d(v) - w(u,v)\phi, \quad \forall (u,v) \in E \tag{6}$$

The following lemma provides a lower bound for ϕ.

Lemma 1 *A feasible ϕ is no smaller than the maximum cycle ratio of G, denoted as ϕ_{lb}.*

Define

$$\Delta(u, v, \phi) \overset{\triangle}{=} \max_{p \in u \rightsquigarrow v \ in \ G} \big(d(p) - w(p)\phi\big), \quad \forall u, v \in V$$

Lemma 1 ensures that $\Delta(u, v, \phi)$ is well-defined on feasible ϕ's.

5 Algorithm

5.1 Overview

The optimal clustering problem asks for a legal clustering with the minimal maximum-cycle-ratio. Since $A > 0$, the clustering with each vertex being a cluster is certainly legal. Starting from it, we will iteratively improve the clustering by reducing its maximum cycle ratio until the optimality is certified.

First of all, the maximum cycle ratio of a legal clustering is feasible and can be efficiently computed using Howard's algorithm [6, 11]. Given a feasible ϕ, we show that, unless ϕ is already the optimal solution, a particular legal clustering can be constructed whose maximum cycle ratio is smaller than ϕ. The smaller ϕ can be obtained by applying Howard's algorithm on the constructed clustering. Therefore, we alternate between applying Howard's algorithm and constructing a better clustering until the optimal ϕ is reached.

5.2 Clustering under a feasible $\phi > \phi_{\rm lb}$

Given a feasible $\phi > \phi_{\rm lb}$, we show in this section how to construct a feasible clustering under ϕ, i.e., a clustering satisfying (1)-(6) under ϕ, whose maximum cycle ratio is no larger than ϕ. The constructed clustering has special properties, based on which we explain in Section 5.3 how to construct a feasible clustering under ϕ whose maximum cycle ratio is strictly smaller than ϕ, if ϕ is not the optimal.

We choose to first satisfy (1) and (6) because they are independent on clustering, and iteratively update $t(v)$ and c_v to satisfy (2)-(5) while keeping (1) and (6).

Let T denote the arrival time vector, i.e.,

$$\mathrm{T} = \big(t(1), t(2), \ldots, t(|V|)\big).$$

A partial order $(\leq)$ can be defined between two arrival time vectors T and T$'$ as follows.

$$\mathrm{T} \leq \mathrm{T}' \overset{\triangle}{=} t(v) \leq t'(v),\ \forall v \in V.$$

According to the lattice theory [12], if we treat assignment $t(v) = 0$, $\forall v \in V$ as the bottom element $(\perp)$ and assignment $t(v) = \infty$, $\forall v \in V$ as the top element $(\top)$, then the arrival time vector space $\Re^{|V|}$ becomes a *complete partially ordered set*, that is, for all $\mathrm{T} \in \Re^{|V|}$, $\perp \leq \mathrm{T} \leq \top$.

To satisfy (1), we set $t(v) = 0$, $\forall v \in V$. Then we apply Bellman-Ford's algorithm [10], denoted as BF, on E to satisfy (6) under ϕ. Bellman-Ford's algorithm is guaranteed to work as long as $\phi \geq \phi_{\rm lb}$.

The resulting arrival time vector is denoted as

$$\mathrm{T}_0 = BF(\perp, \phi).$$

In fact, T_0 is the *least* vector satisfying (1) and (6), as stated in the following lemma.

Lemma 2 $\mathrm{T}_0 \leq \mathrm{T}$*, for all* T *satisfying (1) and (6).*

In order to satisfy (2)-(5) while keeping (1) and (6), we define transformation $\mathcal{L} : (\Re^{|V|}, \Re) \rightarrow \Re^{|V|}$ as follows.

For all $v \in V$, we will construct a new cluster c'_v rooted at v. The procedure starts with $c'_v = \{v\}$ and grows c'_v progressively by including one critical input at a time. Note that when a vertex is put in c'_v, its preceding vertex that is outside c'_v becomes an input of c'_v. Let $t(v)$ and $t'(v)$ denote the arrival time of v before and after c_v is replaced by c'_v, respectively. The procedure will stop only when either $|c'_v| = A$ or $t'(v) \leq t(v)$. If $t'(v) < t(v)$, we keep $t(v)$ and c_v unchanged; otherwise we update $t(v)$ and c_v with $t'(v)$ and c'_v respectively. Be aware that the constructed c'_v may not be unique since the inclusion of a critical input is arbitrary if there are multiple of them. The resulting arrival time of v is denoted as $\mathcal{L}_v(\mathrm{T}, \phi)$. The next lemma helps to identify the critical input to be included at each time.

Lemma 3 *For all $x \notin c'_v$, $t'(v) \geq t(x) + D + \Delta(x, v, \phi)$. In particular, if u is a critical input of c'_v, then $t'(v) = t(u) + D + \Delta(u, v, \phi)$.*

Define $\mathcal{L}(\mathrm{T}, \phi)$ as the arrival time vector when all the $\mathcal{L}_v(\mathrm{T}, \phi)$'s, $\forall v \in V$, are applied once, followed by Bellman-Ford's algorithm to ensure (6), expressed as

$$\mathcal{L}(\mathrm{T}, \phi) \overset{\triangle}{=} BF\Big(\big(\mathcal{L}_1(\mathrm{T}, \phi), \mathcal{L}_2(\mathrm{T}, \phi), ..., \mathcal{L}_{|V|}(\mathrm{T}, \phi)\big), \phi\Big).$$

The following lemma shows that $\mathcal{L}$ is an order-preserving transformation.

Lemma 4 *For any* T *and* $\bar{\mathrm{T}}$ *satisfying (1) and (6), if* $\mathrm{T} \leq \bar{\mathrm{T}}$*, then* $\mathcal{L}(\mathrm{T}, \phi) \leq \mathcal{L}(\bar{\mathrm{T}}, \phi)$.

We say that T is a *fixpoint of* $\mathcal{L}$ under ϕ if and only if $\mathrm{T} = \mathcal{L}(\mathrm{T}, \phi)$. The following theorem bridges the existence of a fixpoint and the feasibility of ϕ.

Theorem 1 *ϕ is feasible if and only if $\mathcal{L}$ has a fixpoint under ϕ.*

In fact, according to the lattice theory [12], if $\mathcal{L}$, defined on a complete partially ordered set, has a fixpoint under ϕ, then it has a *least fixpoint* T^ϕ, defined as

$$\big(\mathrm{T}^\phi = \mathcal{L}(\mathrm{T}^\phi, \phi)\big) \wedge \big(\forall \mathrm{T} : \mathrm{T} = \mathcal{L}(\mathrm{T}, \phi) : \mathrm{T}^\phi \leq \mathrm{T}\big).$$

We use c^ϕ to denote the clustering constructed by $\mathcal{L}(\mathrm{T}^\phi, \phi)$. Note that c^ϕ may not be unique since c'_v may not be unique, $\forall v \in V$.

To reach a fixpoint, iterative method can be used on $\mathcal{L}$. It starts with T_0 as the initial vector, iteratively computes new vectors from previous ones $\mathrm{T}_1 = \mathcal{L}(\mathrm{T}_0, \phi), \mathrm{T}_2 = \mathcal{L}(\mathrm{T}_1, \phi), \ldots$ until it finds a T_n such that $\mathrm{T}_n = \mathrm{T}_{n-1}$. The following lemma states that applying iterative method on $\mathcal{L}$ will converge to its least fixpoint in $|V| - 1$ iterations.

Lemma 5 *If T^ϕ exists under ϕ, applying iterative method on $\mathcal{L}$ will converge to it in at most $|V| - 1$ iterations.*

5.3 Non-critical clustering under a feasible $\phi > \phi_{\rm lb}$

Figure 4 shows two clusterings of the circuit in Figure 3(a) under $\phi = 20$. Their arrival time vectors are both the least fixpoint under $\phi = 20$, that is, $t^\phi(a) = 0$, $t^\phi(b) = 10$, $t^\phi(c) = 0$, $t^\phi(d) = 10$ and $t^\phi(e) = 0$, given that $D = 10$ and other delays are 0. However, the clustering in Figure 4(a) is critical under $\phi = 20$ while the one in Figure 4(b) is not. As a result, ϕ cannot be further reduced in Figure 4(a), but can be reduced to 15 in Figure 4(b).

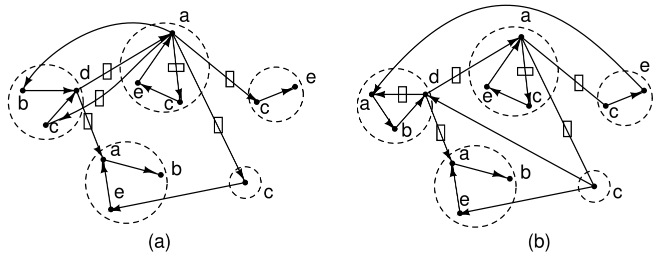

Figure 4: Two clusterings whose arrival time vectors are both equal to T^ϕ under $\phi = 20$ have different maximum cycle ratios.

This example confirms that c^ϕ is not unique. Our goal is to find a non-critical one. To this aim, we define another label $\theta : V \rightarrow Z$, for any clustering c that has no critical cycle under ϕ, such that $\theta(v)$ represents the maximum number of flip-flops on any critical path in E_c terminating at v. Like label t, we also extend the domain of θ to V_c for the ease of the presentation. For all $v \in V_c$ and all $(u, v) \in E_c$, recursively define

$$\theta(v) = \begin{cases} 0, & \text{all } (u,v)\text{'s are non-critical under } \phi \\ \max\big(\theta(u) + w(u,v)\big), & (u,v) \text{ is critical under } \phi \end{cases} \quad (7)$$

The fact that c has no critical cycle under ϕ ensures that θ is well-defined. Let Θ denote the vector

$$\Theta = (\theta(1), \theta(2), \ldots, \theta(|V|)).$$

Another transformation $\mathcal{H} : (Z^{|V|}, \Re) \rightarrow Z^{|V|}$ is defined as follows.

Once T^{ϕ} is obtained, we set $\theta(v) = 0$, $\forall v \in V$, and denote the vector as $\perp_\theta$. Then, for all $(u, v) \in E$ with $t^{\phi}(v) = t^{\phi}(u) + d(u, v) + d(v) - w(u, v)\phi$, we set

$$\theta(v) = \max\big(\theta(u) + w(u, v)\big) \tag{8}$$

Let BF_θ denote the above procedure of updating θ by (8). Then the resulting vector can be represented as

$$\Theta_0 = BF_\theta(\perp_\theta, \phi).$$

After that, for all $v \in V$, we start with $c'_v = \{v\}$ and grow c'_v progressively by including one critical input at a time. If there are multiple critical inputs, we choose the one that gives the maximum $\theta(v)$ by (7). The procedure will stop only when either $t'(v) = t^{\phi}(v)$ and $\theta'(v) \leq \theta(v)$, or $|c'_v| = A$. In either case, we have $t'(v) = t^{\phi}(v)$ since we start with the least fixpoint T^{ϕ}. If $\theta'(v) < \theta(v)$, we keep $\theta(v)$, c_v unchanged; otherwise, we update $\theta(v)$ and c_v with $\theta'(v)$ and c'_v respectively. The resulting $\theta(v)$ is denoted as $\mathcal{H}_v(\Theta, \phi)$.

Define $\mathcal{H}(\Theta, \phi)$ as the vector when all the $\mathcal{H}_v(\Theta, \phi)$'s, $\forall v \in V$, are applied once, followed by BF_θ to ensure (8), expressed as

$$\mathcal{H}(\Theta, \phi) \overset{\triangle}{=} BF_\theta\Big(\big(\mathcal{H}_1(\Theta, \phi), \mathcal{H}_2(\Theta, \phi), ..., \mathcal{H}_{|V|}(\Theta, \phi)\big), \phi\Big).$$

We have the following theorem for $\mathcal{H}$.

Theorem 2 *$\mathcal{H}$ is an order-preserving transformation. A feasible ϕ is not optimal if and only if the least fixpoint Θ^{ϕ} of $\mathcal{H}$ under ϕ exists. Applying iterative method on $\mathcal{H}$ will converge to Θ^{ϕ} with a feasible and non-critical clustering c^{ϕ} under ϕ in at most $|V| - 1$ iterations.*

The next corollary provides a criterion to certify the optimality of ϕ.

Corollary 2.1 *If $\mathcal{H}$ does not converge after $|V|-1$ iterations, then the current feasible ϕ is optimal.*

We then present our algorithm in Figure 5 to find the optimal ϕ. It first computes a feasible ϕ by treating each vertex as a cluster, and computes a lower bound ϕ_{lb} of ϕ by Lemma 1. After that, it iteratively pushes ϕ down by constructing a non-critical clustering c^{ϕ} under ϕ and reducing ϕ to the maximum cycle ratio of c^{ϕ} by Howard's algorithm. The failure of $\mathcal{H}$ to converge in $|V| - 1$ iterations or the fact that ϕ is reduced to its lower bound immediately certifies the optimality of the current feasible ϕ.

The correctness of the algorithm is stated in the following theorem.

Theorem 3 *The algorithm in Figure 5 terminates with the optimal ϕ.*

5.4 Speed-up techniques

In our implementation, we find that $\mathcal{H}$ converges to the least fixpoint very quickly if it has one. However, if $\mathcal{H}$ has no fixpoint, the time for $|V| - 1$ number of iterations to complete is relatively long. We need other criteria to detect the optimality efficiently. The following lemma provides one.

Lemma 6 *$\big(\exists v \in V : \theta(v) > |V|N_{\mathrm{ff}}\big)$ implies that the current feasible ϕ is optimal, where N_{ff} is the maximum number of flip-flops on any acyclic path in G.*

```
Algorithm Optimal clustering
Input: A directed graph $G = (V, E)$, $A$, $D$.
Output: A clustering $c^{\mathrm{opt}}$ with $\phi^{\mathrm{opt}}$.

 $c_v^{\mathrm{opt}} \leftarrow c_v \leftarrow \{v\}$, $\forall v \in V$;
 $\phi^{\mathrm{opt}} \leftarrow \phi \leftarrow$ maximum cycle ratio of $G_c$;
 $\phi_{\mathrm{lb}} \leftarrow$ maximum cycle ratio of $G$;
 Do
   $SUCCESS \leftarrow 1$;
   If $(\phi > \phi_{\mathrm{lb}})$ then
      $\mathrm{T} \leftarrow \mathrm{T}_0 \leftarrow BF(\perp, \phi)$;
      While $\big(\mathrm{T} \neq \mathcal{L}(\mathrm{T}, \phi)\big)$ do
         $\mathrm{T} \leftarrow \mathcal{L}(\mathrm{T}, \phi)$;
      $\Theta_0 \leftarrow BF_\theta(\perp_\theta, \phi)$;
      $iter = 0$;
      While $(iter \leq |V| - 1)$ do
         $\Theta_{iter+1} \leftarrow \mathcal{H}(\Theta_{iter}, \phi)$;
         $iter \leftarrow iter + 1$;
      If ($\mathcal{H}$ converged) then
         $\phi \leftarrow$ maximum cycle ratio of $G_{c^\phi}$;
         $c^{\mathrm{opt}} \leftarrow c^{\phi}$; $\phi^{\mathrm{opt}} \leftarrow \phi$; $SUCCESS \leftarrow 0$;
 While $\big(\neg(SUCCESS)\big)$;
 Return $c^{\mathrm{opt}}$ and $\phi^{\mathrm{opt}}$;
```

Figure 5: Pseudocode of optimal clustering algorithm

In addition, we maintain a vertex pointer $m : V \rightarrow V \cup \{\emptyset\}$, where $\emptyset$ is the reset assignment, and define it as follows. m is reset each time the iterative method is applied on $\mathcal{H}$ for a particular ϕ. After that, for all $v \in V$, we assign $m(v) = u$ if $\theta(v)$ is increased by (8) on $(u, v) \in E$, or by (7) where u will always be a critical input of c_v under ϕ. Intuitively, m captures the fact that any increase of $\theta\big(m(v)\big)$ will be propagated to $\theta(v)$, for all $v \in V$ with $m(v) \neq \emptyset$. It enables us to conclude the following result.

Lemma 7 *If m pointers form a cycle, then the current feasible ϕ is optimal.*

6 Implementation details

6.1 Implementation of $\mathcal{L}_v$ and $\mathcal{H}_v$

Our implementation of $\mathcal{L}_v$ is similar to Cong *et al.* [9]. To characterize critical inputs, we introduce another label $\delta : V \rightarrow \Re$ and define it as follows. Before the construction of c'_v, we assign $\delta(u)$ with $-\infty$ for all $u \neq v$ in V and $\delta(v)$ with 0. At each time, the vertex $u \in i'_v$ with the largest $t(u) + \delta(u)$ is identified. If $t(u) + D + \delta(u) \leq t(v)$, the construction is completed. Otherwise, we put it in c'_v and update $\delta(x)$ with $\max\big(\delta(x),\ \delta(u) + d(u) + d(x, u) - w(x, u)\phi\big)$, for each fanin x of u that becomes an input of c'_v. This procedure will iterate until either $|c'_v| = A$ or all input $u \in i'_v$ has $t(u) + D + \delta(u) \leq t(v)$.

To validate the above procedure, we need to show that it is equivalent to $\mathcal{L}_v(\mathrm{T}, \phi)$, or equivalently, to show that it can always identify the critical input of c'_v. This is fulfilled by the next lemma and corollary.

Lemma 8 *For all $u \in c'_v$, $\delta(u) = \Delta(u, v, \phi)$.*

Corollary 8.1 *The vertex $u \in i'_v$ with the largest $t(u) + D + \delta(u) \geq t(v)$ is the critical input of c'_v.*

The pseudocode for computing $\mathcal{L}_v(\mathrm{T}, \phi)$ is given in Figure 6. It employs a heap Q for bookkeeping the vertices $u \in i'_v$

whose $t(u) + D + \delta(u) > t(v)$. At each iteration, it puts in c'_v the input $u \in Q$ with the largest $t(u) + D + \delta(u)$ and updates $\delta(x)$ for each fanin of u that becomes an input of c'_v. In our implementation, we choose Fibonacci heap [10] for Q. Likewise, the pseudocode for $\mathcal{H}_v(\mathrm{T}, \phi)$ can be similarly derived.

```
Input:  $G = (V, E)$, $A$, $D$, $\mathrm{T}$, $\phi$, $v \in V$.
Output:  $t'(v)$, $c'_v$.

 $\delta(u) \leftarrow -\infty$, $\forall u \in V$; $\delta(v) \leftarrow 0$;
 $Q \leftarrow \{v\}$; $c'_v \leftarrow \emptyset$; $t'(v) \leftarrow t(v)$;
 While $\big((Q \neq \emptyset) \wedge (|c_v| < A)\big)$ do
    $u \leftarrow$ extract from $Q$ with max $t(u) + \delta(u)$;
    $c'_v \leftarrow c'_v \cup \{u\}$;
    For $e = (x, u) \in E$ with $x \notin c'_v$ do
       $\delta(x) \leftarrow \max\big(\delta(x), \delta(u) + d(u) + d(x,u) - w(x,u)\phi\big)$;
       If $\Big((x \notin Q) \wedge \big(t(x) + D + \delta(x) > t(v)\big)\Big)$ then
          $Q \leftarrow Q \cup \{x\}$;
 $t'(v) \leftarrow$ max $t(u) + D + \delta(u)$ in $Q$, if $Q \neq \emptyset$;
 Return $t'(v)$ and $c'_v$;
```

Figure 6: Pseudocode of $\mathcal{L}_v(\mathrm{T}, \phi)$

6.2 Variations of $\mathcal{L}$ and $\mathcal{H}$

In Section 5.2, $\mathcal{L}(\mathrm{T}, \phi)$ is obtained by applying all the $\mathcal{L}_v$'s, $\forall v \in V$, once followed by Bellman-Ford's algorithm. In our implementation, all the $\mathcal{L}_v(\mathrm{T}, \phi)$'s are not computed at the same time. Intuitively, if previously computed $\mathcal{L}_v$'s can be taken into account in later computations of others, the convergence rate may be accelerated.

This motivates our study on a variation of $\mathcal{L}$, in which later computations of $\mathcal{L}_v$'s are based on previously computed ones, and each computation of $\mathcal{L}_v$ is followed by Bellman-Ford's algorithm. Let $\mathcal{J}_v(\mathrm{T}, \phi)$ denote the vector after $t(v)$ is updated with $\mathcal{L}_v(\mathrm{T}, \phi)$, that is,

$$\mathcal{J}_v(\mathrm{T}, \phi) = \big(t(1), ..., t(v-1), \mathcal{L}_v(\mathrm{T}, \phi), t(v+1), ..., t(|V|)\big).$$

Define

$$\bar{\mathcal{L}}(\mathrm{T}, \phi) \overset{\triangle}{=} BF\Big(\mathcal{J}_{i_{|V|}}\big(...BF\big(\mathcal{J}_{i_1}(\mathrm{T}, \phi), \phi\big), ..., \phi\big), \phi\Big),$$

where $i_1, ..., i_{|V|} \in V$. It can be seen that different evaluation orders of V give different $\bar{\mathcal{L}}$'s. However, they all satisfy the following relation.

Lemma 9 *For any* $\mathrm{T} \leq \mathrm{T}^{\phi}$ *satisfying (1) and (6) under a feasible* ϕ *and any evaluation order of* V, $\mathcal{L}(\mathrm{T}, \phi) \leq \bar{\mathcal{L}}(\mathrm{T}, \phi) \leq \bar{\mathcal{L}}(\mathrm{T}^{\phi}, \phi) = \mathrm{T}^{\phi}$.

As a corollary, the next result ensures that we can apply iterative method on $\bar{\mathcal{L}}$ to reach T^{ϕ}.

Corollary 9.1 *If* T^{ϕ} *exists under* ϕ*, applying iterative method on* $\bar{\mathcal{L}}$ *will converge to* T^{ϕ} *in at most* $|V| - 1$ *iterations, independent of the evaluation order of* V *at each iteration.*

Likewise, a variation of $\mathcal{H}$ can be similarly defined.

6.3 Reduced clustering representation

It was shown in [11] that Howard's algorithm was by far the fastest algorithm for maximum cycle ratio computation. Given a clustered circuit $G_c = (V_c, E_c)$ with edge delays and weights specified, Howard's algorithm finds the maximum cycle ratio in $O(N_c|E_c|)$ time, where N_c is the product of the out-degrees of all the vertices in V_c. Since vertex replication is allowed, N_c and $|E_c|$ could be $|V|N$ and $|V||E|$ respectively, where N is the product of the out-degrees of the vertices in V.

To reduce the complexity, we propose a reduced clustering representation. For each cluster, we use edges from its inputs to its output (root) to represent the paths between them such that the delay and weight of an edge correspond to the delay and weight of an acyclic input-to-output path. Figure 7 shows the reduced representation of the clustered circuit in Figure 3(b).

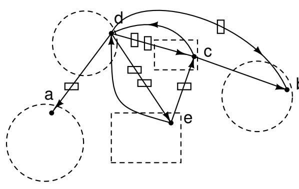

Figure 7: The reduced clustering representation of Figure 3(b).

Let ϕ_c^r denote the maximum cycle ratio of the reduced representation for clustering c. The following lemma formulates the relation among ϕ_c, ϕ_c^r and the lower bound ϕ_{lb} defined in Lemma 1.

Lemma 10 *For any clustering* c, $\phi_c = \max(\phi_c^r, \phi_{\mathrm{lb}})$.

One benefit of the reduced clustering representation is that we can now represent the clustered circuit without explicit vertex replication, that is, using V instead of V_c. Let $G_c^r = (V, E_c^r)$ denote the reduced representation for clustering c. We call an edge in G_c^r *redundant* if its removal will not affect the maximum cycle ratio of G_c^r. The following lemma provides a criterion to prune the redundant edges so that Howard's algorithm can find the maximum cycle ratio of G_c^r more efficiently.

Lemma 11 *Let* c *denote a feasible clustering under* ϕ, E_c^r *denote its reduced representation,* e_1 *and* e_2 *denote two edges from* $u \in V$ *to* $v \in V$ *in* E_c^r, $d(e_1)$ *and* $d(e_2)$ *denote their delays respectively, and* $w(e_1)$ *and* $w(e_2)$ *denote their weights respectively. If* $w(e_1) \geq w(e_2)$ *and* $d(e_1) - w(e_1)\phi \geq d(e_2) - w(e_2)\phi$*, then* e_2 *can be pruned.*

In our implementation, we employ another two parameters $\mathrm{pd} : V \to \{R\}$ and $\mathrm{pw} : V \to \{Z\}$ to record the path delays and weights from the inputs of a cluster to its output, respectively. More specifically, we set $\mathrm{pw}(u) = \mathrm{pd}(u) = \emptyset$, $\forall u \in V$, before $\mathcal{L}_v(\mathrm{T}, \phi)$ is about to be carried out for some $v \in V$. After that, whenever a vertex u is put in c'_v, we compute the pd and pw values of its preceding vertices based on $\mathrm{pd}(u)$ and $\mathrm{pw}(u)$, followed by pruning.

7 Experimental results

We implemented the algorithm in a PC with a 2.4 GHz Xeon CPU, 512 KB 2nd level cache memory and 1GB RAM. To compare with the algorithm in [19], we used the same test files, which were generated from the ISCAS-89 benchmark suite. For each test case, we introduced a flip-flop with directed edges from each PO to it and from it to each PI so that every PI-to-PO path became a cycle. As in [19], the size and delay of each gate was set to 1, intra-cluster delays were 0, and inter-cluster interconnects had delays $D = 2$.

For each circuit, we tested three size bounds: A is 5%, 10% and 20% of the number of gates. The results are shown in Table 1. Since we approach the minimal maximum-cycle-ratio by gradual reduction in the algorithm, we also report

Table 1: Experimental Results

Circuit	$A = 5\%\|V\|$			$A = 10\%\|V\|$			$A = 20\%\|V\|$			`time/step (s)`	
	ϕ^{opt}	#step	time(s)	ϕ^{opt}	#step	time(s)	ϕ^{opt}	#step	time(s)	[19]	presented
s349	18.6	8	0.11	16.7	13	0.32	16.0	7	0.03	0.19	0.02
s420	14.0	3	0.01	13.0	3	0.01	12.0	2	0.00	0.03	0.00
s635	75.0	4	0.02	70.0	4	0.04	68.0	4	0.08	0.10	0.02
s838	17.0	3	0.03	16.0	2	0.01	16.0	2	0.02	0.13	0.01
s1196	26.0	3	0.04	25.0	3	0.05	24.0	2	0.02	0.08	0.02
s1423	55.0	3	11.65	53.0	2	0.13	53.0	2	0.12	0.65	3.88
s1512	25.0	5	0.35	22.5	8	1.43	22.5	8	1.24	1.22	0.18
s3330	14.5	10	2.78	14.0	10	2.48	14.0	10	2.47	1.07	0.28
s4863	30.3	7	15.64	30.0	5	5.00	30.0	5	5.01	82.30	2.30
s5378	21.0	3	0.76	21.0	3	0.77	21.0	3	0.77	7.58	0.26
s9234	38.0	3	3.69	38.0	3	3.73	38.0	3	3.78	n/a	1.26
s35932	27.0	4	74.19	27.0	4	74.69	27.0	4	73.76	n/a	18.67
s38584	48.0	2	24.32	48.0	2	24.48	48.0	2	24.49	n/a	12.25
arith										11.69 X	1
geo										6.33 X	1

the number of reductions for each scenario of A in column "#step". Column "time(s)" lists the running time in seconds. The obtained ϕ^{opt} matches the result in [19] for all the scenarios of A.

The only running time information given in [19] is the largest running time per step among the three scenarios, which we list in column "time/step (s)" under "[19]". We then compute ours in column "presented". Row "arith" ("geo") gives the arithmetic (geometric) mean of the first 10 circuits. We must note that the running times listed are already scaled to take into account the difference in CPU frequencies between their machine and ours.

It can be seen that our algorithm takes much less time per step than the algorithm in [19]. The improvements are greater for larger circuits. The only exception is "s1423", where the algorithm terminates when $\theta(v) > |V|N_{\text{ff}}$ for some $v \in V$. The average speed-up is around one order of magnitude. In addition, for most of the circuits, our algorithm finds the optimal solution in just a few steps, which is generally less than the number of iterations conducted in a binary search, which are not given in [19].

8 Conclusion

Processing rate, defined as the product of frequency and throughput, is identified as an important metric for sequential circuits. We show that the processing rate of a sequential circuit is upper bounded by the reciprocal of its maximum cycle ratio, which is only dependent on the clustering of the circuit. The problem of processing rate optimization is formulated as seeking an optimal clustering with minimal maximum-cycle-ratio in a general graph. An iterative algorithm is proposed that finds the minimal maximum-cycle-ratio. Since our algorithm avoids binary search and is essentially incremental, it has the potential to be combined with other optimization techniques, such as gate sizing, budgeting, etc., thus can be used in incremental design methodologies [7]. In addition, since maximum cycle ratio is a fundamental metric, the proposed algorithm can be adapted to suit other traditional designs.

References

[1] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *ICCAD*, 1999.

[2] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Latency insensitive protocols. In *CAV*, 1999.

[3] M. R. Casu and L. Macchiarulo. A new approach to latency insensitive design. In *DAC*, 2004.

[4] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu. Retiming with interconnect and gate delay. In *ICCAD*, pages 221–226, 2003.

[5] P. Cocchini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *ICCAD*, pages 268–273, 2002.

[6] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc. IFAC Conf. on Syst. Structure and Control*, Nantes, France, 1998.

[7] J. Cong, O. Coudert, and M. Sarrafzadeh. Incremental CAD. In *ICCAD*, 2000.

[8] J. Cong, T. Kong, and D. Z. Pan. Buffer block planning for interconnect-driven floorplanning. In *ICCAD*, pages 358–363, 1999.

[9] J. Cong, H. Li, and C. Wu. Simultaneous circuit partitioning/clustering with retiming for performance optimization. In *DAC*, pages 460 – 465, 1999.

[10] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms.* MIT Press, 1989.

[11] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio. In *DAC*, pages 37–42, 1999.

[12] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge, 1990.

[13] S. Hassoun and C. J. Alpert. Optimal path routing in single and multiple clock domain systems. In *ICCAD*, pages 247–253, 2002.

[14] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing Synchronous Circuitry by Retiming. In *Advanced Research in VLSI: Proc. of the Third Caltech Conf.*, pages 86–116. Computer Science Press, 1983.

[15] C. Lin, J. Wang, and H. Zhou. Clustering for processing rate optimization. Technical Report TR-NUCAD-2005-02, ECE Department, Northwestern University, 2005.

[16] C. Lin and H. Zhou. Optimal wire retiming without binary search. In *ICCAD*, pages 452–458, 2004.

[17] C. Lin and H. Zhou. Wire retiming for system-on-chip by fixpoint computation. In *DATE*, pages 1092–1097, 2004.

[18] V. Nookala and S. S. Sapatnekar. A method for correcting the functionality of a wire-pipelined circuit. In *DAC*, pages 570–575, 2004.

[19] P. Pan, A. K. Karandikar, and C. L. Liu. Optimal clock period clustering for sequential circuits with retiming. *IEEE TCAD*, 17(6):489–498, June 1998.

[20] D. K. Y. Tong and E. F. Y. Young. Performance-driven register insertion in placement. In *ISPD*, pages 53–60, 2004.

[21] H. Zhou and C. Lin. Retiming for wire pipelining in system-on-chip. *IEEE TCAD*, 23(9):1338–1345, September 2004.