A Graph Theoretic Optimal Algorithm for Schedule Compression in Time-Multiplexed FPGA Partitioning*

Huiqun Liu and D. F. Wong
Department of Computer Sciences, University of Texas at Austin, TX 78712
Email: {hqliu, wong}@cs.utexas.edu

Abstract

This paper presents an optimal algorithm to solve the schedule compression problem, which is an open problem proposed by Trimberger [1] for time-multiplexed FPGA partitioning. Time-multiplexed FPGAs have the potential to dramatically improve logic density by time-sharing logic. Schedule compression is an important step in partitioning for time-multiplexed FPGAs [1,4,9,10] and can greatly influence the quality of the partitioning solution. We exactly solve the schedule compression problem by converting it to a constrained min-max path problem. We further extend our algorithm to minimize the communication cost during schedule compression. Experiments show that our optimal algorithm outperforms the existing heuristics and runs very efficiently.

1 Introduction

Time-multiplexed FPGAs have the potential to dramatically improve logic density by time-sharing logic, and have become an active research area in reconfigurable computing. Several different architectures have been proposed for time-multiplexed FPGAs [1, 2, 3, 4, 6, 7]. These time-multiplexed FPGAs allow the dynamic reuse of the logic blocks and wire segments by having more than one configuration controlling them. Thus the logic blocks and interconnect can be reconfigured during runtime by reading a different configuration which only takes time in the order of nanoseconds.

For a time-multiplexed FPGA, a circuit is partitioned into k sub-circuits (or stages), such that the logic in different stages temporally share the same physical FPGA device by reconfiguration and reuse of the logic blocks and the interconnect (Figure 1). Each stage is called a *micro-cycle* and the k micro-cycles form one *user cycle*. One user cycle should produce the same results on the outputs as would be seen by a non-time-multiplexed device. Thus we can use a small physical device to emulate a virtually large device.

Multi-way partitioning is a new and critical step in time-multiplexed FPGA design. The partitioning solution must not only satisfy the area and pin constraints for a physical device, but also satisfy the precedence constraints among the nodes and the timing constraint for each sub-circuit in order to guarantee the correct execution and performance. The partitioning

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658288.

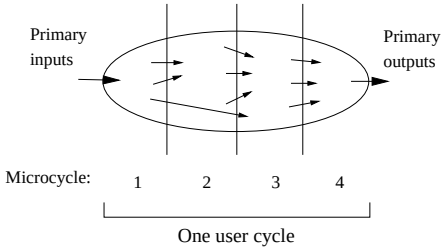

Figure 1: *Partitioning a large logic design into multiple stages to time-share the physical device.*

for time-multiplexed FPGAs can be formulated as a directed acyclic graph (DAG) scheduling problem and the existing approaches include list-scheduling technique [1], enhanced force-directed scheduling [4], and network flow based multi-way partitioning [9, 10].

The partitioning strategies [1, 4, 9, 10] for time-multiplexed FPGAs have the following major steps. First, when the length of the critical paths is larger than the number of stages, more than one level of nodes should be put in one stage. The schedule compression process determines the number of levels in each stage, with the objective of minimizing the critical path width (*i.e.* the maximum number of nodes on the critical paths assigned to any stage). Next, the rest of the nodes on the non-critical paths are scheduled to a proper stage, with the objective of minimizing the communication cost among the stages while satisfying various design constraints. Different approaches [1, 4, 9, 10] have been proposed for this step.

Schedule compression plays an important role in the partitioning for time-multiplexed FPGAs and greatly affects the quality and feasibility of the partitioning process. When the number of levels of nodes on the critical paths is greater than the number of stages in which the design is to be implemented, the schedule compression process merges multiple consecutive levels of nodes on the critical paths into one stage, and decides the depth of each partition.

Trimberger [1] proposed the importance of the schedule compression problem for improving the time-multiplexed FPGA partitioning solutions. He used a heuristic approach and left an open problem whether the schedule compression problem can be optimally solved. [4] also has a similar problem in their enhanced force-directed scheduling process.

In this paper, we present an optimal algorithm to exactly solve the schedule compression problem. We first show that it can be converted to a constrained min-max path problem in a directed acyclic graph (DAG), and then present a polynomial time optimal algorithm.

Besides, we further extend our algorithm to take into account the communication cost during schedule compression. It has been observed that the communication cost is usually a bottleneck for scheduling a design into a time-multiplexed FPGA,

and minimizing the communication cost is the key objective of the partitioning process. Our optimal schedule compression algorithm is extended with the objective of minimizing the maximum communication cost for any stage, and the total communication cost for all the stages.

2 Problem Formulation

Schedule compression is a necessary step in time-multiplexed FPGA partitioning [1, 4, 9, 10]. If the length of the critical path is larger than the number of stages that is allowed to implement the design, the schedule compression process dynamically determines the number of levels in each stage so that the nodes on the critical paths will be evenly distributed over the stages (Figure 2). The maximum number of LUTs on the critical paths in any stage is called the *critical path width*. It is important for the schedule compression process to minimize the critical path width. Otherwise, very tight scheduling constraints can exist if nearly all the LUTs in an FPGA device are consumed by the virtual LUTs on the critical paths. This can lead to designs that can not be scheduled in the given number of stages, thus leading to slower designs or infeasible placement and routing solution. Notice that after the schedule compression process, nodes on the critical paths are fixed to certain stages, further partitioning process is required to partition nodes on the non-critical paths.

Let m be the number of levels on the critical paths in a circuit G and let the i-th level ($1 \leq i \leq m$) of the critical paths has n_i nodes. Let k ($1 < k \leq m$) be the target number of microcycles to implement the design and let s ($1 < s \leq m$) be the maximum number of levels that can be put in one microcycle. Here s is decided by the timing constraint of the design.

The *schedule compression* problem is to divide the m levels into k stages, with each stage having at most s consecutive levels. The objective is to minimize the critical path width, *i.e.* minimizing $max\{w_i \mid 1 \leq i \leq k\}$, where w_i is the total number of nodes assigned in the i-th stage of the schedule compression solution.

The schedule compression problem is also formulated as follows in [4]. Given a sequence of m numbers $n_1, ..., n_m$, divide this sequence into k subsequences with each subsequence having at most s numbers, such that the maximum sum over all the subsequences is minimized. In its enhanced force-directed scheduling approach, [4] uses the concept of distribution graph and needs the schedule compression process to assign depths to the partitions, so that the LUTs are evenly spread over the partitions.

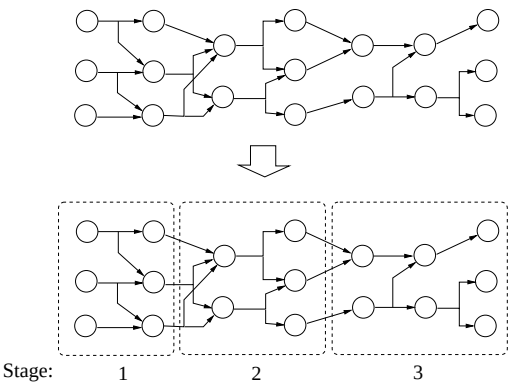

Figure 2: *The schedule compression process merges multiple levels of nodes into one stage.*

Figure 2 shows a schedule compression solution.[1] The length of the critical path is 7, and the number of LUTs on each level of the critical paths is 3, 3, 2, 3, 2, 2, 3 respectively. With $k = 3$ and $s = 3$, we divide the sequence of nodes into 3 subsequences: $\{3, 3\}$, $\{2, 3\}$ and $\{2, 2, 3\}$, each with weight 6, 5, 7. The critical path width equals to $max\{6, 5, 7\} = 7$.

Schedule compression can greatly improve the quality of the partitioning result [1, 4, 9, 10]. It remains an open problem whether there is an optimal solution for the schedule compression problem. [1] used a heuristic which greedily merges two adjacent levels at a time to minimize the critical path width until the number of stages reaches the pre-determined number. [4] used an iterative decision version which takes $O(m^2k)$ time. [9, 10] used a simple heuristic of fixing the number of levels in each stage to be the average of the total number of levels.

In the next section, we show that the schedule compression problem can be optimally solved.

3 Optimal Algorithm for Schedule Compression

Now we present an optimal polynomial time algorithm to solve the schedule compression problem. In Section 3.1, we show how to reduce the schedule compression problem to a constrained min-max path problem in a directed acyclic graph. In Section 3.2 we present a polynomial-time algorithm to optimally solve the constrained min-max path problem.

3.1 Construction of a Directed Acyclic Graph

Given a circuit G, let m be the length of the critical paths, and the i-th level ($1 \leq i \leq m$) of the critical paths has n_i nodes. Let k be the number of stages and s be the maximum number of levels allowed in each stage. We construct a directed acyclic graph $G' = (V', E')$ from G as follows.

1. $V' = \{v_0, v_1, ..., v_m\}$, where v_i ($1 \leq i \leq m$) corresponds to the i-th level in G and v_0 is a dummy node. Each node v_i ($1 \leq i \leq m$) has weight n_i, and v_0 has weight 0.
2. For any two nodes v_i and v_j, if $1 \leq (j - i) \leq s$, then add an edge $v_i \rightarrow v_j$ in E' with weight $w(i, j) = \sum_{k=i+1}^{j} n_k$.

In G', the weight $w(i, j) = \sum_{k=i+1}^{j} n_k$ for edge $v_i \rightarrow v_j$ equals to the total weight of the consecutive levels from $i + 1$ to j. Because a node v_i can only have an edge to v_j when $1 \leq (j - i) \leq s$, the out-degree of each node is no larger than s. So the number of nodes in G' is $m + 1$ and the number of edges is $O(s \cdot m)$.

Definition 1: The *weight* $w(p)$ of a path p: $v_{i_1} \rightarrow ... \rightarrow v_{i_n}$ is the maximum weight of any edge on this path, *i.e.* $w(p) = max\{w(i, j) \mid \text{edge } v_i \rightarrow v_j \text{ is on path } p\}$.

Definition 2: The *constrained min-max path problem* in G' is to find a path of length k from v_0 to v_m with the minimum weight, *i.e.* with minimum $max\{w(i, j) \mid \text{edge } v_i \rightarrow v_j \text{ is on the path }\}$.

Figure 3(b) shows an example of the construction of the directed acyclic graph G' from a circuit G in Figure 3(a). There are seven levels on the critical paths in G, with each level having weight 3, 3, 2, 3, 2, 3, 3 respectively. A directed acyclic graph G' is constructed with 8 nodes (Figure 3(b)) where v_1 to v_7 correspond to levels 1 to 7 in G. The weight of each v_i is the total number of nodes in the i-th level of the critical paths in G. For example, v_1 has weight 3 because there are 3 nodes in level 1 on the critical paths. Node v_0 is a dummy node with weight

[1] Since the schedule compression only merges the nodes on critical paths into stages, in the following examples in the paper, only the nodes on the critical paths in G are shown.

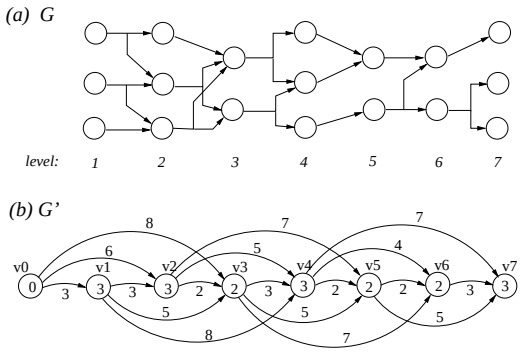

Figure 3: *Building a directed acyclic graph G', node v_i corresponds to the i-th level in G.*

0. The weight of each edge $v_i \rightarrow v_j$ is the sum of weight of the consecutive nodes from v_{i+1} to v_j. For example, the weight on edge $v_0 \rightarrow v_3$ is the sum of weight of v_1, v_2 and v_3, which is $3+3+2=8$.

Given a path p of length k in G', we can find a schedule compresssion solution in G by the following: if edge $v_i \rightarrow v_j$ is in p, then merge nodes from levels $i+1$ to j as one stage in the schedule compression solution. Figure 4 shows an example. The path $v_0 \rightarrow v_2 \rightarrow v_5 \rightarrow v7$ corresponds to a schedule compression solution with each stage having levels $\{1,2\}$, $\{3,4,5\}$ and $\{6,7\}$. We have the following properties of G'.

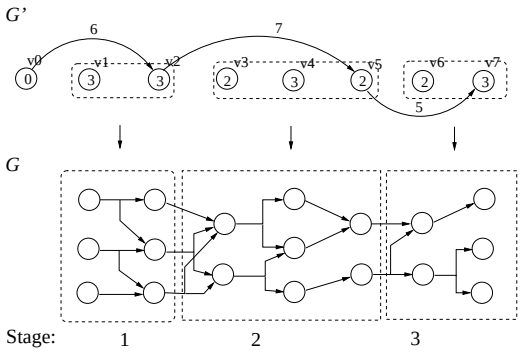

Figure 4: *A path from v_0 to v_7 with length 3 can form a feasible schedule compression. Each edge on this path corresponds to a stage in the schedule compression.*

Lemma 1: *A path of length k from v_0 to v_m in G' corresponds to a feasible schedule compression solution in G.*

Lemma 2: *A path of length k from v_0 to v_m with the minimum weight in G' corresponds to an optimal schedule compression solution in G.*

By Lemmas 1 and 2, we have the following theorem.

Theorem 1: *The schedule compression problem in G can be reduced to the constrained min-max path problem in G'.*

By Theorem 1, the schedule compression problem in G can be optimally solved if we can optimally find a constrained min-max path in G'. In section 3.2, we present an optimal algorithm for finding a constrained min-max path.

3.2 Finding a Constrained Min-Max Path

Now we present a polynomial time optimal algorithm CMP for the constrained min-max path problem. The strategy is similar to the algorithm of finding a shortest path of length k.

Algorithm CMP(G'):
Finding a constrained min-max path in G';
1. **for** $i = 1$ to m **do**
   $d^1(i) = w(0,i)$;
2. **for** $n = 2$ to k **do**
   for $i = 1$ to m **do**
   begin
   $d^n(i) = \infty$;
   for $r = i-s$ to $i-1$ **do**
   if $(max(d^{n-1}(r),\ w(r,i)) < d^n(i))$ **then**
   $d^n(i) = max(d^{n-1}(r),\ w(r,i))$;
   $p^n(i) = r$;
   endif
   end
3. return $d^k(m)$;

In algorithm CMP, array $d^n(i)$ $(1 \le i \le m)$ store the minimum weight of a path from v_0 to node v_i with length n, and array $p^n(i)$ $(1 \le i \le m)$ is used to store the node r if edge $v_r \rightarrow v_i$ is on the path. Let $w(i,j)$ be the weight of edge $v_i \rightarrow v_j$; if there is no edge from v_i to v_j, then $w(i,j) = \infty$. Initially, $n = 1$ and $d^1(i) = w(0,i)$ $(1 \le i \le m)$, which is the minimum weight of a path from v_0 to v_i with length 1. In step 2, the algorithm runs in a total of $k-1$ iterations. In the n-th iteration $(2 \le n \le k)$, $d^n(i)$ $(1 \le i \le m)$ is recursively derived from $d^{n-1}(i)$ as follows:

$$d^n(i) = min\{max(d^{n-1}(r),\ w(r,i)) \mid (i-s) \le r \le (i-1)\}$$

Since $d^{n-1}(r)$ is the minimum weight of a path from v_0 to v_r with length $n-1$, by definition, $max(d^{n-1}(r),\ w(r,i))$ is the weight of a path of length n from v_0 to v_i via edge $v_r \rightarrow v_i$ (Figure 5). We search through all v_r to find a path with the minimum weight (*i.e.* minimum $\{max(d^{n-1}(r),\ w(r,i)) \mid (i-s) \le r \le (i-1)\}$), and the node v_r resulting in the minimum weight is stored in $p^n(i)$. So $d^n(i)$ is the minimum weight of a path from v_0 to v_i with length n. Notice the range for r is $(i-s) \le r \le (i-1)$.

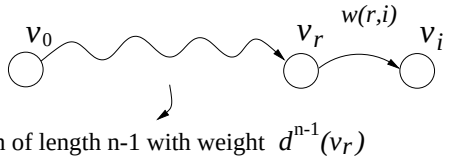

Figure 5: *$d^n(i)$ is the minimum of $max(d^{n-1}(r),\ w(r,i))$ over all r.*

After $k-1$ iterations, $d^k(m)$ is the minimum weight of a path from v_0 to v_m of length k, and the corresponding path can be retrived from the array p. Therefore, $d^k(m)$ is the optimal solution to the constrained min-max path problem.

Lemma 3: *Algorithm CMP finds an optimal solution to the constrained min-max path problem.*

The time complexity for Algorithm CMP is $O(k \cdot s \cdot m)$. By Theorem 1 and Lemma 3, the schedule compression problem can also be optimally solved. We have the following algorithm SC for finding an optimal schedule compression solution. First G' is constructed and a constrained min-max path is found, then the optimal schedule compression solution is obtained correspondingly.

Algorithm *SC*: Optimal Schedule Compression
1. Construct DAG G';
2. Apply Algorithm CMP(G') to find a constrained min-max path p in G';
3. For each edge $v_i \rightarrow v_j$ in path p,
   group nodes in levels from $i+1$ to j into one stage;

To find the length of the critical paths and which nodes are on the critical paths, we can do the following. First, do a AS-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling in G, where each node v is assigned to the earliest possible level $e(v)$ and the latest possible level $t(v)$. If $e(v) = t(v)$, then v is on the critical paths and on level $e(v)$; otherwise if $e(v) < t(v)$, then v is on a non-critical path. The maximum $e(v)$ $(v \in G)$ is the critical path length. After schedule compression assigns nodes on the critical paths to stages, some of the nodes on the non-critical paths are also fixed to certain stages accordingly because of the precedence constraints among the nodes. To take into account the nodes on the non-critical paths, a variation of assigning edge weight in the construction of G' can be the following: for each edge $v_i \rightarrow v_j$, let weight $w(i,j) = |\{v|\ i+1 \leq e(v) \text{ and } t(v) \leq j,\ v \in G\}|$. That is, the weight of edge $v_i \rightarrow v_j$ is the total number of nodes in G which can only be put within levels $i+1$ to j according to the ASAP and ALAP scheduling.

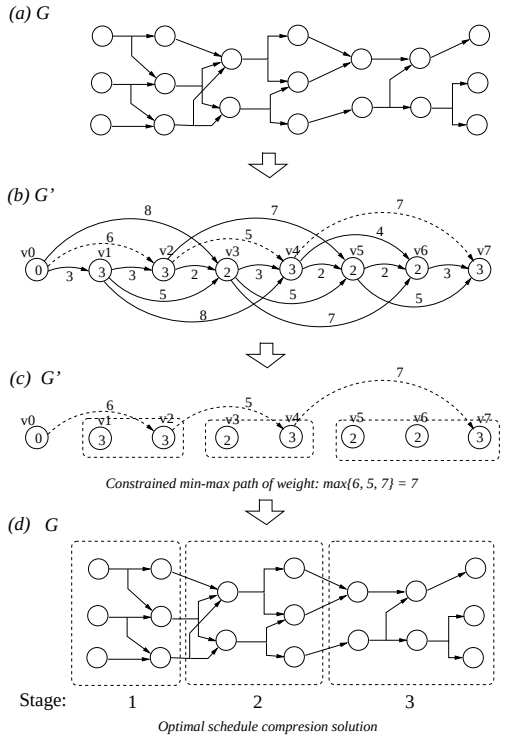

Figure 6: *(a) Circuit G has seven levels of nodes on the critical paths. (b) G' is constructed and path $v_0 \rightarrow v_2 \rightarrow v_4 \rightarrow v_7$ is a constrained min-max path. (c) Each edge on the constrained min-max path corresponds to a stage in the schedule compression. (d) An optimal schedule compression solution.*

Figure 6 shows an example of the process of obtaining the optimal schedule compression. In Figure 6(a), a circuit G has 7 levels on the critical paths, and the 7 levels have 3, 3, 2, 3, 2, 2, 3 number of nodes respectively. Figure 6(b) shows the DAG G' constructed from G, with $k = 3$ and $s = 3$. The path $v_0 \rightarrow v_2 \rightarrow v_4 \rightarrow v_7$ is a constrained min-max path which has weight $max\{6, 5, 7\} = 7$. Figure 6(c) shows that an optimal schedule compression can be derived from the constrained min-max path. Since $v_0 \rightarrow v_2$ is on the path, levels 1 and 2 are grouped into one stage; $v_2 \rightarrow v_4$ is on the path, so levels 3 and 4 are merged into one stage. Similarly $v_4 \rightarrow v_7$ is on the path, so levels 5, 6 and 7 are grouped into one stage. Therefore the three stages will contain levels $\{1, 2\}$, $\{3, 4\}$ and $\{5, 6, 7\}$ respectively. Figure 6(d) shows the optimal schedule compression solution.

4 Schedule Compression for Minimizing Communication Cost

The communication cost is usually a bottleneck in partitioning for time-multiplexed FPGAs. In addition to minimizing the critical path width, it is also necessary and important to minimize the number of interconnections among the stages. For example, a schedule compression solution for minimizing the maximum number of nodes in each stage may cause a larger amount of communication cost for one stage, which will influence both the feasibility and quality of the partitioning, placement and routing result. For circuit G, Figure 7(a) shows an optimal schedule compression solution which minimizes the number of nodes in each stage, but the total number of pins for stage 2 is 6. Figure 7(b) shows a schedule compression solution which minimizes the communication cost and the total number of pins for stage 2 is 4.

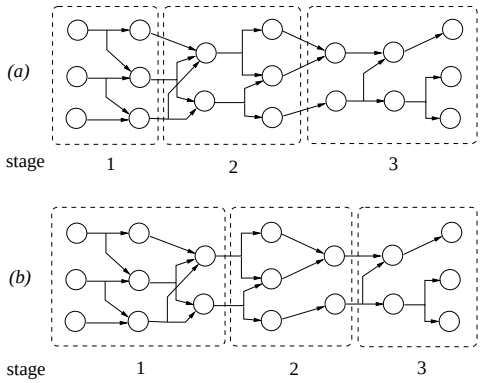

Figure 7: *Schedule compression without considering communication cost can lead to larger communication cost. (a) A schedule compression solution which minimizes the critical path width. (b) A schedule compression solution which minimizes the number of interconnections.*

In this section, we extend the formulation of the schedule compression problem. It is desirable to find a schedule compression which minimizes the maximum communication cost for any stage and also minimizes the total communications cost for all the stages.

Let m be the maximum number of levels in a netlist, and the i-th level on the critical paths has n_i nodes $(1 \leq i \leq m)$. Let k be the target number of stages to implement the design, A be the area limit for each stage, s be the maximum number of levels allowed in one stage. The problem of *schedule compression with minimum communication cost* is to divide the m levels into k stages, with each stage having at most s consecutive levels $(s \leq m)$ and with area no more than A. Let p_i $(1 \leq i \leq k)$ be the communication cost for the i-th stage in the schedule compression solution. The objective is to minimize the maximum communication cost in any of the stages and minimize the total communication cost for all the stages (*i.e.* minimize $max\{p_i \mid 1 \leq i \leq k\}$ and $\sum_{i=1}^{k} p_i$).

We show that this problem can also be solved in polynomial time by converting to the *shortest constrained min-max path* problem, which is to find a constrained min-max path p such that the total edge weight (*i.e.* the sum of weight of all the edges on p) is minimized.

A directed acyclic graph $G'' = (V'', E'')$ is constructed as follows.

1. $V'' = \{v_0, v_1, ..., v_m\}$, where v_i $(1 \leq i \leq m)$ corresponds to the i-th level in the circuit and v_0 is a dummy node. Each node v_i $(1 \leq i \leq m)$ has weight n_i, v_0 have weight 0.

2. Add an edge $v_i \rightarrow v_j$ in E'' if $1 \leq (j-i) \leq s$ and $\sum_{k=i+1}^{j} n_k \leq A$, with weight $w(i,j)$ on edge $v_i \rightarrow v_j$ equals to the communication cost assuming nodes from levels $i+1$ to j form one stage.

When i and j are set, the communication cost for all the nodes in levels $i+1$ to j can be uniquely decided. The weight $w(i,j)$ on the edge $v_i \rightarrow v_j$ measures the communication cost if levels $i+1$ to j are merged in one stage. Notice the calculation of the communication cost depends on the time-multiplexed FPGA architecture. Though different architectures [1, 4] have different models for buffering signals, the communication cost can be uniquely calculated based on a given architecture.

Similar to Lemma 1, by the construction of G'', a path of length k from v_0 to v_m corresponds to a feasible schedule compression solution. The weight of the path corresponds to the maximum communication cost for any stage, and the total edge weight along the path is the total communication cost for all the stages. Similar to Lemma 2, we have the following Lemma 4.

Lemma 4: *A shortest constrained min-max path in G'' corresponds to an optimal solution to the schedule compression with minimum communication cost problem in G.*

We design the following Algorithm SCMP to find a shortest constrained min-max path.

Algorithm SCMP(G''):
Finding a shortest constrained min-max path;
1. **for** $i = 1$ to m **do**
   begin
   $d^1(i) = w(0,i)$;
   $s^1(i) = w(0,i)$;
   end
2. **for** $n = 2$ to k **do**
   for $i = 1$ to m **do**
   begin
   $d^n(i) = \infty$; $s^n(i) = \infty$;
   for $r = i - s$ to $i - 1$ **do**
   begin
   if $(max(d^{n-1}(r),\ w(r,i)) < d^n(i))$ **then**
   $d^n(i) = max(d^{n-1}(r),\ w(r,i))$;
   $s^n(i) = s^{n-1}(r) + w(r,i)$;
   $p^n(i) = r$;
   else if $(max(d^{n-1}(r),\ w(r,i)) = d^n(i))$ **then**
   if $((s^{n-1}(r) + w(r,i)) < s^n(i))$ **then**
   $s^n(i) = s^{n-1}(r) + w(r,i)$;
   $p^n(i) = r$;
   endif
   endif
   end
   end
3. return $d^k(m)$;

In Algorithm SCMP, array $d^n(i)$ and $p^n(i)$ serves the same purpose as in Algorithm CMP. Array $s^n(i)$ is added to store the total edge weight of the constrained min-max path with length n from v_0 to v_i. In the n-th ($2 \leq n \leq k$) iteration, the path with the minimum weight from v_0 to v_i of length n is recorded in $d^n(i)$; if there are multiple paths with different weight, then the one with the minimum total edge weight is selected and the total edge weight is saved in $s^n(i)$. After $k-1$ iterations, a constrained min-max path with the minimum total edge weight is found, with $d^k(m)$ being the weight and $s^k(m)$ being the total edge weight of this path.

After the construction of G'', Algorithm SCMP is applied to find a shortest constrained min-max path. Then an optimal schedule compression solution can be obtained by the following mapping: if edge $v_i \rightarrow v_j$ is on this path, then the nodes in levels $i+1$ to j will form one stage in the schedule compression.

The time complexity of building G'' is $O(n \cdot s \cdot m)$, with n being the total number of nets in the circuit. This is because for finding the weight of each edge $v_i \rightarrow v_j$, the nodes in levels from $i+1$ to j are grouped into one stage and the communication cost is calculated, so in the worst case, every net in G need to be checked whether it passes through the stage. There are a total of $O(s \cdot m)$ edges in G''. Thus, it takes $O(n \cdot s \cdot m)$ time to construct G''. Algorithm SCMP for finding a shortest constrained min-max path takes time $O(k \cdot s \cdot m)$. Therefore the time complexity for the optimal schedule compression with minimum communication cost is $O(n \cdot s \cdot m)$.

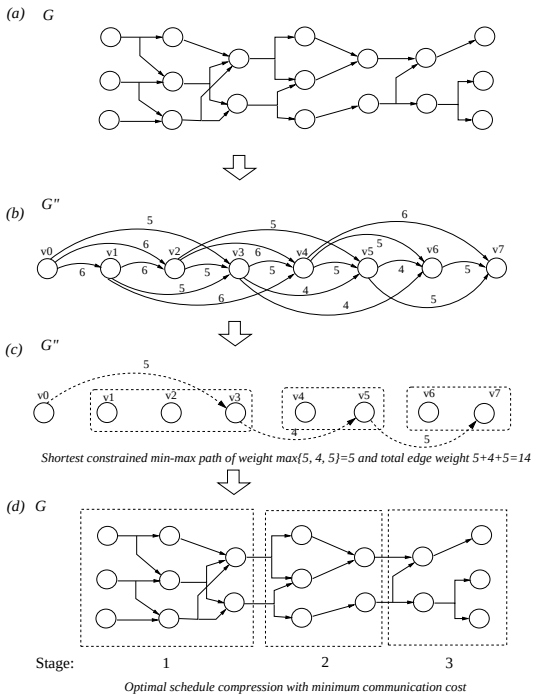

Figure 8: *(a) Circuit G has 7 levels. (b) The directed acyclic graph G'', (c) A shortest constrained min-max path $v_0 \rightarrow v_3 \rightarrow v_5 \rightarrow v_7$ with weight 5, and total edge weight $5+4+5=14$. (d) The corresponding optimal schedule compression with minimum communication cost.*

For the circuit G in Figure 8(a), a directed acyclic graph G'' is constructed in Figure 8(b). The number of stages is 3 ($k = 3$), and the area limit for each stage is 10 ($A = 10$). There are 8 nodes, v_1 to v_7 correspond to levels 1 to 7 respectively. The weight on the edge is calculated based on the Xilinx time-multiplexed FPGA model, where the signals to be passed to a later stage is stored in micro-registers and the number of micro-registers in each stage is a measure of the communication cost. The weight on edge $v_0 \rightarrow v_3$ is 5 because if levels 1 to 3 are merged into one stage, the number of micro-registers for it will be 5. Similarly the weight on edge $v_3 \rightarrow v_6$ is 4 because if the nodes in levels 4 to 6 are merged into one stage, the number of micro-registers for this stage will be 4. Figure 8(c) shows a shortest constrained min-max path of length 3, $v_0 \rightarrow v_3 \rightarrow v_5 \rightarrow v_7$. The weight of this path is $max\{5,4,5\} = 5$, and the total edge weight is $5+4+5 = 14$. The corresponding schedule compression solution with minimum communication cost is shown in Figure 8(d). The three stages have levels $\{1,2,3\}$, $\{4,5\}$ and $\{6,7\}$ respectively, the maximum communication cost for any stage is 5 which equals to the weight of the shortest constrained min-max path, the total communication cost is 14 which equals to the total edge

Table 1: Comparing the schedule compression results in terms of critical path width

Circuits	#Nodes	#Nets	Depth	$k=4$			$k=6$			$k=8$		
				Fix	Optimal	Impv.%	Fix	Optimal	Impv.%	Fix	Optimal	Impv.%
c3540	1038	1016	38	14	13	7.1	10	9	10.0	9	7	22.2
c5315	1778	1655	30	16	12	25.0	12	7	41.7	10	5	50.0
c6288	2856	2824	145	213	172	19.2	147	126	14.3	119	92	22.7
c7552	2247	2140	25	14	10	28.6	10	7	30.0	8	6	25.0
s9234	6098	5846	59	200	112	44.0	160	104	35.0	112	66	41.1
s13207	9445	8653	60	91	64	29.7	75	44	41.3	64	35	45.3
s15850	11071	10385	83	190	156	17.9	120	80	33.3	108	69	36.1
s35932	19880	17830	31	2848	2592	9.0	2304	1920	16.7	1728	1440	16.7
s38417	25589	23845	84	39	26	33.3	33	20	39.4	28	12	57.1
s38584	22451	20719	57	139	127	8.6	101	76	24.8	123	56	54.5
Average						22.2			28.6			37.07

weight along the path.

5 Experimental Results

We implemented the optimal schedule compression algorithm in C++ and experimented on the MCNC Partitioning93 Benchmark circuits. Table 1 shows the characteristic of these circuits. Columns 2, 3 and 4 show the number of nodes, the number of nets and the length of the critical paths in each circuit.

Table 1 compares the optimal schedule compression Algorithm SC with the heuristic (Fix) in [9] which fix the number of levels in each stage to be the average number of levels *i.e.* $\lceil \frac{m}{k} \rceil$. In our experiments, each circuit is compressed into 4, 6 and 8 stages, with the objective of minimizing the critical path width. Columns 6, 9 and 12 show the critical path width of each circuit in our optimal schedule compression solution, and Columns 5, 8 and 11 show the critical path width resulting from the heuristic. Columns 7, 10 and 13 show the improvement of the optimal schedule compression over the heuristic. Our optimal schedule compression algorithm gains more improvement over the heuristic when the number of stages increases.

The experiments show that the critical path width can be greatly reduced when applying the optimal schedule compression algorithm, so that the partitioning result can be improved. After the schedule compression process, the rest of the flexible nodes on the non-critical paths will be partitioned. By minimizing the critical path width in schedule compression, fewer buffers and wires are consumed by nodes on the critical paths, which leave more room to accommodate the other nodes on non-critical paths and improve the final partitioning result. Our experiments also show that the optimal schedule compression can improve the partitioning solution for time-multiplexed FPGAs by an average of 8%.

Since we do not have the implementation of the heuristic in [1], the comparison is not conducted here. However, our algorithm will produce better results than all the other heuristics since it guarantees the optimal schedule compression.

Our algorithm runs very efficiently as expected, as the time complexity of Algorithm SC is $O(s \cdot m)$ with m being the length of the critical paths and s being the maximum number of levels that is allowed in one stage ($s \leq m$). In our experiment, m is no more than a few hundred for each circuit (as shown in Column 4 of Table 1, $m \leq 145$ for the benchmark circuits). It only takes a total of less than 1 second on Pentium Pro 200MHz PC to compute the optimal schedule compression for all the ten circuits.

6 Conclusion

We presented a polynomial time optimal algorithm for the schedule compression problem proposed by Trimberger in [1]. By constructing a directed acyclic graph, the schedule compression problem is reduced to a constrained min-max path problem and can be solved optimally in polynomial time. We further extended our algorithm to solve the problem of schedule compression with minimum communication cost. Experiments show that our optimal algorithm outperforms the existing heuristics and runs very efficiently.

References

[1] Steve Trimberger, "Scheduling Designs into a Time-Multiplexed FPGA", *International Symposium on Field Programmable gate Arrays*, Feb. 1998.

[2] N.B. Bhat, K. Chaudhary and E.S. Kuh, "Performance-oriented fully routable dynamic architecture for a field programmable logic device", Memorandum No. UCB/ERL M93/42, university of California, Berkeley, 1993.

[3] Jeremy Brown, Derrick Chen, et al. "DELTA: Prototype for a first- generation dynamically programmable gate array", Transit Note 112, MIT, 1995.

[4] Douglas Chang and Malgorzata Marek-Sadowska, "Partitioning Sequential Circuits on dynamically Reconfigurable FPGAs", *International Symposium on Field Programmable Gate Arrays*, Feb., 1998.

[5] Douglas Chang and Malgorzata Marek-Sadowska, "Buffer Minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs", *International Symposium on Field Programmable Gate Arrays*, Feb., 1997.

[6] Andre DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century", In *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994.

[7] D. Jones and D.M. Lewis, "A time-multiplexed FPGA architecture for logic emulation", In *IEEE Custom Integrated Circuits Conference*, 1995.

[8] Xilinx, *The Programmable Logic Data Book*, 1996.

[9] Huiqun Liu and D. F. Wong, "Network Flow Based Circuit Partitioning for Time-multiplexed FPGAs", *International Conference on Computer Aided Design*, Nov. 1998.

[10] Huiqun Liu and D. F. Wong, "Circuit Partitioning for Dynamically Reconfigurable FPGAs", *ACM International Symposium on Field Programmable Gate Arrays*, Feb. 1999.