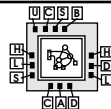


# Symbolic Scheduling Techniques [1]

**Ivan Radivojević, Forrest Brewer**

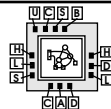
**Department of Electrical and Computer Engineering  
University of California, Santa Barbara, U.S.A.**

- 
1. This work has been supported in part by fellowship donation from Mentor Graphics Corp. and UC-MICRO under project No. 92-019.



## ABSTRACT

*We propose a new exact formulation of resource-constrained control/data-flow scheduling. Unlike current techniques, a solution is generated in which all satisfying schedules are encapsulated in a compressed representation. The technique supports various forms of code motion and extraction of parallelism not explicit in the input description. In addition, the formulation allows development of set-based heuristics enabling application to large scheduling problems. Currently, exact scheduling of arbitrary forward-branching control structures is supported. Control model generalizations are planned.*

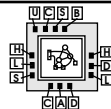


# 1. INTRODUCTION

- ***Operation Scheduling*** -- assignment of operations to bounded time slots of a synchronous system subject to:
  - **Data/control-flow dependencies**
  - **Resource constraints (functional units, busses, registers)**

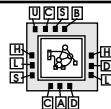
**Additional constraints: timing, synchronization...**

- ***Current Solutions:***
  - **Priority based heuristics**
  - **Integer Linear Programming (ILP) based optimizations**
  - **Symbolic techniques**



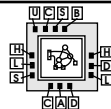
## 1.1. HEURISTIC SCHEDULERS

- **Applicable to wide variety of problems and large CDFGs (e.g. *list scheduling*, *path-based scheduling*, *force-directed scheduling*)**
- **May fail to find solutions in tightly constrained problems**
  - **Often very large pool of candidates satisfying pre-specified criteria**
  - **Typically only one representative is selected**
  - **Cannot recuperate from suboptimal decisions**
  - **Potentially expensive to perform lookahead or to backtrack**
- **Difficult to add new constraints incrementally**



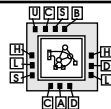
## 1.2. ILP-BASED OPTIMIZATIONS

- **Exact ILP methods [Hwang *et al.*] are computationally very expensive**
- **Implementations:**
  - **remapping of constraints and conversion to relaxed LP models [Gebotys]**
  - **ILP-based heuristics (e.g. *zone partitioning, stepwise expansion*)**
- **Problems:**
  - **no practical model for control-dependent behavior**
  - **sensitive to the number of formulation variables (problem size)**
  - **only a single representative solution**
  - **heuristic approach: subproblems solved optimally, but decisions are local**



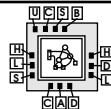
## 1.3. SYMBOLIC TECHNIQUES

- Describe scheduling constraints as *Boolean functions*
- Build *BDD* (Binary Decision Diagram) encapsulating *all* feasible solutions
- Recently introduced:
  1. Radivojević, Brewer [SASIMI'93, HLSS'94, DAC'94, IEICE'95]
    - topic of this presentation...
  2. De Micheli *et al.* [EDAC'94, ICCAD'94]
    - FSM to capture resource/timing/synchronization constraints
    - specification-level formalism restricts code motion
    - potentially exponential number of formulation variables



# MOTIVATION

- Encapsulate *all* feasible schedules
- Why is this advantageous?
  - Constraints can be *incrementally* applied
  - Increased *freedom* in subsequent synthesis steps
  - *Exact* scheduling for *arbitrary* control
  - For very large problems, allows *set-based* high-quality heuristics
- What is needed?
  - Flexible control model that can be treated *systematically*
  - *Efficient* representation to manipulate large data sets

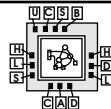


# SYMBOLIC vs. ILP

- Is there any difference?

	constraint type	#solutions	#variables	support for			
				DFGs	CDFGs	cyclic DFGs	heuristics
<b>Symbolic</b>	<i>any Boolean function</i>	<i>all</i>	$O [ (\#cycles) * (\#ops) + (\#cond) ]$	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes (based on sets)</i>
<b>ILP</b>	<i>linear</i>	<i>1</i>	$O [ (\#cycles) * (\#ops) * 2^{(\#cond)} ]$	<i>yes</i>	<i>no control model</i>	<i>yes</i>	<i>yes (locally optimal at subproblem level)</i>

*#cycles* - number of time steps, *#ops* - number of operations, *#cond* - number of conditionals.

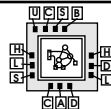


## 2. EXPLOITING IMPLICIT PARALLELISM

- **Ability to extract parallelism not explicit in the input description dramatically increases scheduling quality**

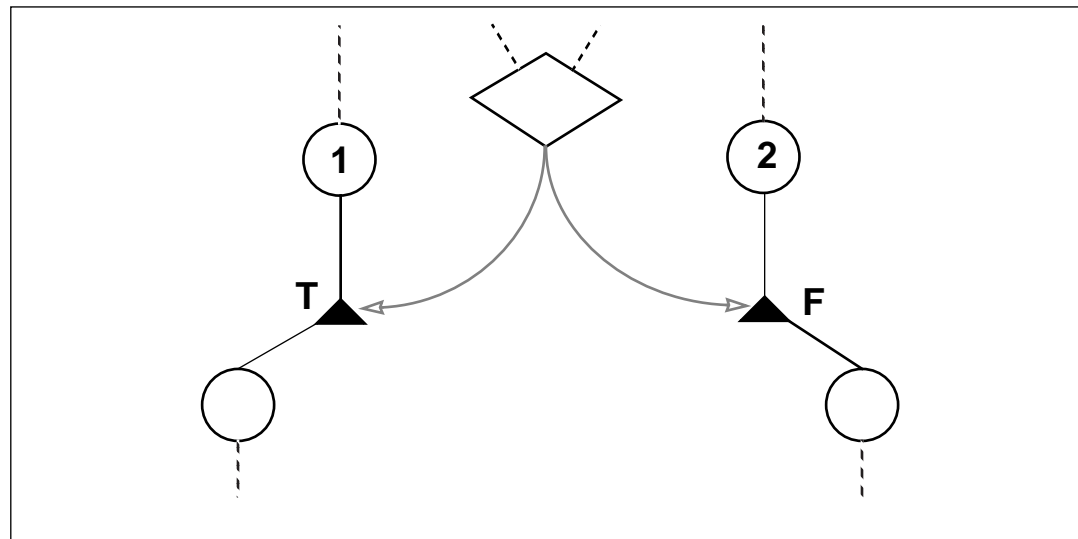
(*e.g.* irredundant operation scheduling, out-of-order execution of conditionals, speculative execution, global treatment of parallel control structures)

- **Problems:**
  - **As the number of control paths increases, it becomes very difficult to keep track of the mutual exclusiveness among the operations**
  - **Impossible to accurately predict resource usage and availability in a static fashion**

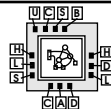


# IRREDUNDANT OPERATION SCHEDULING

- If operation is *redundant* on a particular control path it need not be scheduled there

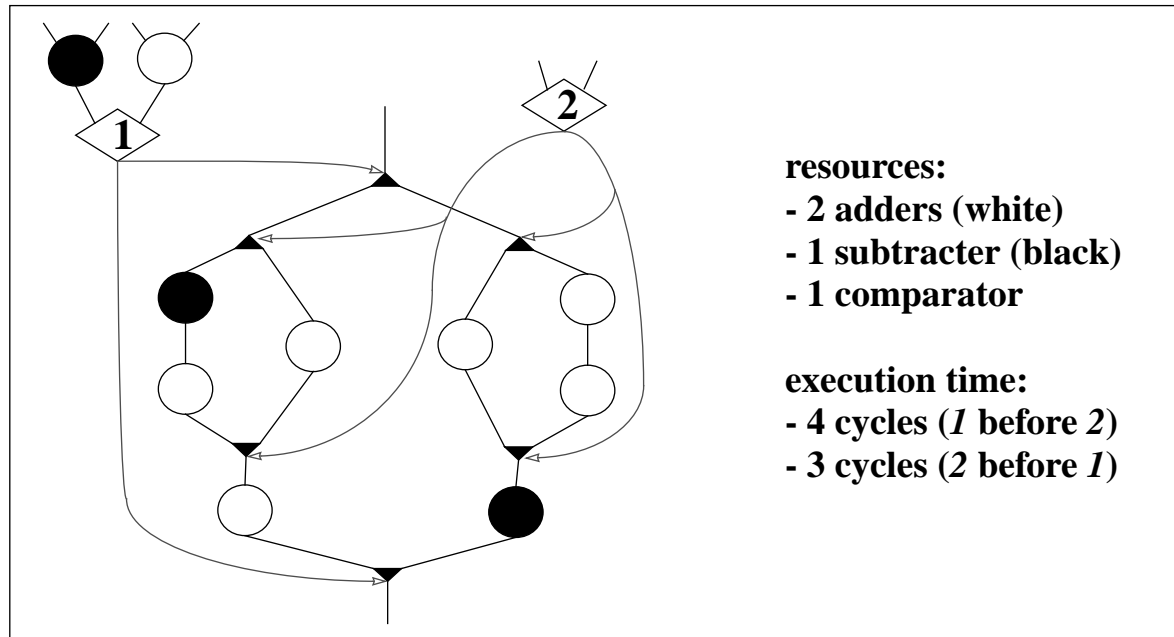


- Operation 1 is redundant on 'False' path and operation 2 is redundant on 'True' path -- this can be used to reduce resource requirements

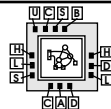


# OUT-OF-ORDER EXECUTION OF CONDITIONALS

- Dynamically resolve execution order of *conditionals* (operations that generate control signals)

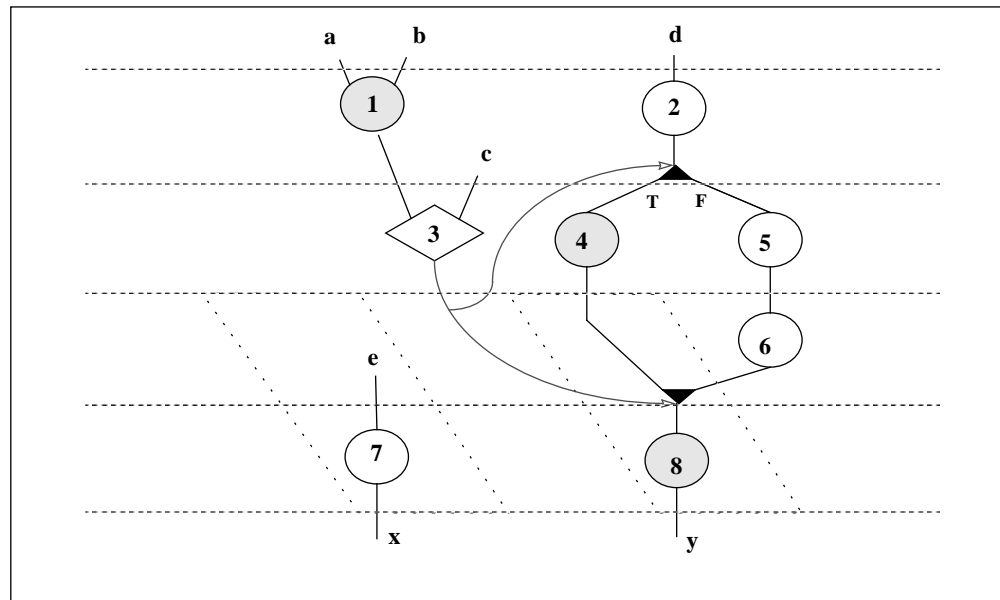


- Faster execution if conditional *2* is evaluated before conditional *1*

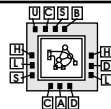


# SPECULATIVE OPERATION EXECUTION

- Operations from branch arcs executed before the condition is known

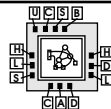


- Improves execution time when there are sufficient resources

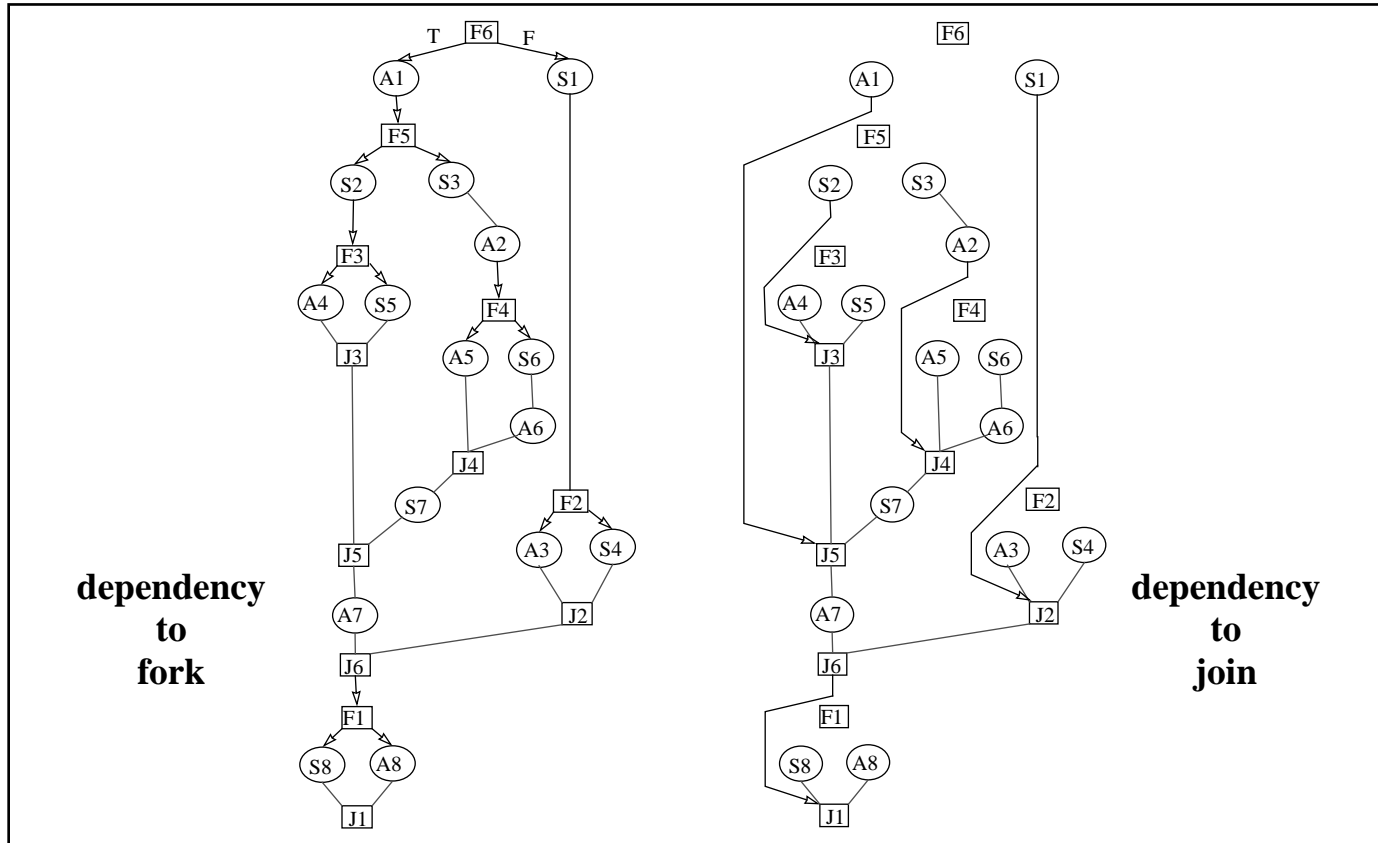


## SPECULATIVE EXECUTION MODEL

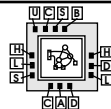
- **Dependency between the conditional and the *fork* node is removed**
- **Dependency between the conditional and the *join* node is enforced**
- ***Restriction*: operations after the join node cannot be executed before the corresponding conditional**  
⇒ **Each operation is scheduled at most once per control path (prevents a potential explosion of operator's instances)**
- **Solved *exactly* [DAC'94]**



## CDFG transformation for MAHA [Parker *et al.*, DAC'86]:

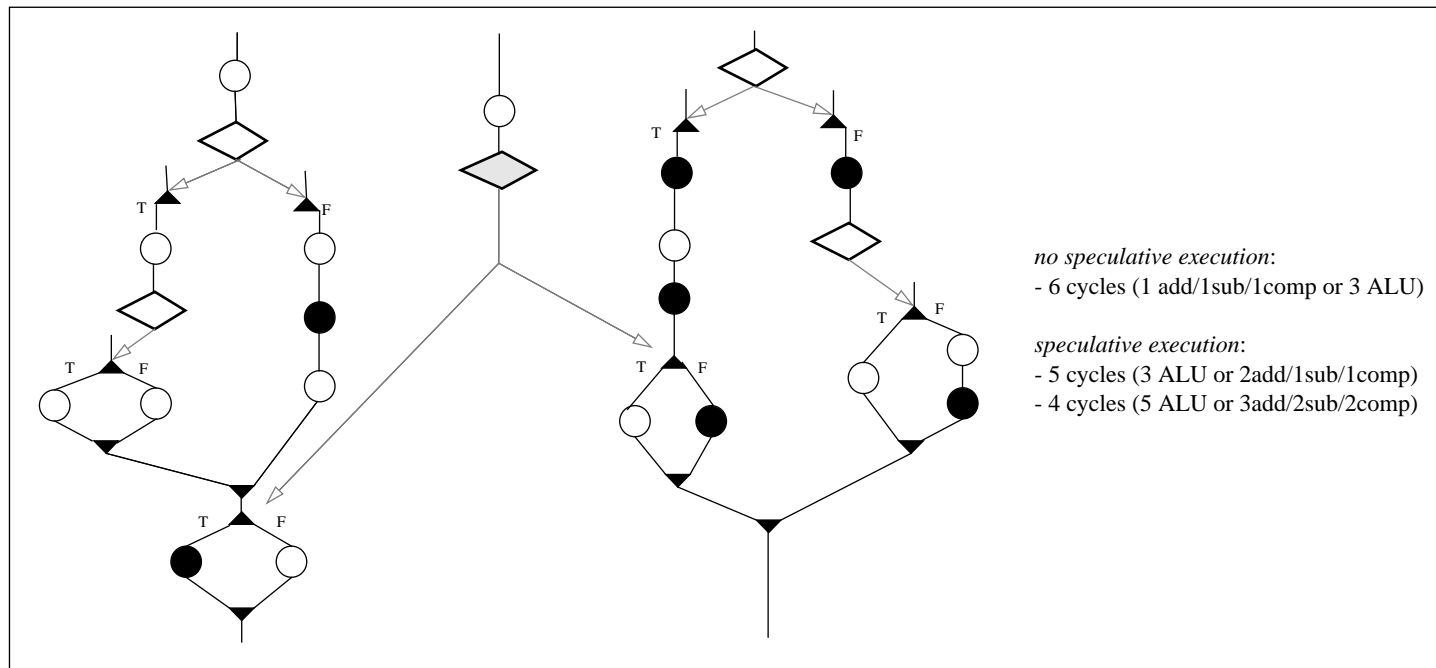


- Critical path length bounded by data dependencies
- Execution *order* of conditionals not dictated by input description

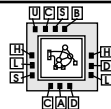


# PARALLEL CONTROL STRUCTURES

- **Globally** schedule operations belonging to parallel or correlated trees
- **False control path management essential**

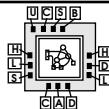


- **Hard problem**  $\Rightarrow$  attracted very little attention so far

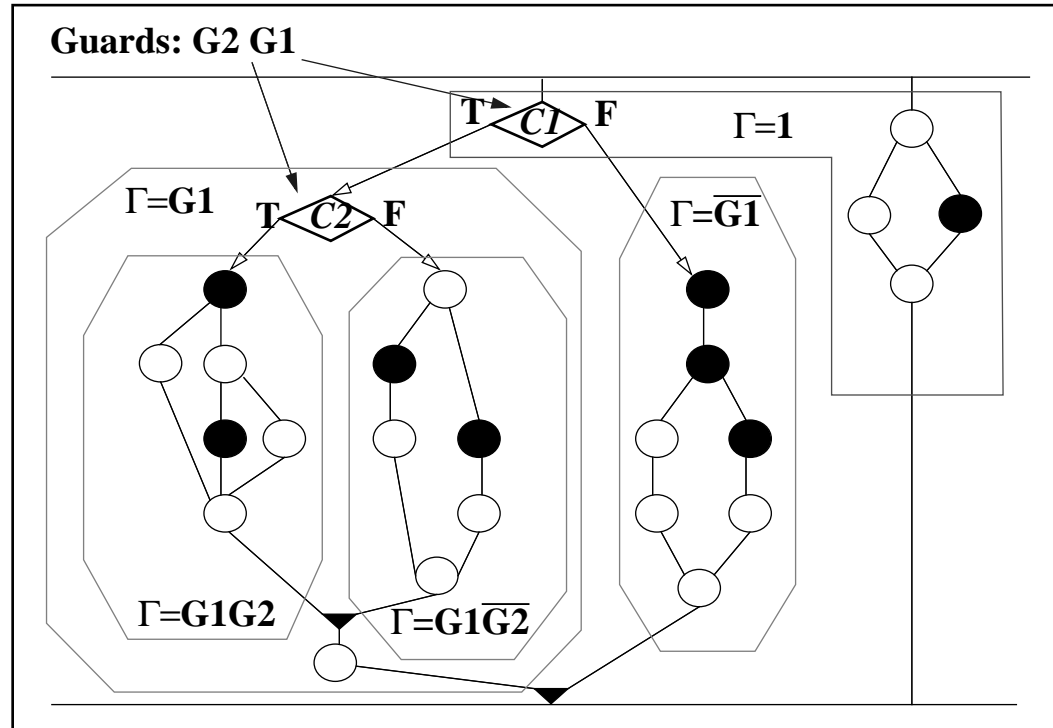


## 2.1. FORMULATION

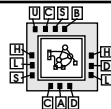
- **Scheduling constraints as Boolean functions**  
⇒ build *OBDD* corresponding to the intersection
- Each *variable*  $C_{ij}$  represents operation  $j$  occurring at time step  $i$
- Add a '*Guard*' variable for each conditional  
⇒ control path is a *product* of corresponding guards
- Define a '*Guard*' function  $\Gamma_i$  for each operation  $i$   
⇒ identifies control paths where operation  $i$  has to be scheduled
- *Solution* is a collection of traces  
⇒ *trace* is a valid execution instance for a particular control path
- Trace is a *product term* containing the information on a control path (guard variables) and its schedule (0/1 assignment of  $C_{ij}$  variables)



## EXAMPLE [Kim *et al.*, ICCAD'91]:

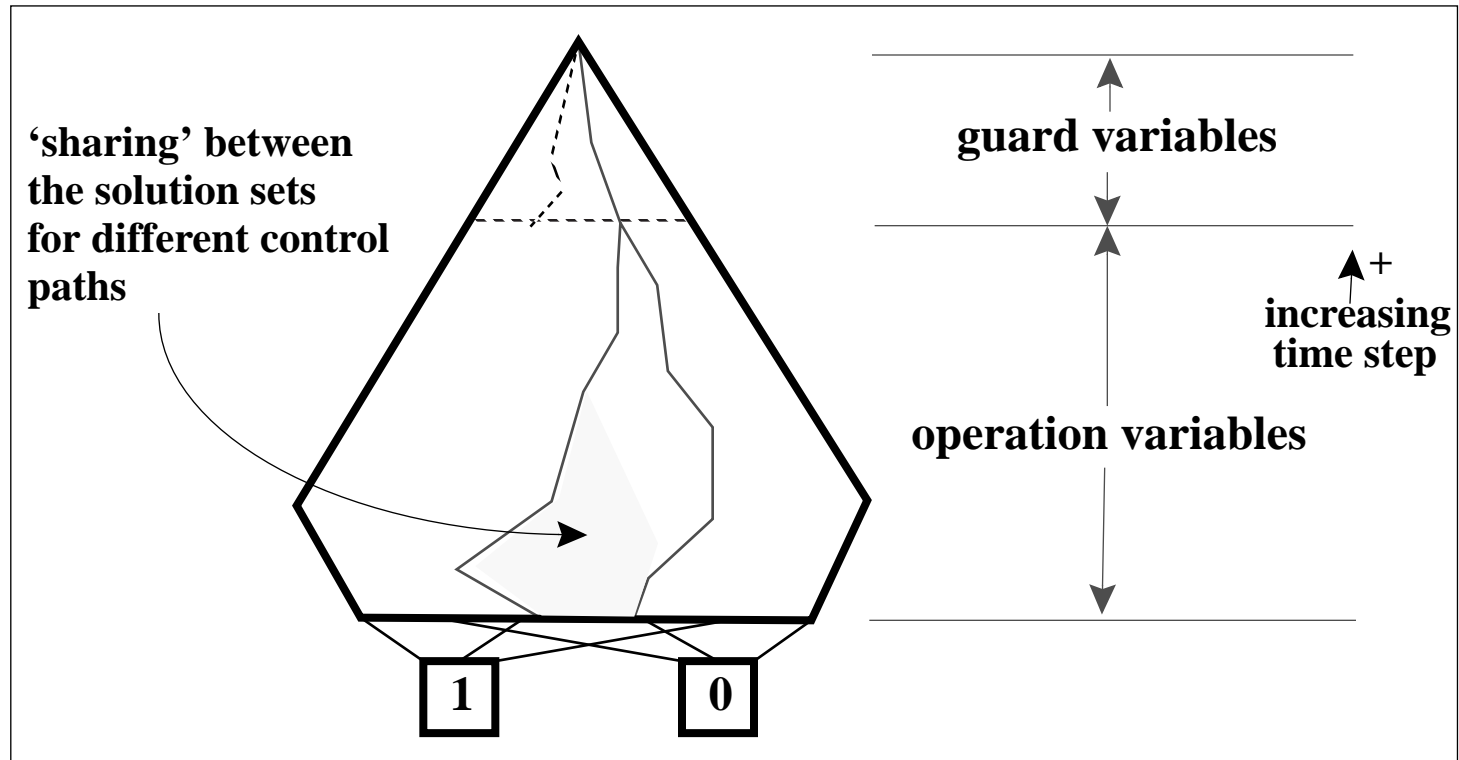


- **Blocks indicate guard functions  $\Gamma$**   
 $\Rightarrow \Gamma$ 's not restricted to product terms (can handle: *goto*, *exit*, *case*)
- **Operations with  $\Gamma=1$  scheduled simultaneously under all control combinations**

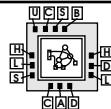


# OBDD REPRESENTATION

- **Compressed representation of very large number of solutions**

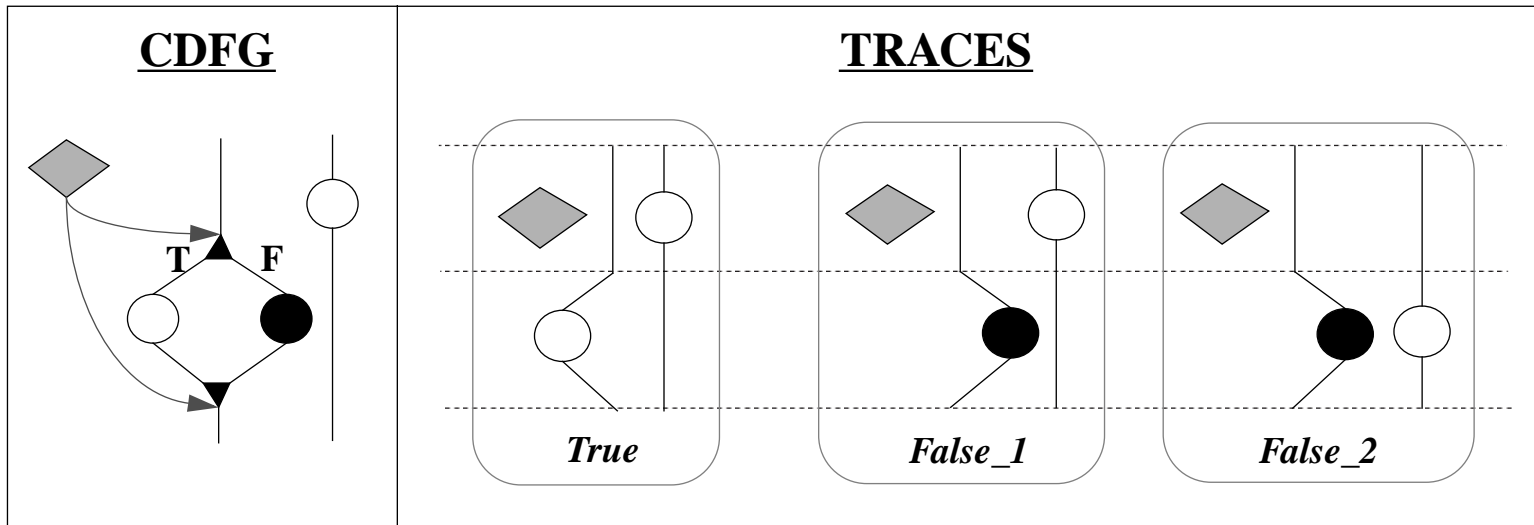


- **Relatively small number of OBDD nodes  $\Rightarrow$  fast manipulations**



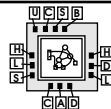
## WHAT IS INCLUDED IN THE SOLUTION?

- Three possible execution traces exist for the CDFG below (assume: 1 “white” resource available, no speculative execution)



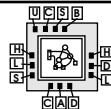
- “*True*” and “*False\_1*” form a deterministic executable (*ensemble*) schedule
- “*False\_2*” is an invalid trace -- cannot be paired with “*True*”

**Solution includes only traces belonging to at least one ensemble schedule**

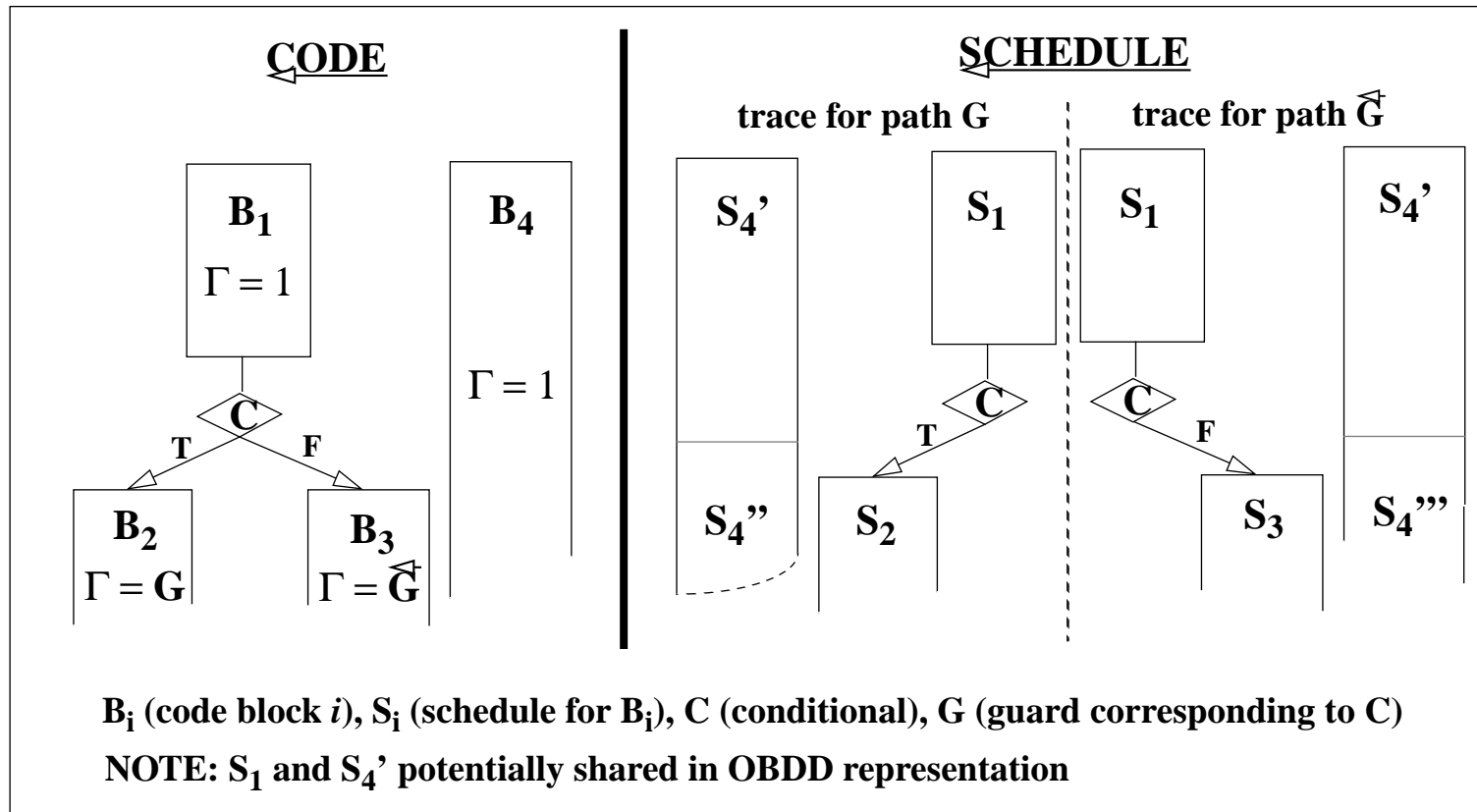


## TREATMENT OF RESOURCES

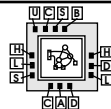
- If two operations  $i$  and  $j$  have guard functions  $\Gamma_i$  and  $\Gamma_j$  such that  $\Gamma_i \Gamma_j = 0$ ,  $i$  and  $j$  are pairwise mutually exclusive [SASIMI'93], however...
    - pairwise mutual exclusion not sufficient for optimal use of resources
    - higher-order analysis potentially expensive
  - We introduce Trace Validation as an alternative to mutex analysis
  - Trace Validation imposes causality and completeness on the set of all traces and ensures that each trace is a part of some ensemble schedule
    - *causality* -- traces must match before condition is evaluated
    - *completeness* -- valid execution traces must exist for all control paths
- ⇒ implicitly verifies that resource constraints satisfied on individual traces are not violated in the ensemble schedule



# TRACE VALIDATION



- Proposed algorithm *iteratively* validates all traces for all ensemble schedules *simultaneously*, using only Boolean operations on the OBDD data-structure
- Generates polynomial #constraints (regardless of #traces)



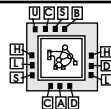
# TRACE VALIDATION

```

i = 0;
do {
  i++;
  S(i) = S(i-1);
  for each time step j {
    S' =  $\exists_{(V - V'(j))} S(i)$ 
    for each conditional  $c_k$  {
      S' =  $S'R_k(j) + \forall G_k(S'\overline{R_k(j)})$ 
      if (S'==0) { S(i)=0; exit; }
    }
    S(i) = S(i)S';
  }
} while (S(i)!=S(i-1));

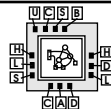
```

- $S$  - set of all traces,  $S(0)$  - initial set of non-validated traces,  $S(i)$  - set of traces at iteration  $i$
- $V$  - set of all variables not including guard variables,  $V'(j)$  - subset of  $V$  corresponding to time steps  $\leq j$
- $S'$  - set of traces from which all variables  $(V - V'(j))$  are removed:  $S' = \exists_{(V - V'(j))} S(i)$
- $C = [c_1, c_2 \dots c_n]$  - set of all conditionals
- $G = [G_1, G_2 \dots G_n]$  - set of guards corresponding to the conditionals
- $R(j) = [R_1(j), R_2(j) \dots R_n(j)]$  - *resolution vector*, set of Boolean functions indicating whether a conditional  $c_k$  was scheduled prior to time step  $j$ :  $R_k(j) = \sum C_{lk}$  for  $(l < j)$
- $G_{res}$  - subset of guards that are resolved prior to time step  $j$
- $\exists_x f = f_x + f_x^-$  - *existential abstraction*,  $\forall_x f = f_x f_x^-$  - *universal abstraction*



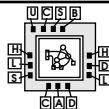
## 2.2. CONSTRUCTION

- Solution is constructed *iteratively* -- at every time step the following constraints are applied:
  1. Uniqueness
  2. Precedence relations
  3. Resource bounds
  4. Removal of redundantly scheduled operations
  5. Trace validation
  6. Completion test
- *All* of the solutions for *all* control paths are constructed *simultaneously*
- Trace validation ensures that each trace is part of some executable (causal and complete) schedule

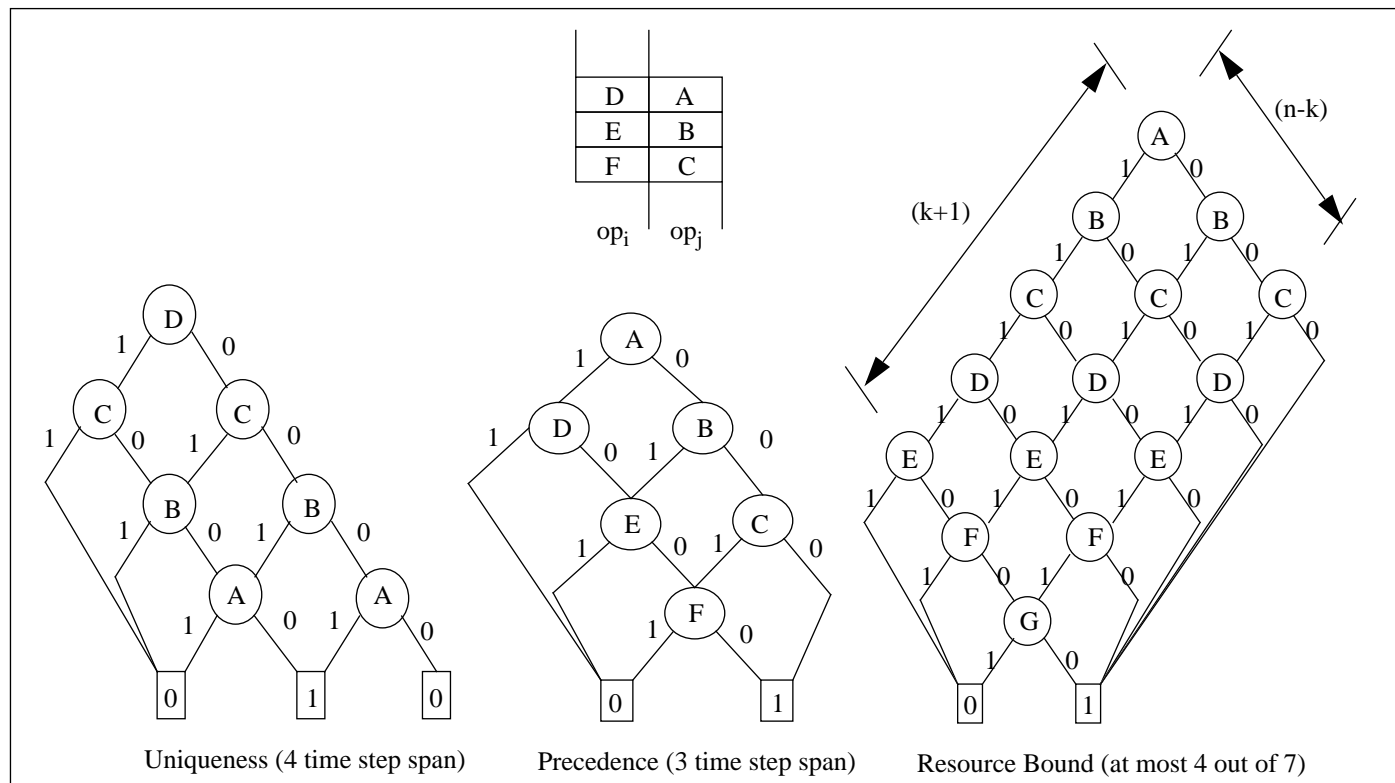


## WHY ITERATIVE CONSTRUCTION?

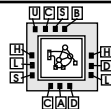
- **It is possible [SASIMP'93] to construct all constraints during preprocessing and form their intersection**
- **In this version *iterative* construction is implemented:**
  - **constraints incrementally generated during construction**
  - **easy to test for termination**
  - **no spurious partial solutions**
  - **smaller intermediate OBDDs**
- **Valid partial solutions simplify development of heuristics**



# OBDD FORM OF CONSTRAINTS

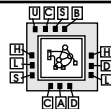


- **BDD representations have regular structure -- built *directly* from the CDFG (no need for intermediate Boolean equation form)**



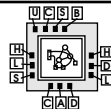
## 2.3. APPLICABILITY TO LARGE PROBLEMS

- **The main challenge for symbolic techniques -- large solution sets**
- *“... In many combinatorial optimization problems, symbolic methods using OBDDs have not performed as well as more traditional methods. In these problems we are typically interested in finding only one solution that satisfies some optimality criterion. Most approaches using OBDDs, on the other hand, derive all possible solutions and then select the best from among these. Unfortunately, many problems have too many solutions to encode symbolically...” (R. E. Bryant, 1992)*
- **We have shown [HLSS'94] that some standard benchmark instances have billions of optimal solutions**  
**⇒ In such cases, OBDD representation can become too large to be practical**



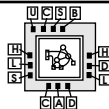
## APPLICABILITY TO LARGE PROBLEMS

- It has been generally considered that symbolic techniques are applicable only to small scheduling problems
- We demonstrate [ED&TC'95] that applicability of these techniques can be extended to large DFGs by:
  - using *Zero-Suppressed BDDs* [Minato]
  - applying the set of *interior constraints* to reduce size of intermediate solutions
  - *implicit* application of complex constraints
  - formulation of heuristics that preserve whole *sets* of partial solutions exhibiting desirable properties



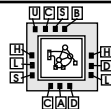
# SYMBOLIC ZBDD REPRESENTATION

- **Example: 28-cycle Elliptic Wave Filter (EWF) benchmark**
  - 34 operations but 437 representation variables
  - **OBDD**: Every solution includes all 437 variables -- only 34 of them “1”
    - ⇒ extremely large solution ( > 130,000 nodes)
  - **ZBDD**: “0”-variables are implicit (suppressed)
    - ⇒ *ten-fold* reduction in solution size
- **Drastically decreased memory requirements**
- **ZBDD form of constraints can be more complex, but the construction is still efficient -- solutions (both intermediate and final) are smaller**



# INTERIOR CONSTRAINTS

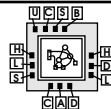
- Although the size of the final solution is typically very moderate, the *partial solutions* can become *prohibitively large*  
⇒ Ideally, intermediate size should never exceed the size of the final solution
- Identify and discard partial schedules that cannot contribute to a set of optimal solutions  
⇒ at a particular time step such partial schedules cannot terminate for given resources and pre-specified upper bound on execution time
- Set of *interior constraints* is dynamically generated to prune the OBDD/ZBDD
- Very cost-effective for large DFGs



# INTERIOR CONSTRAINTS

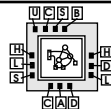
- Assume that at the beginning of step  $i$  there are:
  - $n$  addition operations that have ALAP bounds in the range  $[i... (i+k-1)]$
  - $m$  single-cycle adders available

⇒ At least  $(n-km)$  of these additions must be completed prior to step  $i$  in a feasible solution
- Similar constraints for pipelined and multicycle functional units
- $k^{\text{th}}$ -order constraints enable an early detection of many (not necessarily all) partial schedules destined to be discarded within the next  $k$  steps
- The completeness of the solution set preserved, hence no impact on optimality

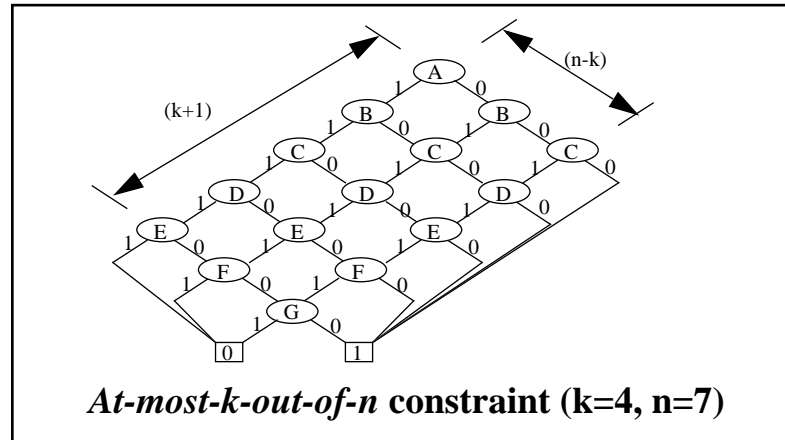


# IMPLICIT APPLICATION OF CONSTRAINTS

- It can happen that the partial solution is of a very moderate size, but the constraint to be applied *cannot be built*
- Similar problem arises in many symbolic applications  
(e.g. in FSM traversal, where transition functions may become prohibitively complex and Characteristic Functions [Coudert *et al.*] are introduced to alleviate the problem)
- Apply constraints *implicitly* without attempting to build constraint BDD
- *Example*: application of *register constraints* that can be build explicitly for small problems only

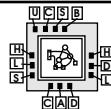


## Example: Generalized resource bound applied to registers



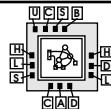
- Vertices in above template can be arbitrary Boolean functions ( $f_1, \dots, f_n$ )  
 $\Rightarrow$  typically, constraint *too large* to be built
  - introduce a set of auxiliary variables ( $y_1, \dots, y_n$ ) corresponding to ( $f_1, \dots, f_n$ )
  - build above template  $T$  using ( $y_1, \dots, y_n$ )
  - compute  $P^0 = \text{And}(P', T)$ , where  $P'$  is a partial solution
  - compute new partial solution  $P'' = P^n$  using recursive substitution:

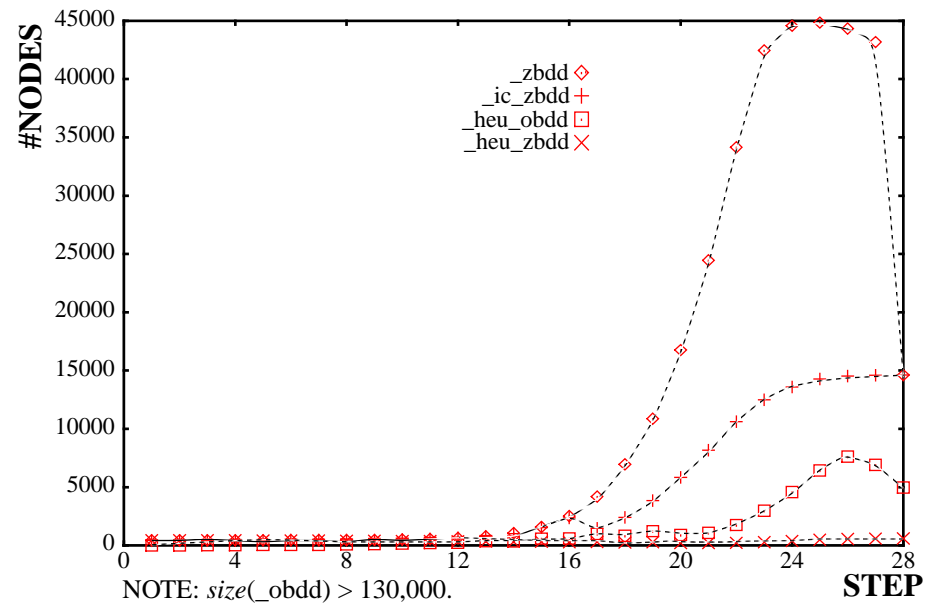
$$P^i = P^{(i-1)} \mid y_i \equiv f_i$$



# SET-HEURISTICS

- **Since valid partial schedules are available after each step, it is possible to devise efficient heuristics**
- ***Utility-based* heuristic propagates only the subset of schedules with *maximum utilization* of resources**
  - **Since *all* such schedules are propagated, this *simple* heuristic has good behavior**
- ***Set-Heuristics* are very *robust***
  - **Construction pace shows little sensitivity to the upper bound on scheduling latency (initialization parameters)**





## 28-cycle EWF: exact and heuristic constructions

- *resources*: 1 single-cycle adder, 1 two-cycle multiplier ( $> 10e+9$  solutions)  
 - *#variables*: 437

*\_zbdd*: exact solution (ZBDD construction), no interior constraints: ~ 20.5 min

*\_ic\_zbdd*: exact solution (ZBDD construction) built using interior constraints: ~ 12.5 min <sup>[1]</sup>

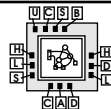
*\_heu\_obdd*: utility-based set-heuristic solution (OBDD construction): ~ 38 s <sup>[2]</sup>

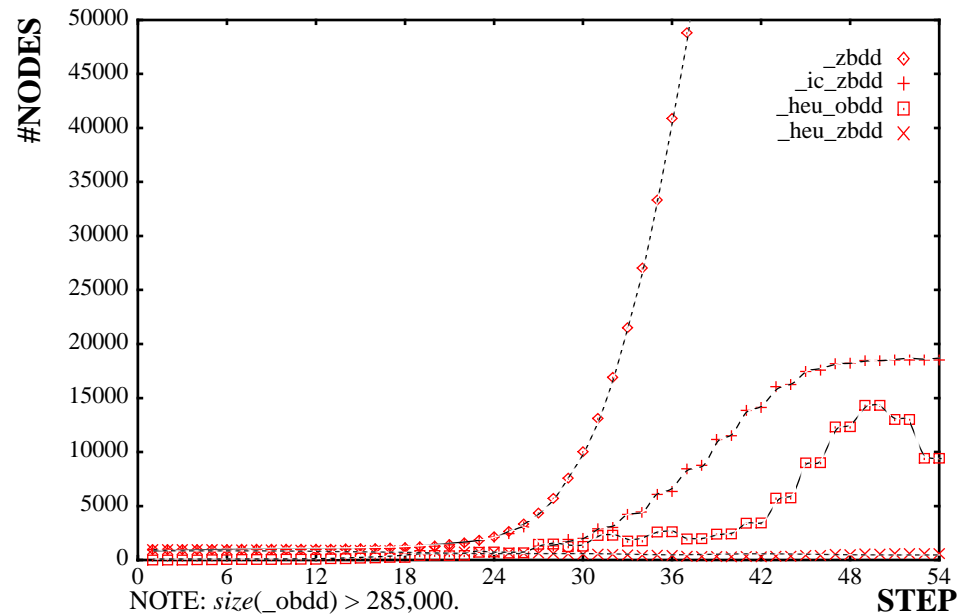
*\_heu\_zbdd*: utility-based set-heuristic solution (ZBDD construction): ~ 110 s

*\_obdd*: exact solution (OBDD construction): > 1.5 h

1. For optimal number of registers (10), the size of exact ZBDD solution decreases from ~14.5 to ~3.5 Knodes.

2. ~ 9 s if both utilization and critical path are used as heuristic criteria





## 54-cycle EWF: exact and heuristic constructions

- *resources*: 1 two-cycle adder, 1 two-cycle multiplier ( $> 10e+13$  solutions)

- *#variables*: 967

\_zbdd: exact solution (ZBDD construction), no interior constraints: could not be constructed

\_ic\_zbdd: exact solution (ZBDD construction) built using interior constraints:  $> 1.5$  h <sup>[1]</sup>

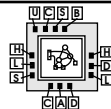
\_heu\_obdd: utility-based set-heuristic solution (OBDD construction):  $\sim 2$  min <sup>[2]</sup>

\_heu\_zbdd: utility-based set-heuristic solution (ZBDD construction):  $\sim 12.5$  min

\_obdd: exact solution (OBDD construction) -- not constructed (converted from ZBDD solution)

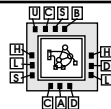
1. For optimal number of registers (10), the size of exact ZBDD solution decreases from  $\sim 18.5$  to  $\sim 6.5$  Knodes.

2.  $\sim 30$  s if both utilization and critical path are used as heuristic criteria.



## 2.4. EXPERIMENTAL RESULTS

- **Experiments on standard set of benchmarks:**
  - *EWF, EWF-2, EWF-3* (*EWF* unfolded 2/3 times), *FDCT, Hal, Maha, Kim, Wakabayashi* [DAC'89], *MulT* [Wakabayashi *et al.*, DAC'92]
- **CDFGs with up to 102 operations**
- **Execution times up to 105 time steps**
- **Exact scheduler: ~ 1,000 formulation variables**
- **Heuristic scheduler: ~ 6,000 formulation variables**
- **Very moderate CPU times and memory requirements**



# EXPERIMENTAL RESULTS

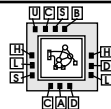
- *EWF (Elliptic Wave Filter, 34 operations):*

## EWF experiments (exact solutions)

#cycles	17	17	18	18	19	20	20	21	28	28
#adders	3	3	3	2	2	2	2	2	1	1
#multipliers	2 (*)	3	1 (*)	2	1 (*)	2	1 (*)	1	1 (*)	1
#busses	6	6	6	6	6	4	4	4	4	4
#registers	10	10	10	10	10	10	10	10	10	10
#variables	63	63	97	97	131	165	165	199	437	437
#nodes	82	82	194	209	2,237	2,760	1,905	704	48,649	32,487
#schedules	18	18	336	18	10,692	52,821	5,142	2,355	4.29e+9	2.63e+8
CPU time [s]	0.26	0.32	1.14	1.43	7.92	35.12	26.00	7.89	2,420.2	1520.51
									5	

2-cycle multiplier and single-cycle adder except: (\*) 2-cycle pipelined multiplier.

- *All optimal solutions to all benchmark instances*
- **Number of OBDD nodes significantly smaller than  $(\#variables)^2$**



# EXPERIMENTAL RESULTS

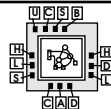
## Benchmarks with branching (exact solutions)

		<i>Maha</i>	<i>Parker</i>	<i>Kim</i>	<i>Waka</i>	<i>MulT</i>	
#cycles(spec)	longest	5	4	4	6	7	3
	average	3.31	2.25	2.13	5.75	5.0	3.0
#cycles(non_spec)		8	8	8	7	7	4
#adders		1	2	2	2	1	2
#subtracters		1	3	3	1	1	1
#comparators		-	-	-	1	2	1
#variables		65	49	49	71	55	26
#nodes		428	325	220	543	271	116
#traces		15	43	12	124	21	15
CPU time [s]		13.80	6.32	8.91	7.63	2.84	3.95
single-cycle adders, subtracters and comparators assumed							

## Comparison with others: average (longest) path

	<i>Maha</i>	<i>Parker</i>	<i>Kim</i>	<i>Waka</i>	<i>MulT</i>	
our	3.31 (5)	2.25 (4)	2.13 (4)	5.75 (6)	5 (7)	3 (3)
TS [12]	3.31 (5)	-	-	-	4.75 (7)	-
CVLS [33]	3.31 (5)	2.38 (4)	2.38 (4)	5.75 (6)	-	2.88 (4)
Kim <i>et al.</i> [17]	4.62 (8)	-	-	6.25 (7)	4.75 (7)	-

- **Comparable or superior results using very moderate CPU resources**



# EXPERIMENTAL RESULTS

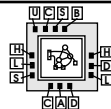
- **54-cycle *EWF*:**

## Robustness analysis of the heuristic scheduler (*EWF*)

cycles	upper bound	# vars	<i>utility-based</i>		<i>utility + CP</i>	
			max # nodes	CP U [s]	max # nodes	CP U [s]
54	54	967	14,328	119	2,210	30
	55	1,001	14,839	124	2,292	31
	56	1,035	15,559	138	2,392	34
	59	1,137	17,151	159	2,616	36
	64	1,307	19,378	202	2,914	40

2-cycle adder, 2-cycle multiplier

- ***Set-Heuristics* are very *robust*: the construction pace shows little sensitivity to the upper bound on scheduling latency (initialization parameters)**
  - Can be used to derive accurate upper bounds for exact schedulers whose runtime efficiency is more sensitive to such estimates



## EXPERIMENTAL RESULTS

- *FDCT (Finite Discrete Cosine Transform, 42 operations):*

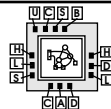
### FDCT experiments (heuristic vs. exact)

ad d	su b	mul	bus	cycles [ZS]	cycle s [our]	optima l	reg [h/e]	#vars	max #node s	CPU [s]	CPU rel
2	2	2	10	10	<b>10</b>	yes	11/ 10	251	1,490	79.1	0.30 0
2	2	2 <sup>(*)</sup>	10	-	<b>11</b>	yes	9/9	229	2,252	65.4	0.15 3
1	1	2	8	14	<b>13</b>	yes	12/ 11	377	3,988	52.8	0.01 1
1	1	2 <sup>(*)</sup>	8	-	<b>14</b>	?	11/-	355	4,451	42.7	-
1	1	1	6	20	<b>18</b>	yes	11/ 10	587	12,340	179. 3	0.08 4

- **Heuristic OBDD scheduler**

(*max #nodes*) - max OBDD size during construction, (*CPU [s]*) - CPU time, (*reg [h/e]*)  
- heuristic/exact #registers, (*CPU rel*) - heuristic/exact(ZBDD) CPU time ratio

- **Outperforms *Zone Scheduling* (ILP-based heuristic) which partitions large problems and solves the subproblems optimally**

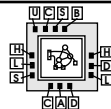


# EXPERIMENTAL RESULTS

- *EWF* unfolded 2 times (68 operations):

## EWF-2 experiments (heuristic vs. exact)

ad d	mul	bus	cycle s	optima l	reg [h/e]	#vars	max #node s	CPU [s]	CPU rel
3	3	6	<b>33</b>	yes	11/ 11	135	178	0.7	0.15 6
3	2 <sup>(*)</sup>	6	<b>33</b>	yes	11/ 11	203	178	0.7	0.15 6
3	1 <sup>(*)</sup>	6	<b>34</b>	yes	11/ 11	203	203	1.5	0.11 5
3	2	6	<b>35</b>	no (34)	11/ 11	271	291	3.2	0.14 8
2	2 <sup>(*)</sup>	6	<b>35</b>	yes	11/ 11	271	661	4.3	0.05 9
2	2	6	<b>35</b>	yes	11/ 11	271	639	4.2	0.06 1
		4	<b>39</b>	yes	12/ 11	543	1,770	23.8	0.00 8
2	1 <sup>(*)</sup>	6	<b>36</b>	yes	11/ 11	339	686	4.2	0.02 4



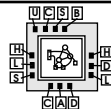
# EXPERIMENTAL RESULTS

- *EWF* unfolded 3 times (102 operations):

## EWF-3 experiments (heuristic vs. exact)

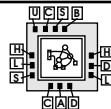
add	mul	bus	<b>cycles</b>	optimal	reg [h/e]	#vars	max #nodes	CPU [s]	CPU rel
3	3	6	<b>49</b>	yes	12/12	207	293	1.1	0.104
3	2 <sup>(*)</sup>	6	<b>49</b>	yes	12/12	207	293	1.1	0.104
3	1 <sup>(*)</sup>	6	<b>50</b>	yes	12/12	309	309	2.4	0.066
3	2	6	<b>52</b>	no (50)	12/12	513	549	8.8	0.140
2	2 <sup>(*)</sup>	6	<b>52</b>	yes	12/12	513	1,263	12.1	0.022
		4	<b>58</b>	?	13/-	1,125	3,450	71.9	-
2	2	6	<b>52</b>	yes	12/12	513	1,289	11.9	0.024
		4	<b>58</b>	?	13/-	1,125	3,450	70.9	-
2	1 <sup>(*)</sup>	6	<b>53</b>	yes	12/12	615	1,176	9.5	0.009
		4	<b>58</b>	?	12/-	1,125	4,065	74.4	-
2	1	4	<b>59</b>	?	12/-	1,227	2,249	56.9	-
1	1 <sup>(*)</sup>	4	<b>84</b>	?	15/-	3,777	5,408	186.2	-
		2	<b>102</b>	?	15/-	5,613	7,762	547.2	-
1	1	4	<b>84</b>	?	15/-	3,777	5,408	176.9	-
		2	<b>105</b>	?	15/-	5,919	8,010	591.2	-

1-cycle adder, 2-cycle multiplier except: (\*) 2-cycle pipelined multiplier



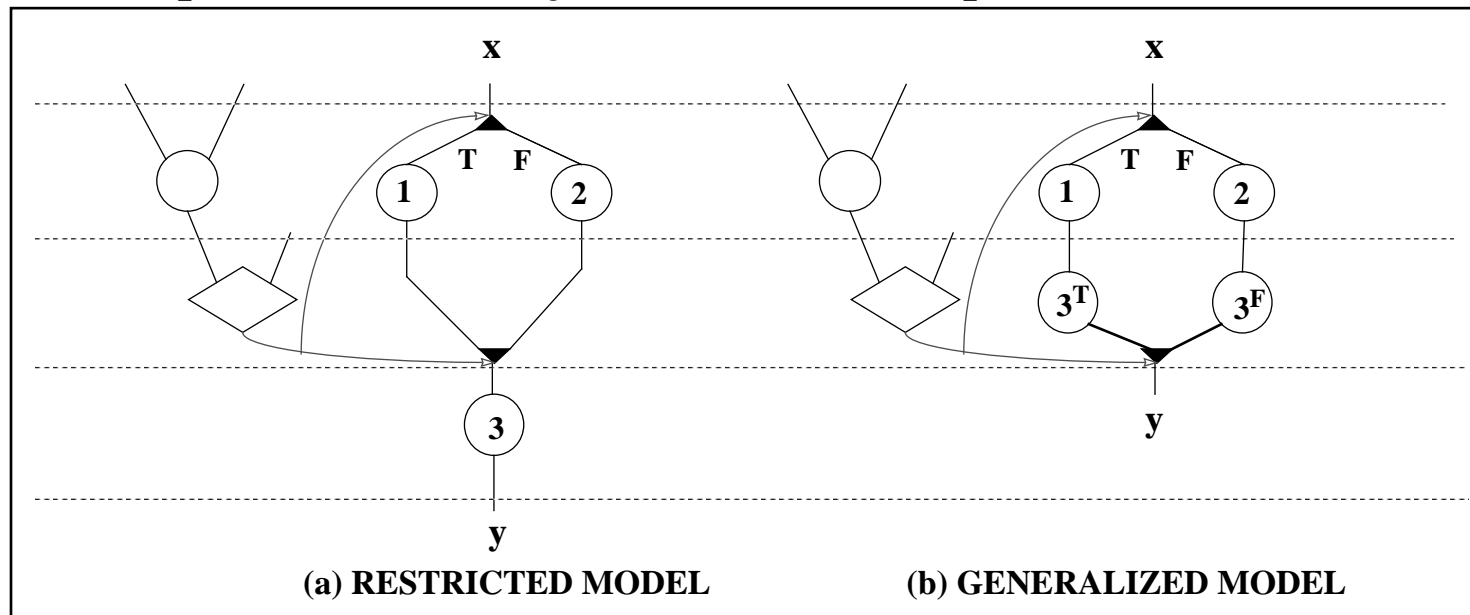
# EXPERIMENTAL RESULTS

- **Very encouraging results for large DFGs:**
  - **heuristics are robust and efficient (CPU time, memory usage) while still generating excellent results**
- **Topics to be investigated:**
  - **incorporate register cost (not just a bound on the number) in heuristics**
  - **develop efficient heuristics for CDFGs**
  - **speed-up ZBDD manipulations**

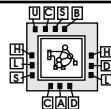


## 2.5. MODEL GENERALIZATION

- **Restriction in current model for speculative execution:**  
-- operation after the join node cannot be pre-executed

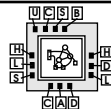
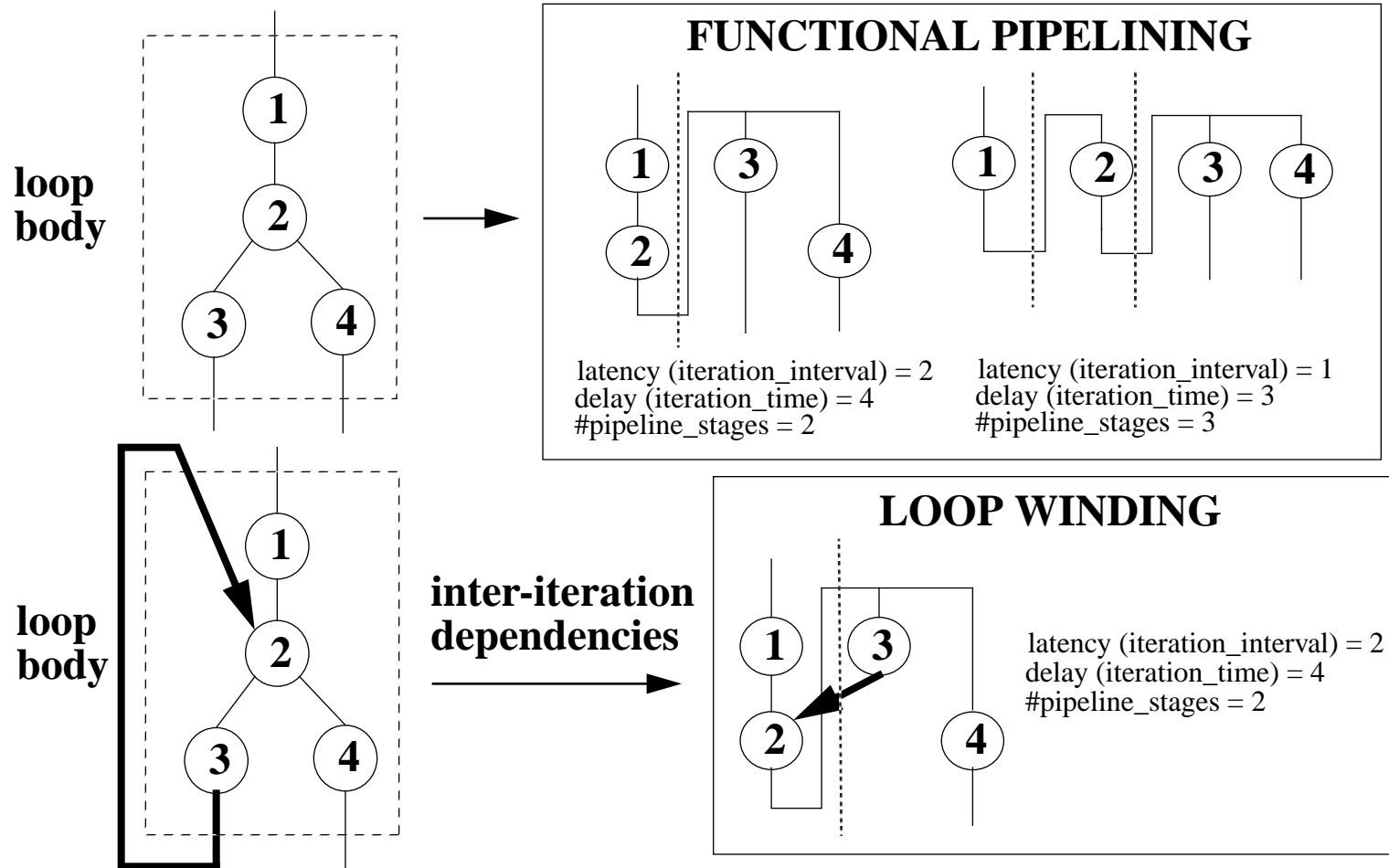


- **Generalized model can lead to an explosion of operation instances due to a node duplication [Percolation Based Synthesis]**  
-- is there an *efficient* implementation?



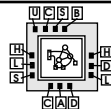
## 2.6. LOOP OPTIMIZATIONS

- Goal: increase *throughput* by decreasing loop latency



## LOOP OPTIMIZATIONS (cont.)

- **Exact formulation**
  - (i) **Modification of resource constraints:** if latency is  $l$ , then operations at time steps  $i, i+l, i+2l...$  ( $1 \leq i \leq l$ ) share resources
  - (ii) **Enforce additional inter-iteration dependencies (loop winding)**
- **Heuristics**
  - greedy set-based heuristics not directly applicable: tend to fill early time slots during iterative construction process
- **Formulation extensions to allow control-dependent behavior within the loop body?**
  - due to the problem complexity, very little research reported so far...



## PUBLICATIONS:

- [SASIMI'93] I. Radivojević, F. Brewer, “Symbolic Techniques for Optimal Scheduling”, *Proc. 4th SASIMI Workshop*, October 1993.
- [HLSS'94] I. Radivojević, F. Brewer, “Ensemble Representation and Techniques for Exact Control-Dependent Scheduling”, *Proc. 7th High Level Synthesis Symposium*, May 1994.
- [DAC'94] I. Radivojević, F. Brewer, “Incorporating Speculative Execution in Exact Control-Dependent Scheduling”, *Proc. 31st ACM/IEEE Design Automation Conference*, June 1994.
- [ED&TC'95] I. Radivojević, F. Brewer, “On Applicability of Symbolic Techniques to Larger Scheduling Problems”, to appear in the *Proc. European Design and Test Conference*, March 1995.
- [IEICE'95] I. Radivojević, F. Brewer, “Symbolic Scheduling Techniques”, to appear in *the IEICE Trans. Information and Systems*, Japan, March 1995.

