Retiming for Synchronous Data Flow Graphs

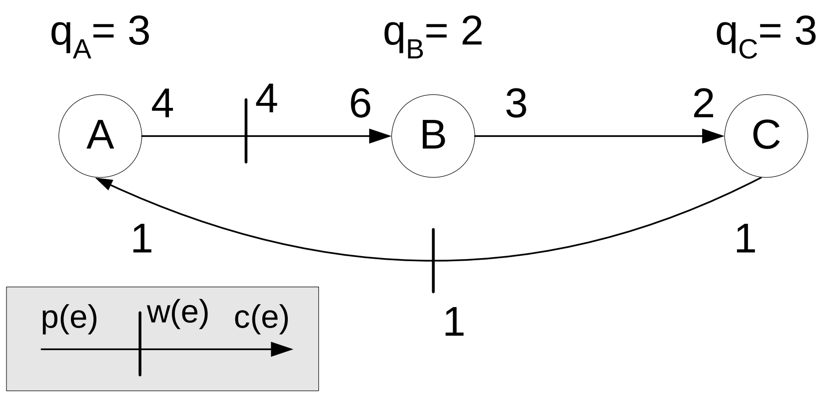
N. Liveris, C.Lin, J. Wang, H. Zhou, P. Banerjee*
Northwestern University, Evanston IL
*University of Illinois, Chicago IL

ASP-DAC 2007, Yokohama, Japan
January 25, 2007

Outline

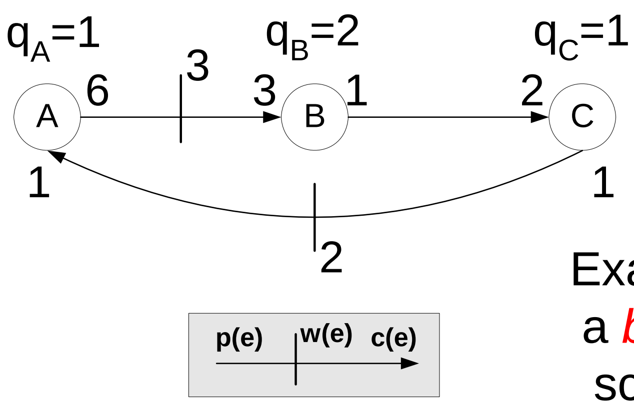
- Intro to SDF and retiming
- Previous work
- First Algorithm
- Improved Algorithm
- Experimental Results
- Conclusion

Synchronous Dataflow Graphs

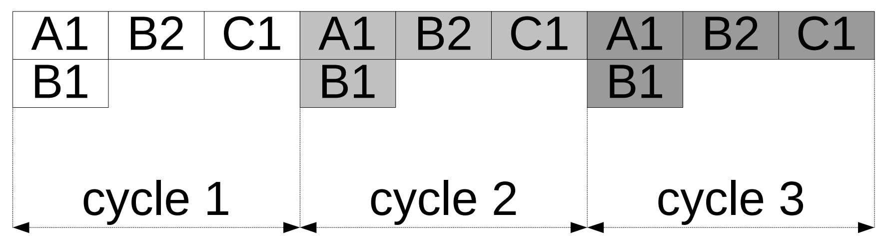


- Each node represents a computation process
  - constant production and consumption rate
  - executed a specific number of times during each complete cycle
- Edge represents a channel between two processes
  - FIFO protocol for tokens
  - initial number of tokens on edge (delays)

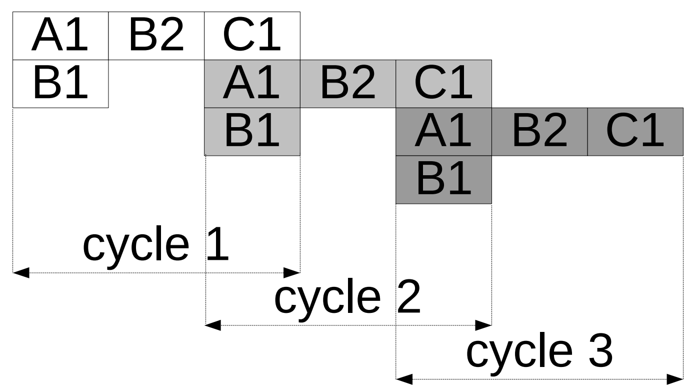
Blocking vs Non-blocking Schedule



Example of a *blocking* schedule



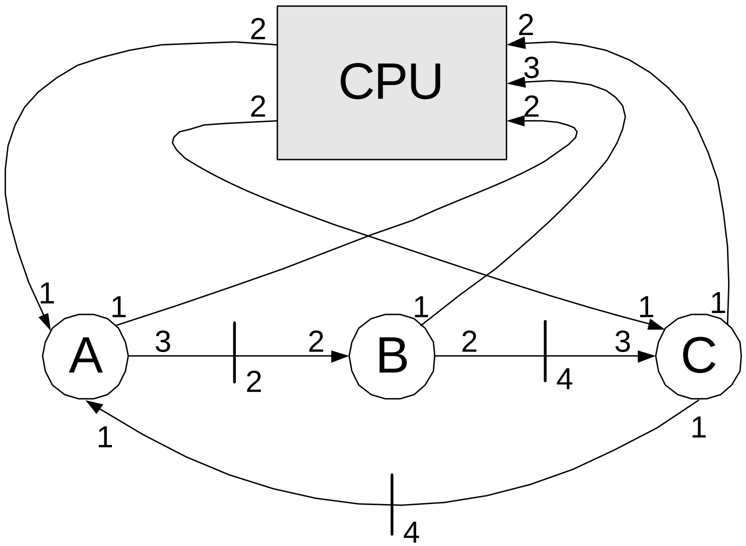
Example of a *non-blocking* schedule



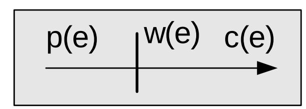
Retiming SDF Graphs

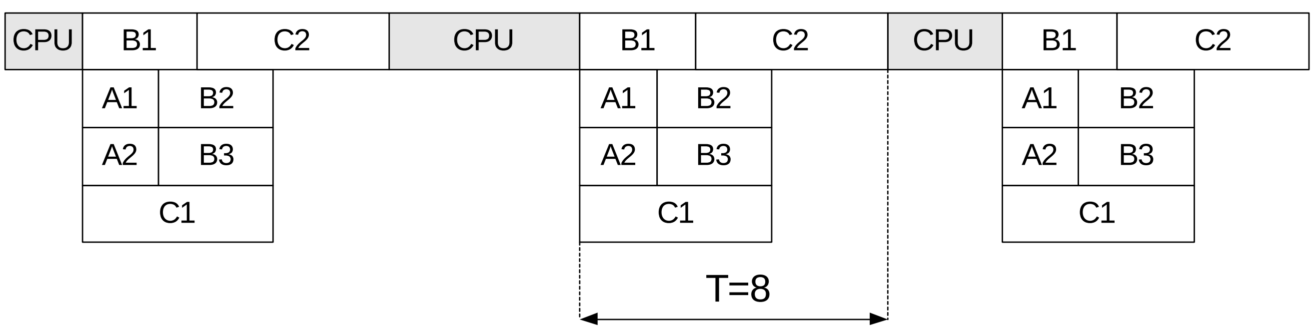
- DSP applications with constant consumption and production data rates and predictable execution time are modeled by SDF graphs
- Some applications whose behavior is determined at run-time or that share resources with high-priority tasks are normally executed on programmable cores
- When data dependencies exist between SDF actors and tasks executed on programmable cores, a non-blocking schedule may not be feasible

Example

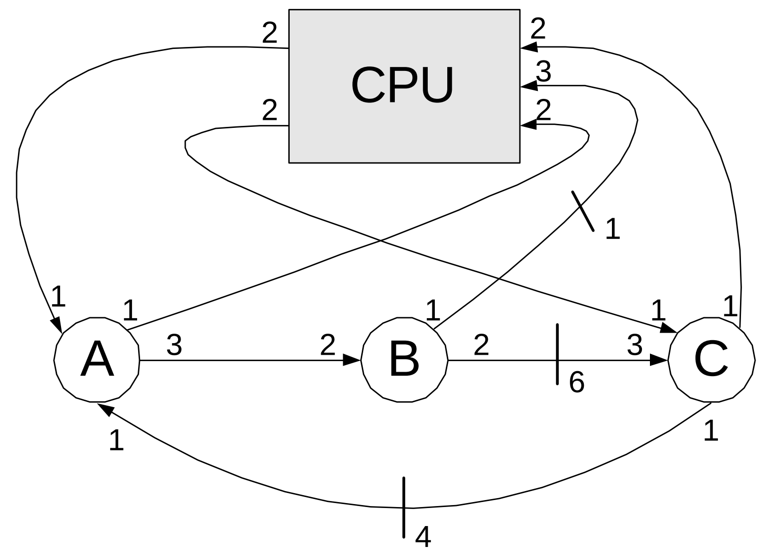


	d	*q*
A	2	2
B	3	3
C	5	2

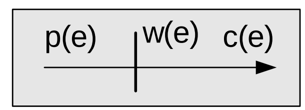


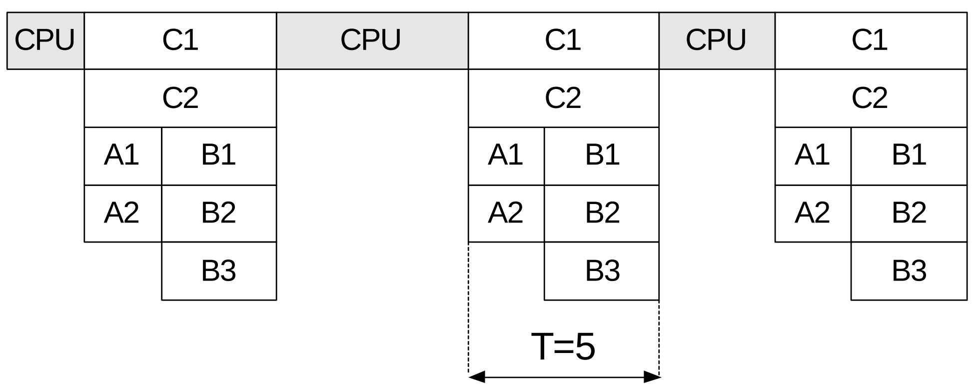


Example – retimed



	d	*q*
A	2	2
B	3	3
C	5	2





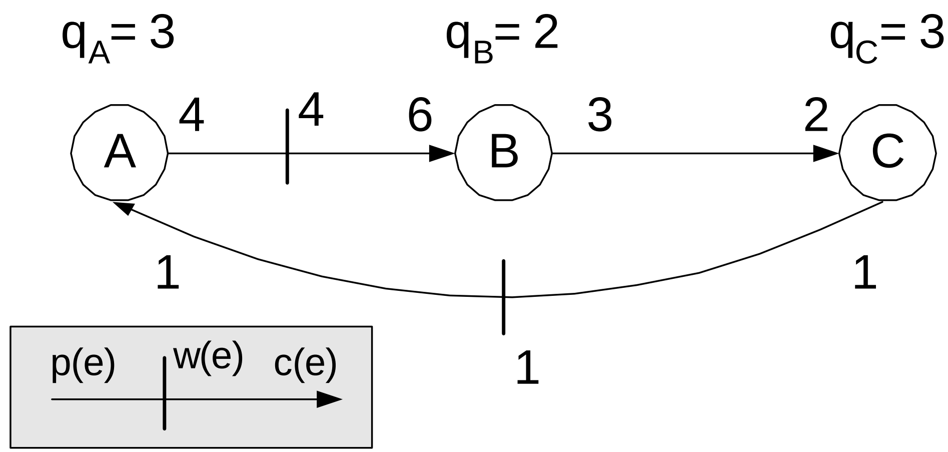
Previous Approach

- T. O’Neil, E. Sha; “Retiming Synchronous Dataflow Graphs to Reduce Execution Time”;IEEE Transaction on Signal Processing, Oct 2001
- Only check whether a given cycle time is feasible
- Computing the maximum path in the EHG (Equivalent Homogenous Graph)
  – a distinct node for each node instance
  – each token transferred on a separate edge
  – p(e)=c(e)=1
  – number of edges $\Sigma_{(u,v) \text{ in } E}$ q(v) c(u,v)
- Selection of node v, whose r(v) will be increased, is based on heuristic
- Termination criteria is not provable

Our Approach

- Computation of max length is done on the SDF graph
  - avoiding expensive generation of EHG
  - avoiding computation for nodes that cannot affect the max length path
- Selection of nodes is justified based on properties
- Algorithm reduces cycle time at each iteration or proves that the cycle time of the iteration is optimal
- Upon termination an optimal solution is generated

Dependence Walk

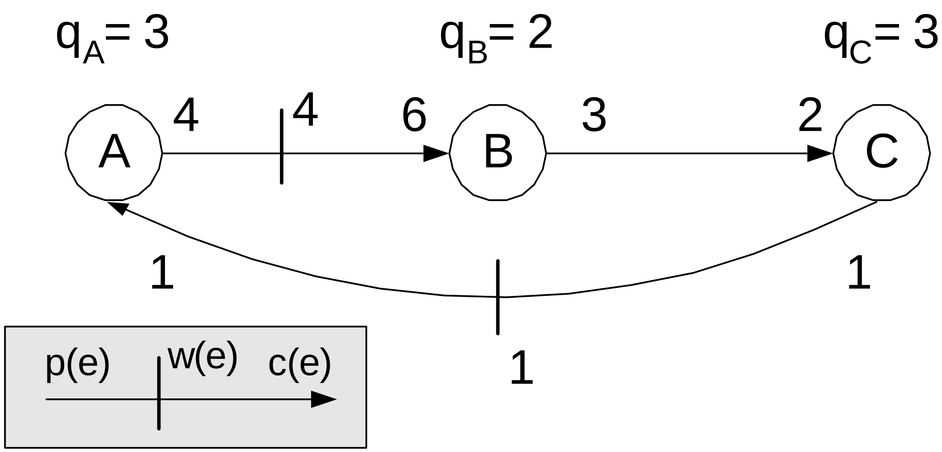


W = (A,1)→(B,1) →(C,1) →(A,2)→(B,2)→(C,3)

(node name, instance number)

Execution of (v_i,l_i) can start only after execution of (v_{i-1},l_{i-1}) has been completed.

Critical Dependence Walk

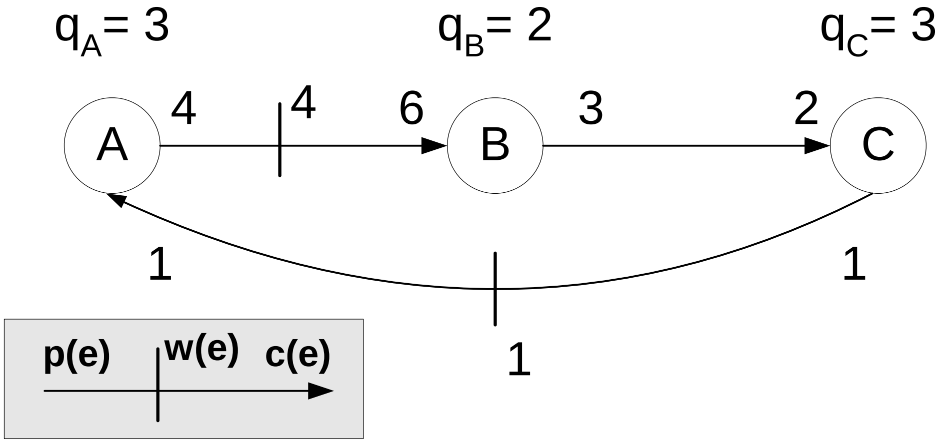


W = (A,1)→(B,1) →(C,1)→(A,2)→(B,2)→(C,3)

(A,1) = (v_0,l_0)

Execution of (v_i,l_i) starts exactly when execution of (v_{i-1},l_{i-1}) completes and (v_0,l_0) starts at the beginning of the period (time 0)

Node Selection

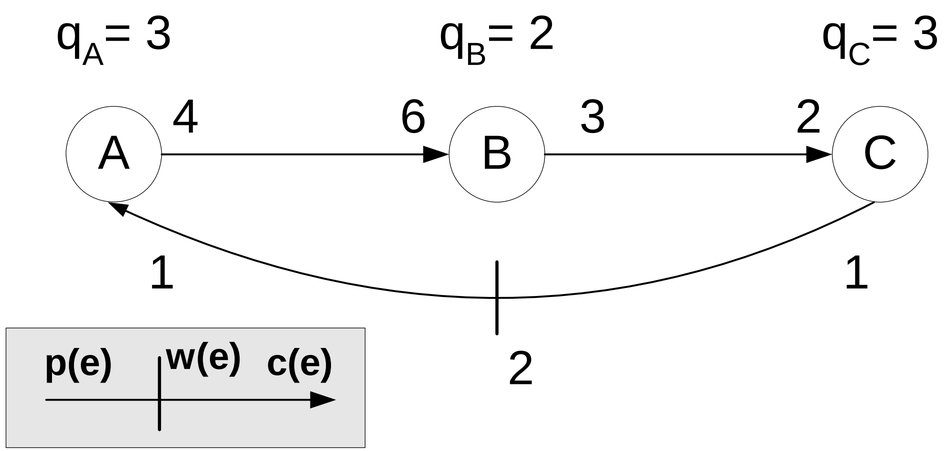


If W is a critical walk, with $t(v_n,l_n)+d_n=T$, then the only way to obtain graph with T' < T is by increasing $r(v_n)$.

W = (A,1) → (B,1) → (C,1) → (A,2) → (B,2) → (C,2) → (A,3)

$(A,3) = (v_n,l_n)$

Retimed Graph



W = (A,2) →(B,1) →(C,1) →(A,3) →(B,2) →(C,3)

In this example the length of W has been reduced after the retiming operation.

Maximum Length Walk Computation

```
proc get_t(v,k,r)
    if (k < 1) then
        return −d(v);
    fi;
    if (t[v,k] ≠ −1) then
        return t[v,k];
    fi;
    maxt ← −1;
    for each (u,v) ∈ E
        l ← ⌈(k·c(u,v) − w_r(u,v)) / p(u,v)⌉;
        t1 ← get_t(u,l) + d(u);
        if (maxt < t1) then
            maxt ← t1;
        fi;
    endfor;
    t[v,k] ← maxt;
    return t[v,k];
```

- Execution of (v_i, l_i) cannot start before execution of (v_{i-1}, l_{i-1}) has finished
- Computing the arrival time of each walk starting from the last instance of each node
- Dynamic programming algorithm (memory function)

Termination Conditions

- It is proven that the algorithm will always find a basic optimal solution, i.e. in the solution there will exist v such that r(v) < q(v)
- Following from the above condition and from the conditions that can trigger an r change:

$$(\forall v : r(v) \leq 2 \cdot q_v \cdot |V|)$$

If any of these conditions are violated, the algorithm cannot improve the best solution found thus far.

First Version of the Algorithm

- Finds last node of a critical walk for which $t(v_n,l_n)+d_n=T$
- Increments $r(v_n)$ ($r'(v_n) = r(v_n) + 1$)
- Recomputes arrival times for the nodes using the dynamic programming algorithm
- Stores solution if $T' < T$
- Continues this process until any of the termination conditions are satisfied
- Worst-case complexity $O(|V|^3|E|q_{ave}^2)$

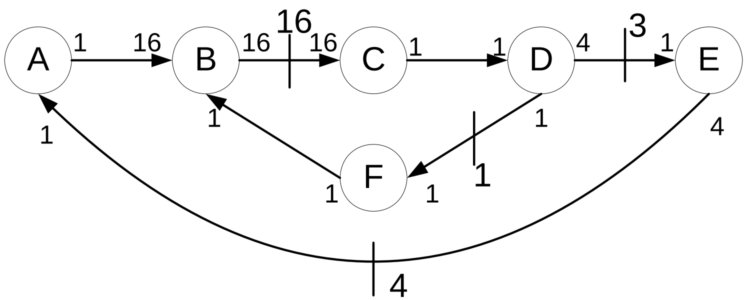
Improved Version

- First version changes the r(v) of one node by 1 and then tries to find critical walk again
  - guarantees that the edge weight will never become negative, but
  - for each r change, arrival times have to be recomputed
- Improved version relaxes the non-negativity constraint for edges, and does more than one change in each iteration
- Mechanism can be used to validate additional constraints for edges

Improved Version

- Maintains two queues:
  - First queue holds the nodes, which require an r-value increase in order for a potential reduction of T to occur
  - Second queue holds edges with negative weights. The r-value of the head of each edge needs to be increased, so that the non-negativity constraint is satisfied
- Arrival times are recomputed only after queues are empty (all necessary r-value increases have occurred)

Execution Snapshot 1



$r^o(A) = 0$ $r(A) = 0$ $t_f(A,q_A) = 2$

$r^o(B) = 0$ $r(B) = 0$ $t_f(B,q_B) = 3$

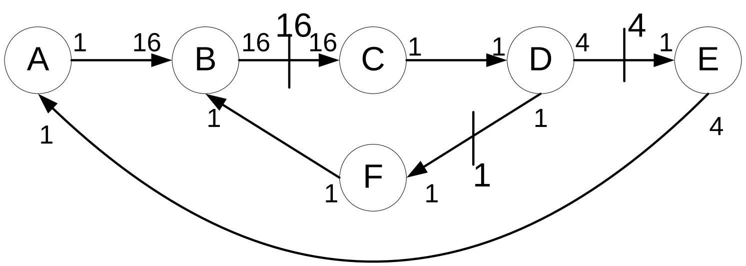
$r^o(C) = 1$ $r(C) = 1$ $t_f(C,q_C) = 2$

$r^o(D) = 0$ $r(D) = 0$ $t_f(D,q_D) = 3$

$r^o(E) = 0$ $r(E) = 3$ $t_f(E,q_E) = 4$ $W=C_1 \twoheadrightarrow D_1 \rightarrow E_4$

$r^o(F) = 0$ $r(F) = 0$ $t_f(F,q_F) = 2$

$T_{step} = 4$ Q1 ={E} Q2 = { }



$r^o(A) = 0$ $r(A) = 0$ $t_f(A,q_A) = 2$

$r^o(B) = 0$ $r(B) = 0$ $t_f(B,q_B) = 3$ $W=E_4 \rightarrow A_{16} \rightarrow B_1$

$r^o(C) = 1$ $r(C) = 1$ $t_f(C,q_C) = 2$

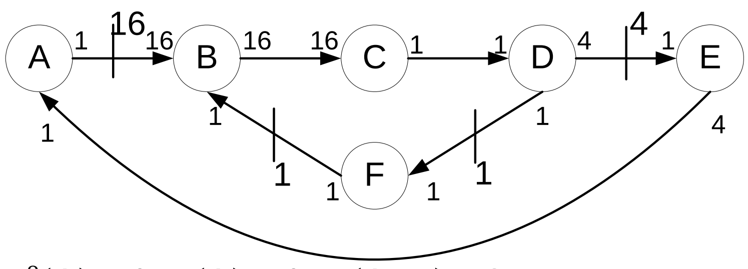
$r^o(D) = 0$ $r(D) = 0$ $t_f(D,q_D) = 3$

$r^o(E) = 4$ $r(E) = 4$ $t_f(E,q_E) = 1$

$r^o(F) = 0$ $r(F) = 0$ $t_f(F,q_F) = 2$

$T_{step} = 3$ Q1={B} Q2 = { }

Execution Snapshot 2



$r^o(A) = 0$ $r(A) = 0$ $t_f(A,q_A) = 2$

$r^o(B) = 0$ $r(B) = 1$ $t_f(B,q_B) = 1$

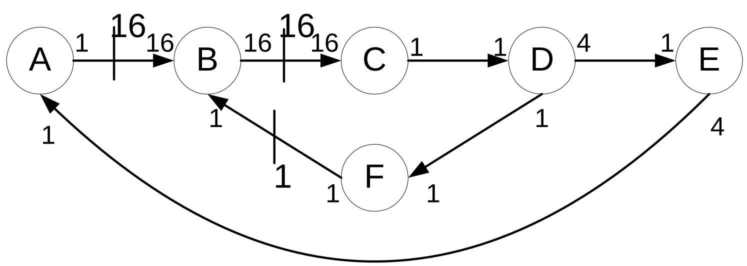
$r^o(C) = 1$ $r(C) = 1$ $t_f(C,q_C) = 3$ $W = B_1 \rightarrow C_1$

$r^o(D) = 0$ $r(D) = 0$ $t_f(D,q_D) = 4$ $W = B_1 \rightarrow C_1 \rightarrow D_1$

$r^o(E) = 4$ $r(E) = 4$ $t_f(E,q_E) = 1$

$r^o(F) = 0$ $r(F) = 0$ $t_f(F,q_F) = 2$

$T_{step} = 3$ $Q1=\{C,D\}$ $Q2 = \{\ \}$



$r^o(A)=0$ $r(A) = 0$ $t_f(A,q_A)=5$ $W=C_1 \rightarrow D_1 \rightarrow E_4 \rightarrow A_{16}$

$r^o(B)=0$ $r(B) = 1$ $t_f(B,q_B)=1$

$r^o(C)=1$ $r(C) = 2$ $t_f(C,q_C)=2$

$r^o(D)=0$ $r(D) = 1$ $t_f(D,q_D)=3$ $W = C_1 \rightarrow D_1$

$r^o(E)=4$ $r(E) = 4$ $t_f(E,q_E)=4$ $W = C_1 \rightarrow D_1 \rightarrow E_4$

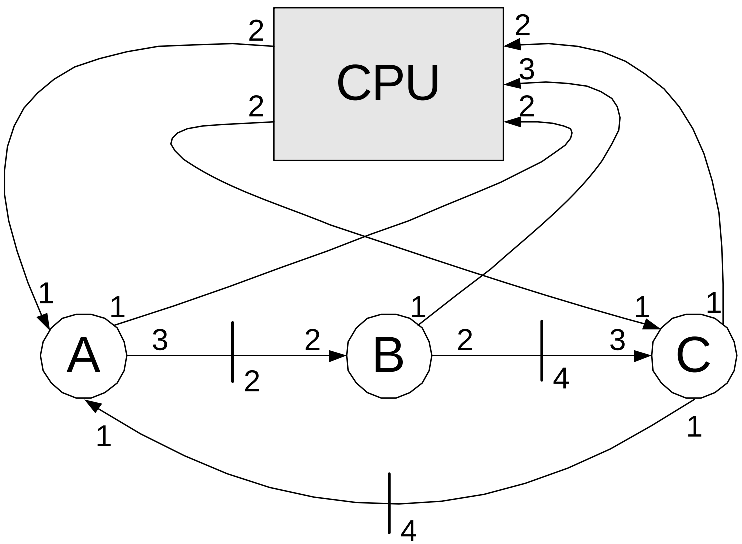
$r^o(F)=0$ $r(F) = 0$ $t_f(F,q_F)=5$ $W = C_1 \rightarrow D_1 \rightarrow F_1$

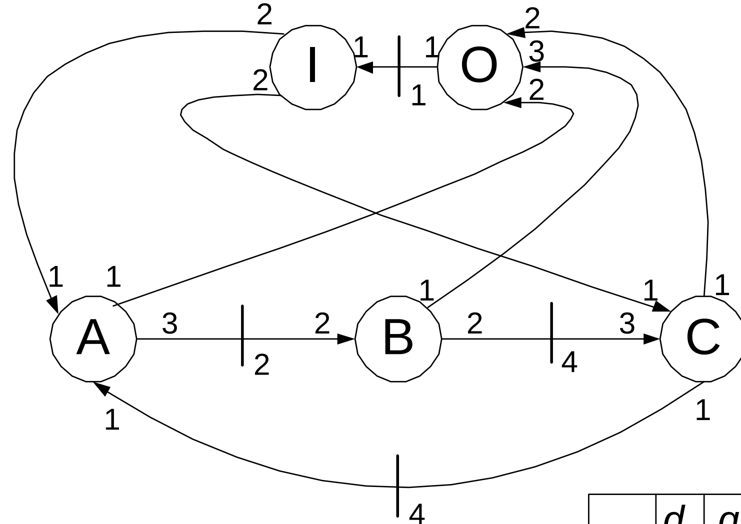
$T_{step} = 3$ $Q1=\{A,D,E,F\}$ $Q2 = \{\ \}$

Experimental Results (q_{max}=32)

Graph	T			execution time (sec)		
	O'Neil's	First	Improved	O'Neil's	First	Improved
*s*27	459	416	416	1.924	0.012	0.060
*s*208.1	834	834	834	2m:50.537	1.287	0.049
*s*298	1083	1027	1027	55m:30.897	2.696	0.095
*s*344	2534	2468	2468	70m:29.472	3.457	0.415
*s*349	1503	1415	1415	8m:18.343	4.140	0.257
*s*382	1312	1273	1273	19m:29.061	5.261	0.344
*s*386	938	806	806	1m:40.775	2.733	0.129
*s*444	1185	888	888	48m:18.215	2.825	0.191
*s*526	2161	2007	2007	120m:00.000	7.796	0.479
*s*641	690	610	610	54.758	9.837	0.534
*s*820	1594	1573	1573	46m:26.437	11.805	0.622
*s*953	1776	1776	1776	5m:26.620	16.650	0.919

Modeling Environment





	d	*q*
A	2	2
B	3	3
C	5	2
I	0	1
O	0	1

Experimental Results

Graph	T		Execution Time (sec)	
	Initial	Final		
*s*27	368	351	0.005	
*s*208.1	1035	852	0.020	
*s*298	1052	742	0.045	*
*s*344	1062	928	0.164	
*s*349	933	833	0.016	
*s*382	951	908	0.021	
*s*386	745	650	0.051	
*s*444	902	882	0.027	
*s*526	1690	1690	0.009	*
*s*641	694	665	0.011	
*s*820	1264	1219	0.032	
*s*953	1558	1558	0.010	

Summary

- Presented two new algorithms for retiming SDF graphs
- Algorithms aim at minimizing the cycle length of the SDF and are optimal
- Improved version is orders of magnitude faster than other approaches

Thank you