Stage Scheduling:

A Technique to Reduce the Register Requirements of a Modulo Schedule

Alexandre E. Eichenberger and Edward S. Davidson

Advanced Computer Architecture Laboratory
University of Michigan
alexe, davidson@eecs.umich.edu

MICRO-28
Friday, December 1

Motivation

Modulo Scheduling:

- exploits instruction level parallelism efficiently
- extends to a wide range of loops

Register requirements:

- high due to high level of parallelism
- increase for wider issue machines and higher latency pipelines

Objectives:

- highest steady-state performance
- highest transient performance
- lowest register requirements

Motivation (cont.)

Optimal modulo schedulers:

- optimal solutions minimizing any combinations of previous objectives
- computationally intensive (esp. with minimum register requirements)

Heuristics:

- fulfilling distinct constraints simultaneously:
  - resource constraints
  - schedule length along critical dependence cycles
  - schedule length of iterations
  - register requirements
- for realistic machines:
  - finite resources (FUs, registers)
  - potentially complex resource usage patterns (FUs, ports, busses)

Stage Scheduling Approach

Objective:

- low register requirements without impact on the steady-state performance
- fast scheduling algorithms (e.g. linear-time)

Approach:

- modifies an existing modulo schedule to reduce its register requirements

Key concepts of stage scheduling:

- preserves the original *Initiation Interval* (***II***) and *Modulo Reservation Table* (***MRT***):
  - same throughput
  - same patterns of resource usages
- shifts operations by multiples of ***II*** cycles:
  - decreases the register requirements
  - satisfies data dependences

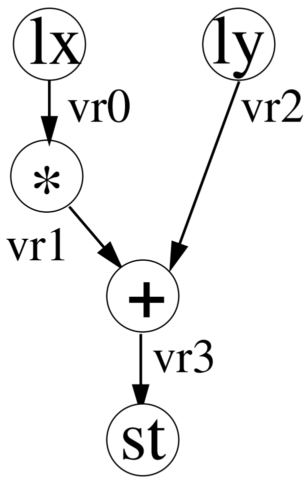
Outline

- Motivation
- Introductory Example
- Formulation of Stage Scheduling
- Heuristics for Stage Scheduling
- Benchmark Examples
- Conclusions

Early Stage Scheduling

Dependence graph:

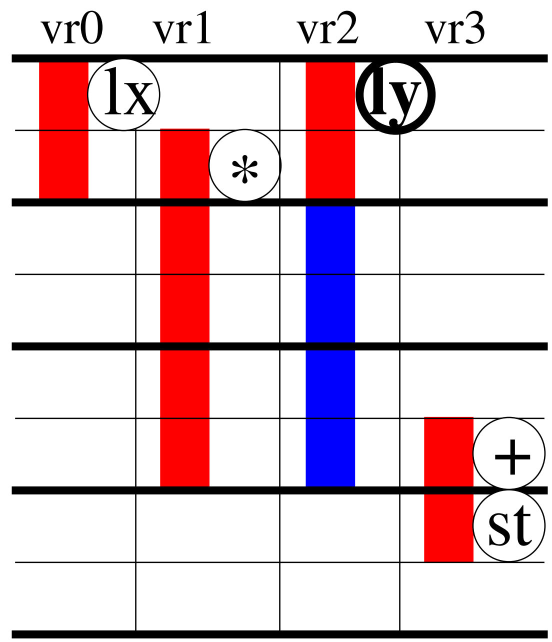
z[i] = 2*x[i] + y[i]



Stage schedule:

Time			
0	lx	ly	st
1	*	+	
2	lx	ly	st
3	*	+	
4	lx	ly	st
5	*	+	
6	lx	ly	st
7	*	+	

Lifetimes:



Minimum Lifetime (Latency + 1)

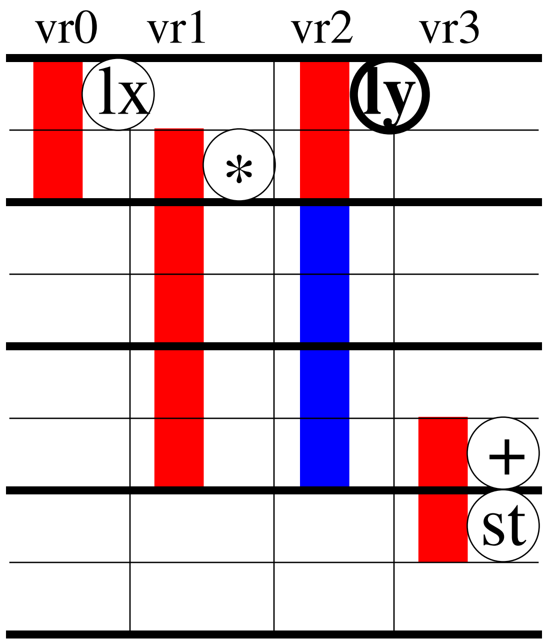
Additional lifetime

MRT (II = 2):

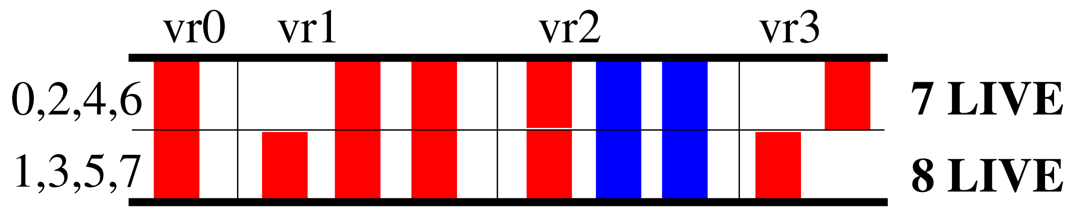
lx	ly	st
*	+	

Early Stage Scheduling (cont.)

Lifetimes:



Register requirements:



MaxLive:

= Maximum live virtual registers over all rows

= max(7, 8)

= 8

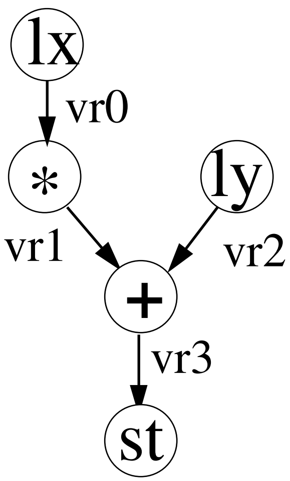
Minimum Lifetime

Additional lifetime

- number of LIVE virtual registers are counted accurately in each row
- there is an assignment with MaxLive physical registers

Late Stage Scheduling

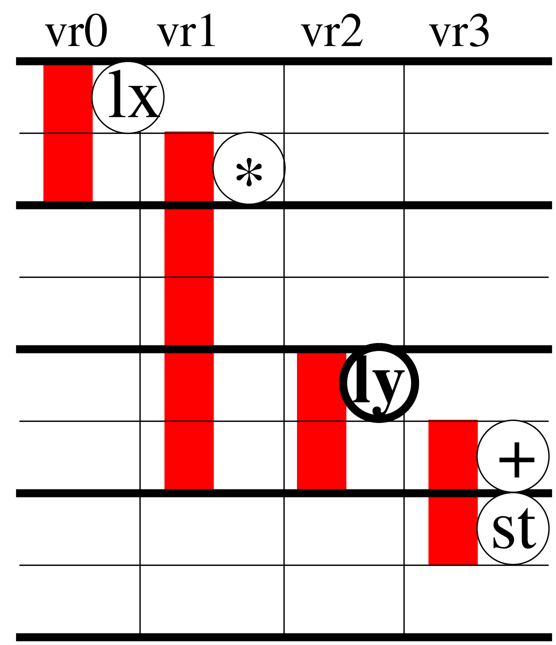
Dependence graph:



Stage schedule:

Time			
0	**lx**	ly	st
1	*	+	
2	lx	ly	st
3	*	+	
4	lx	**ly**	st
5	*	**+**	
6	lx	ly	**st**
7	*	+	

Lifetimes:



MRT (II = 2):

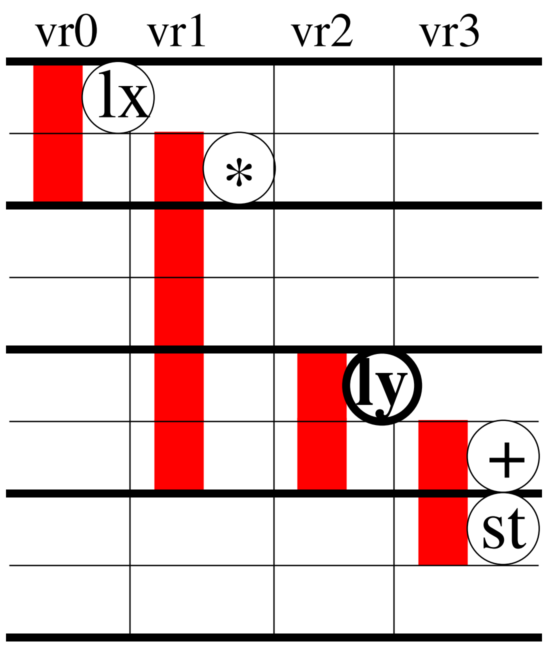
lx	ly	st
*	+	

Minimum Lifetime (Latency + 1)

Additional lifetime

Late Stage Scheduling (cont.)

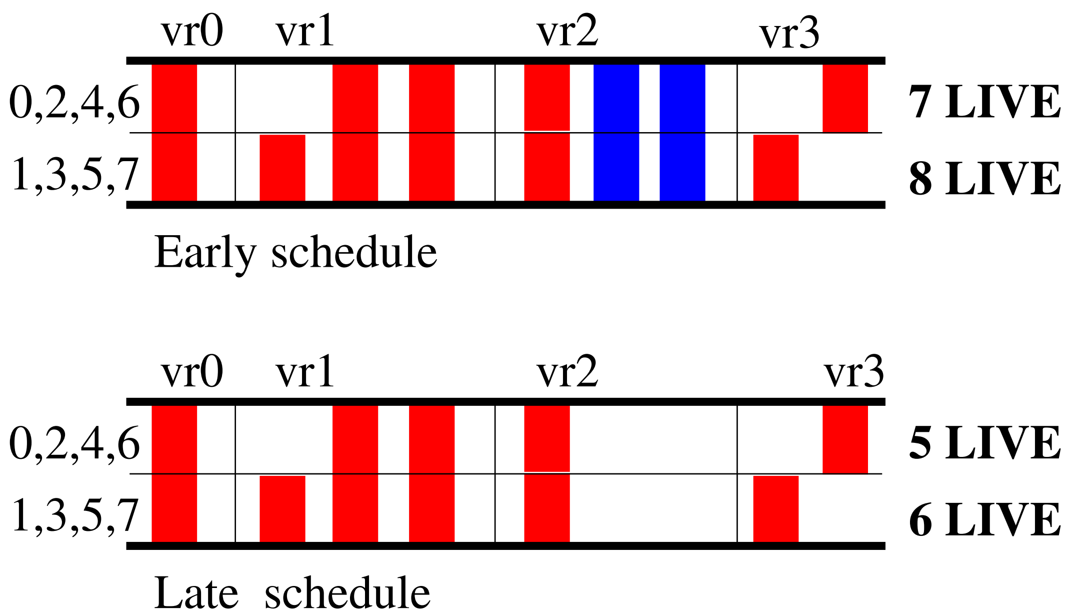
Lifetimes:



Minimum Lifetime

Additional lifetime

Register requirements:



MaxLive is 6 vs. 8 earlier

MaxLive reduced by scheduling ly
2*II (= 4 cycles) later

Formulation of Stage Scheduling

Input:

- dependence graph
  - latency between dependent operations ***i*** and ***j*** : $latency_{i,j}$
  - dependence distance between dependent operations ***i*** and ***j*** : $\omega_{i,j}$
- modulo schedule
  - initiation interval: II
  - schedule time of operation ***i***: $time_i$

Internal representation of a stage schedule:

- time difference along edges in the dependence graph, i.e.

  edge (i, j): $time_j - time_i = rdist_{i,j} + skip_{i,j} * II$

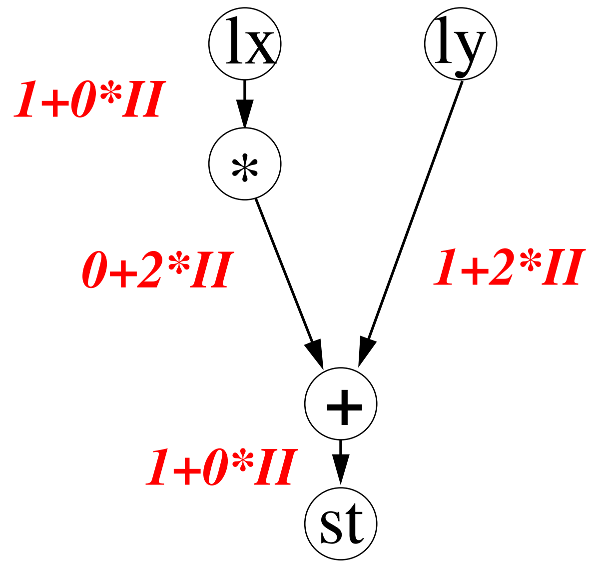
  - fractional distance: $rdist_{i,j}$ $(0 \leq rdist_{i,j} < II)$
  - integral distance: $skip_{i,j}$

Formulation of Stage Scheduling (cont.)

Early stage schedule (*II*=2):

Time			
Time 0	lx	ly	st
1	*	+	
2	lx	ly	st
3	*	+	
4	lx	ly	st
5	*	+	
6	lx	ly	st
7	*	+	

Schedule / dependence graph:



$$time_j - time_i = rdist_{i,j} + skip_{i,j} * II$$

Stage scheduling modifies the skip factors such that:

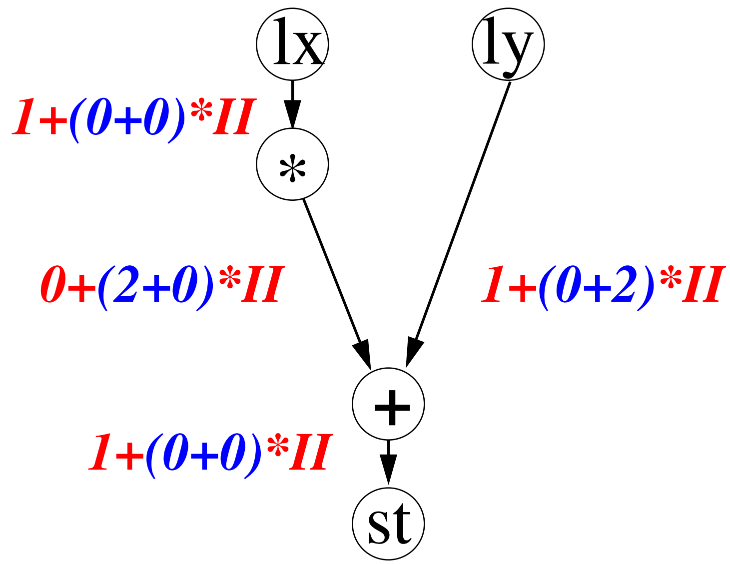
- the dependence constraints are fulfilled
- the skip factors are consistent
- the register requirements are reduced

Formulation of Stage Scheduling (cont.)

Dependence constraints:

- define $skip_{i,j} = s_{i,j} + p_{i,j}$
  - minimal skip factor to satisfy the dependence constraints (integer): $s_{i,j}$
  - additional skip factor (nonnegative integer): $p_{i,j}$
- dependence constraints are fulfilled $\Leftrightarrow$ $p_{i,j} \geq 0$

Time 0	lx	ly	st
1	*	+	
2	lx	ly	st
3	*	+	
4	lx	ly	st
5	*	+	
6	lx	ly	st
7	*	+	



$$time_j - time_i = rdist_{i,j} + (s_{i,j} + p_{i,j}) * II$$

Formulation of Stage Scheduling (cont.)

Consistent skip factors:

- must enforce the underlying cycle constraints [Eichenberger *et al*, 94], i.e.

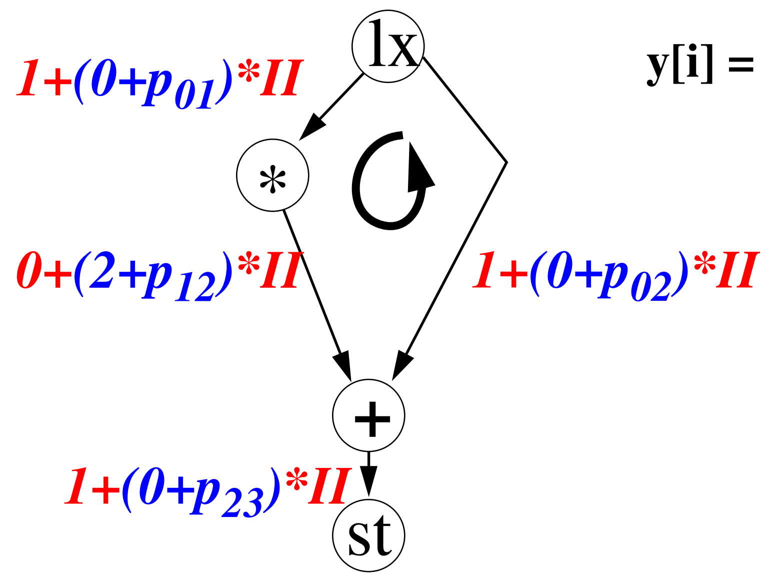
  cumulative sum of the $p_{i,j}$ along an underlying cycle remains constant

Time			
0	**lx**		st
1	*****	+	
2	lx		st
3	*	+	
4	lx		st
5	*	**+**	
6	lx		**st**
7	*	+	

$p_{01} = 0$

$p_{12} = 0$

$p_{02} = 2$



$y[i] = x[i]^2 + x[i]$

Cumulative time in underlying cycle:

$$[1+(0+p_{01})*II] + [0+(2+p_{12})*II] - [1+(0+p_{02})*II] = 0$$

$$p_{01} + p_{12} - p_{02} = -2$$

Formulation of Stage Scheduling (cont.)

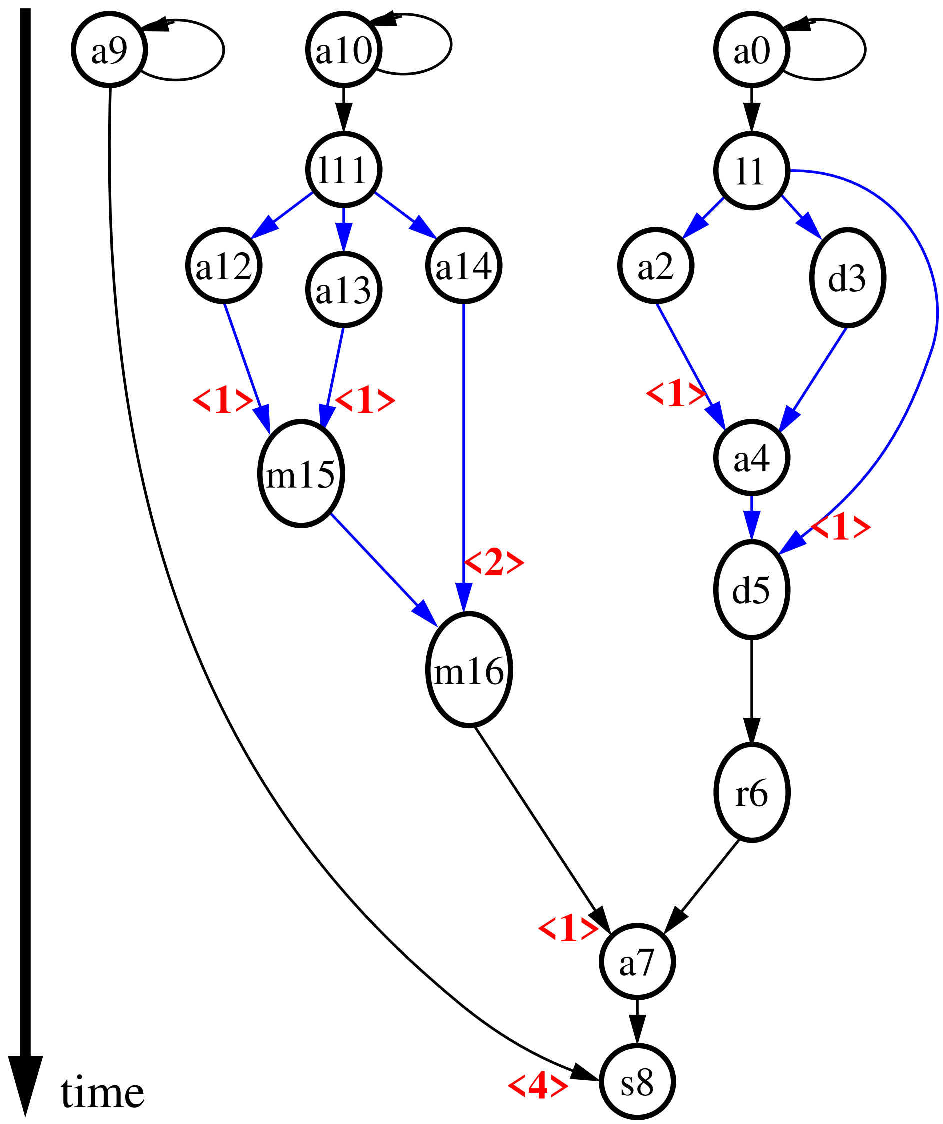
Problem formulation:

- given the input p-factor on each edge
- reduce the p-factor (i.e. reduce the register requirements) such that:
  - dependence constraint fulfilled $\Leftrightarrow$ $p_{i,j} \geq 0$
  - underlying cycle constraint fulfilled $\Leftrightarrow$ cumulative sum of the $p_{i,j}$ along an underlying cycle remains constant
- initial values:

$$s_{i,j} = \left\lceil \frac{latency_{i,j} - rdist_{i,j}}{II} \right\rceil - \omega_{i,j} \qquad p_{i,j} = \left\lfloor \frac{time_j - time_i - latency_{i,j}}{II} \right\rfloor + \omega_{i,j}$$

Running Example

Schedule / dependence graph:



mark each edge with its corresponding $p_{i,j}$ value

invariant1: $p_{i,j} \geq 0$

invariant2: sum of the $p_{i,j}$ along an underlying cycle remains constant

<x> non-zero $p_{i,j}$ variables

s7 short latency ops

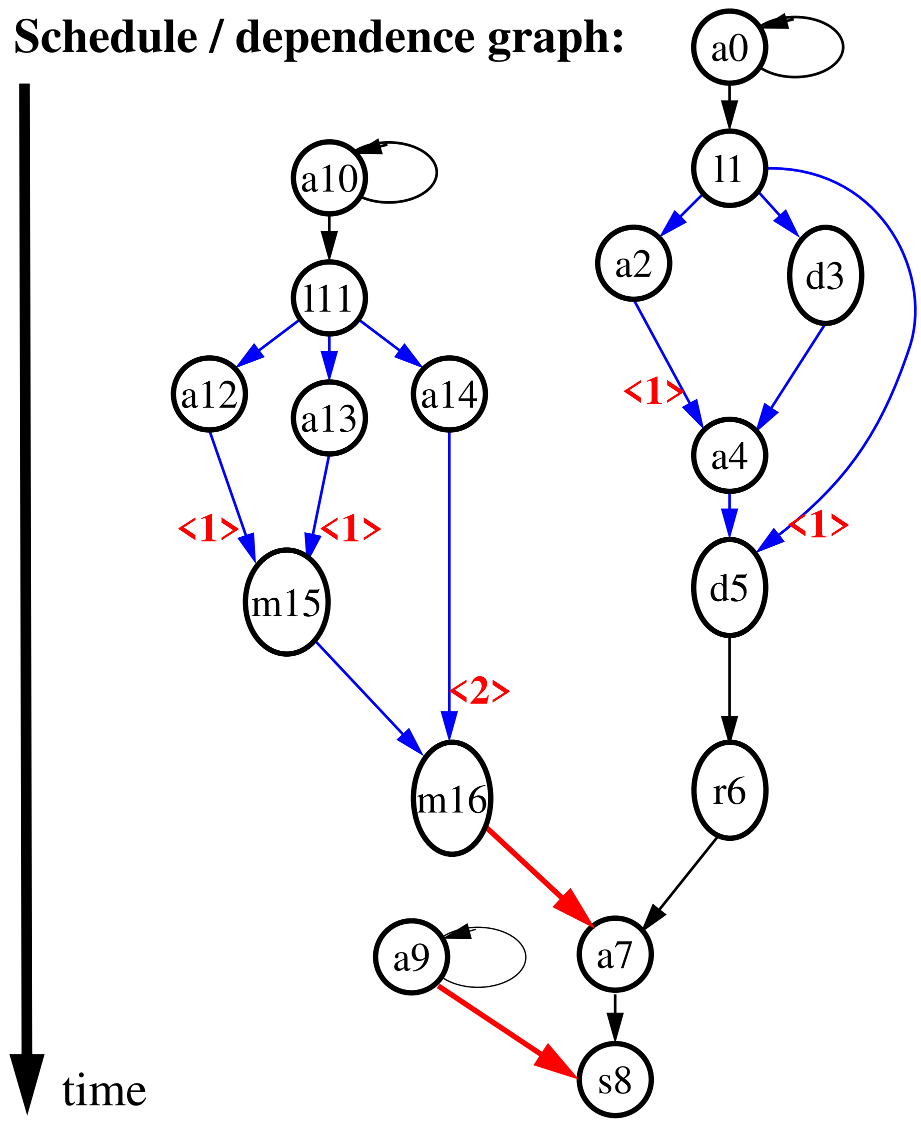
d5 long latency ops

→ dependences

→ dependences in underlying cycles

Acyclic Heuristic (AC)

Schedule / dependence graph:



for all edges not in an underlying cycle (black edges):

$$p_{i,j} = 0$$

modified edges

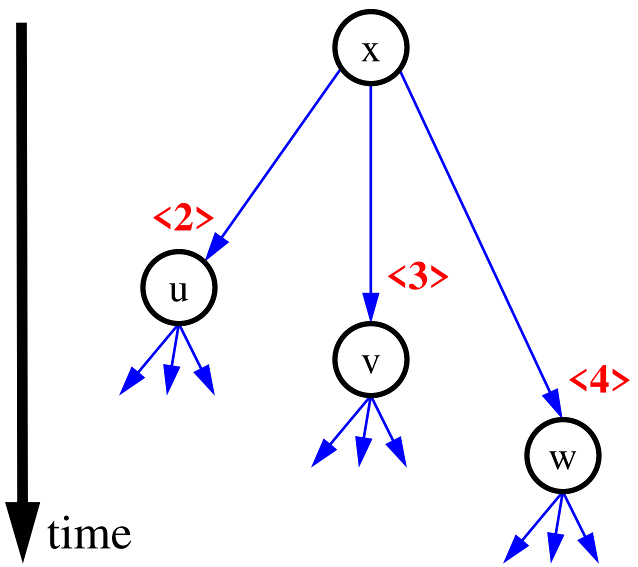
invariant1: $p_{i,j} \geq 0$

invariant2: sum of the $p_{i,j}$ along an underlying cycle remains constant

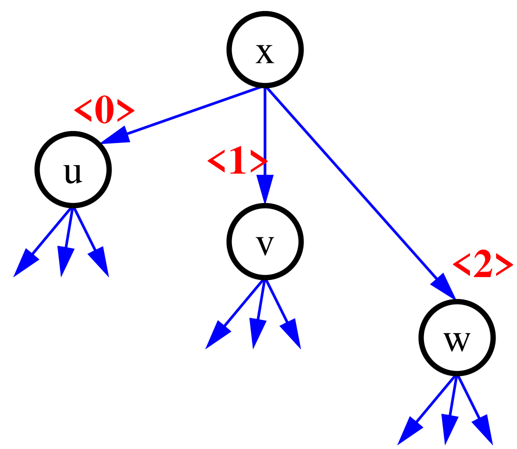
We now consider underlying cycles ONLY (blue edges)

Sink / Source Heuristics (SS)

Original schedule for source node:



Improved schedule for source node:



min-p := min($p_{i,j}$ among outgoing edges)

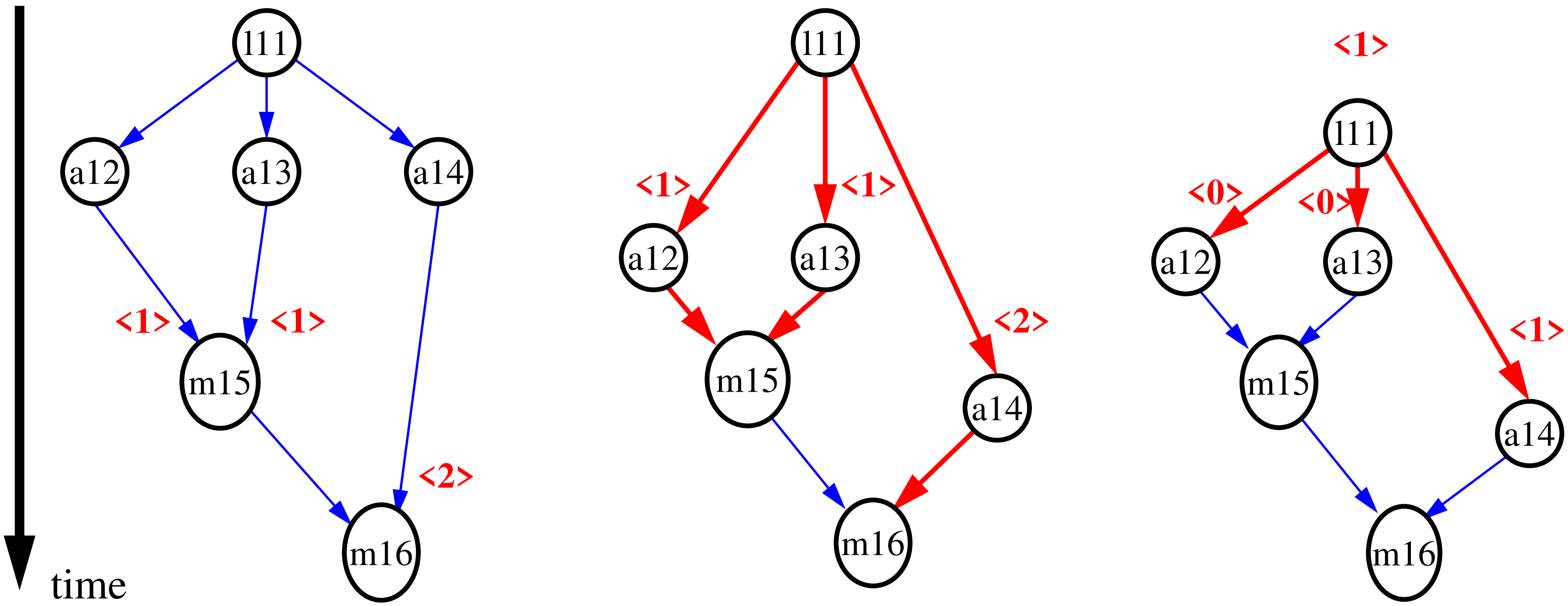
subtract ***min-p*** on each outgoing edge

invariant1: $p_{i,j} \geq 0$

invariant2: sum of the $p_{i,j}$ along an underlying cycle remains constant

Up Heuristic (UP)

Original schedule: **After 3 up-propagations:** **After 4 up-propagations:**



min-p := min($p_{i,j}$ among outgoing edges)

subtract ***min-p*** on each outgoing edge

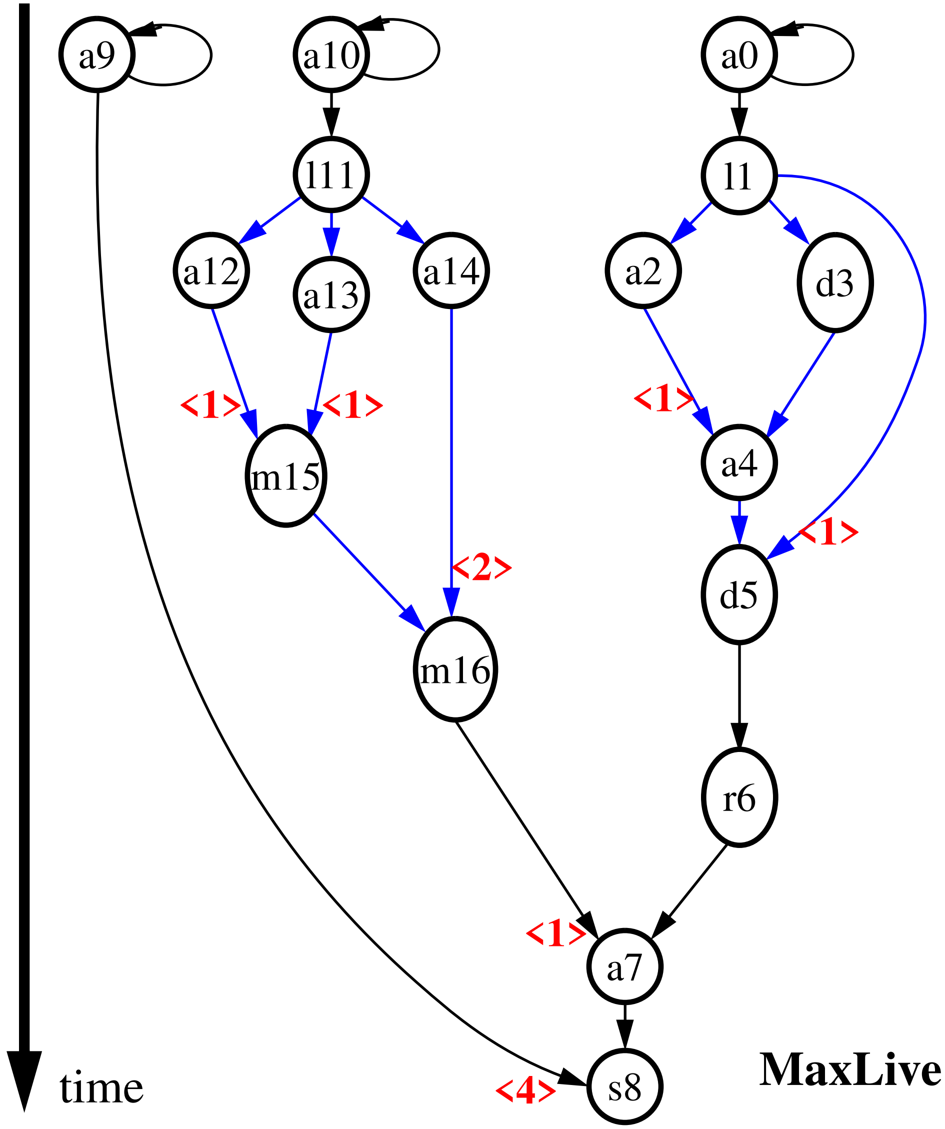
add ***min-p*** to each incoming edge

invariant1: $p_{i,j} \geq 0$

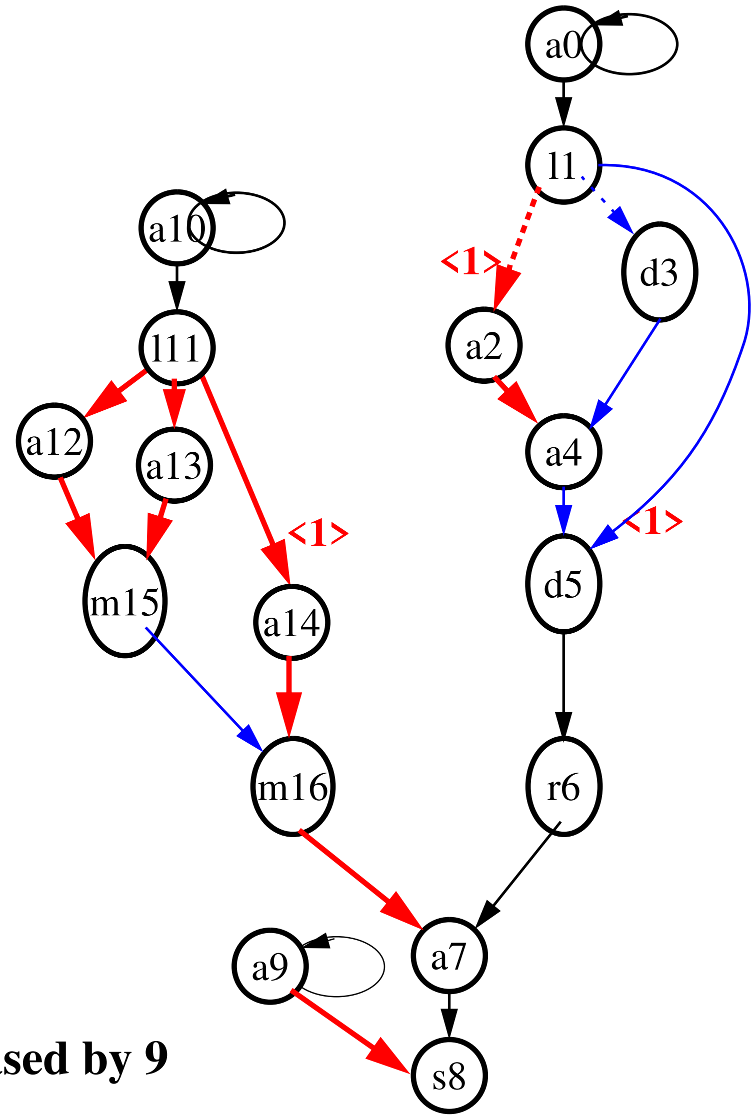
invariant2: sum of the $p_{i,j}$ along an underlying cycle remains constant

Running Example (cont.)

Schedule / dependence graph:



Improved schedule:



MaxLive decreased by 9

Benchmark Examples

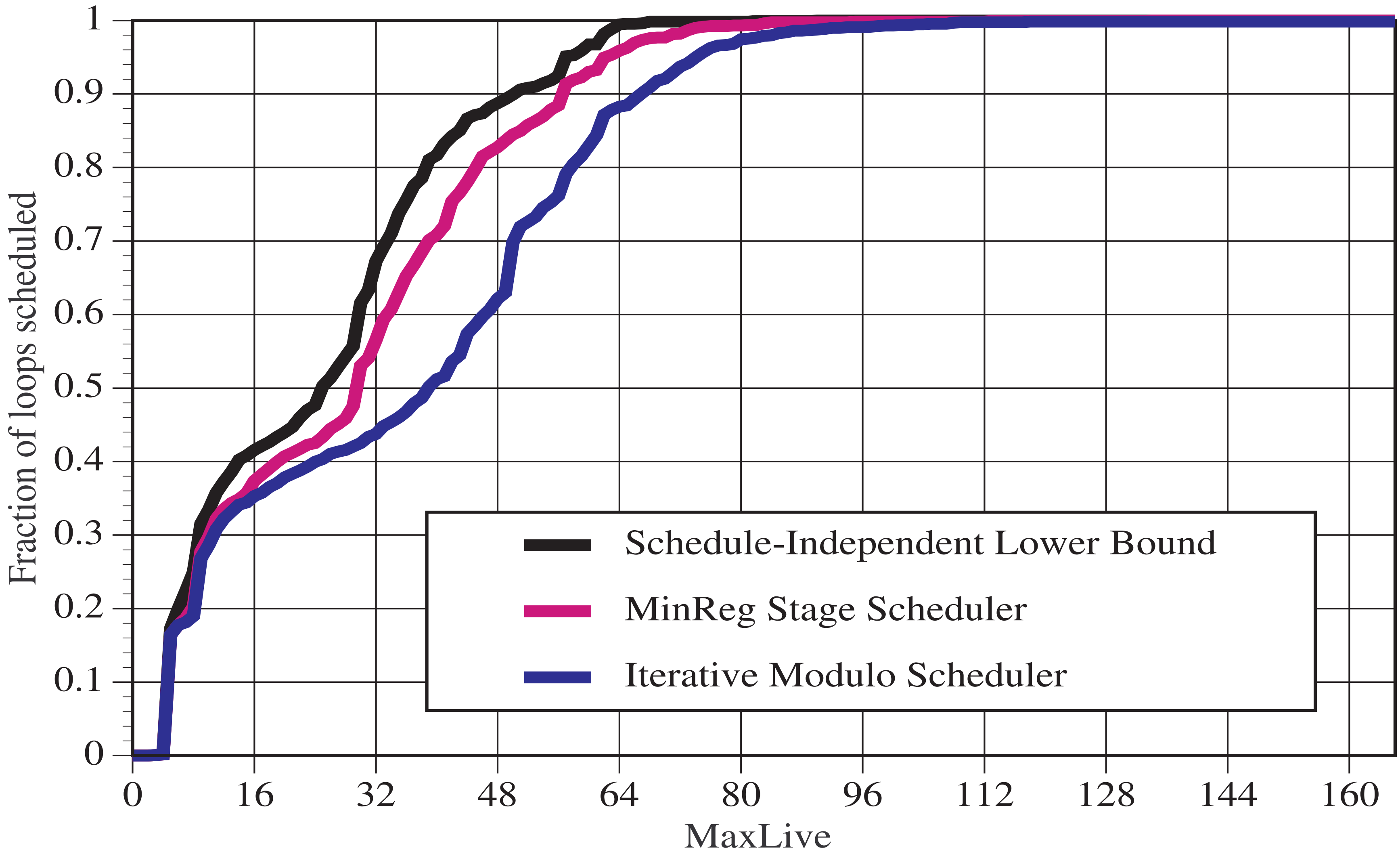
Benchmark suite:

- 1298 loops from the Perfect Club (1002), SPEC-89 (298), and Livermore (27)
- compiled for the Cydra 5 machine
- scheduled heuristically by the Iterative Modulo Scheduler [Rau, 94]
  - optimal *II* for 96% of the loop in the benchmark
  - no attempt to minimize register requirements
- compared to an optimal stage scheduler [Eichenberger *et al*, 94]
  - finds a stage schedule with minimum register requirements

Statistics:

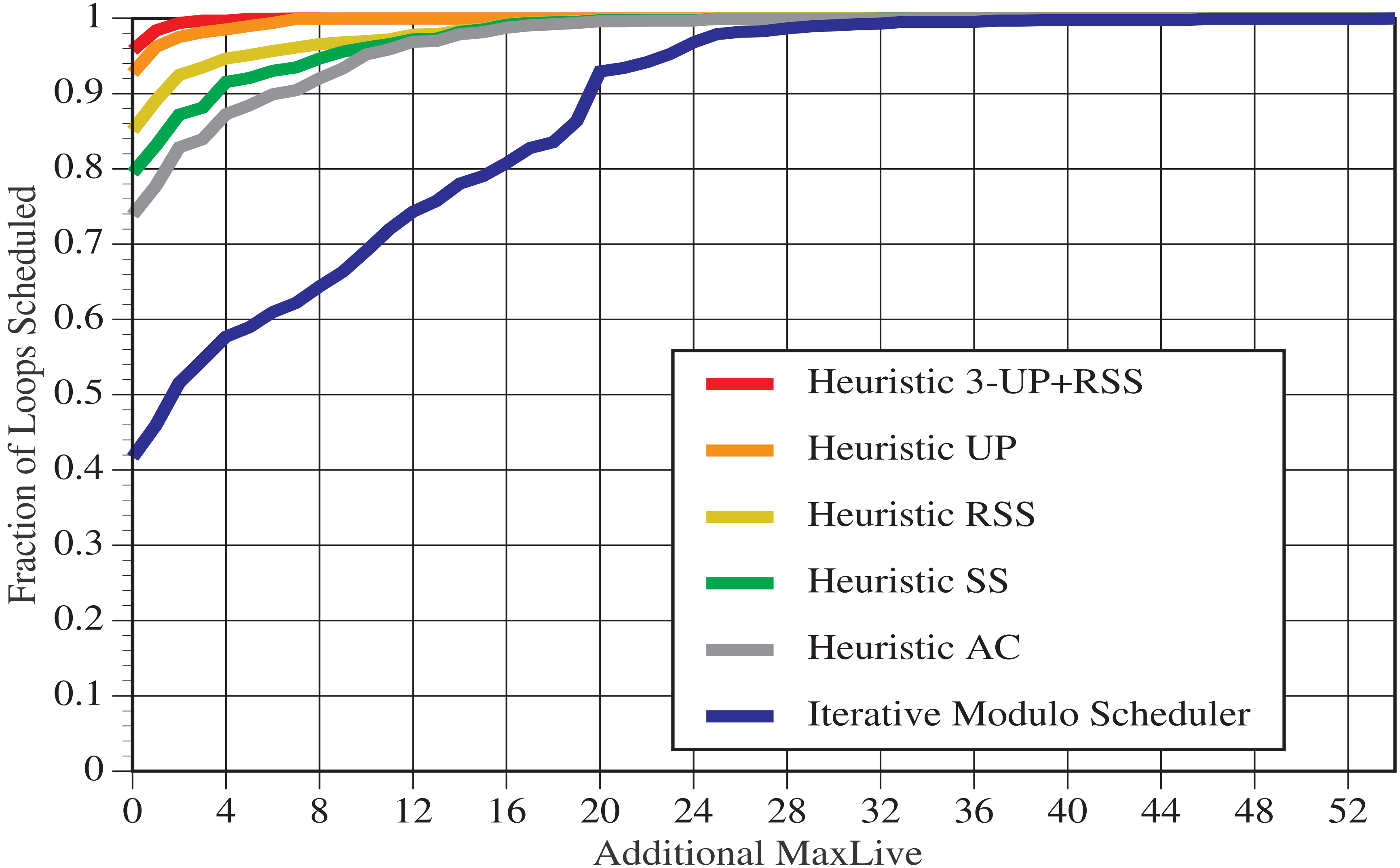
	ops	ops in underlying cycles	register edges	scheduling edges
min.	4	1	1	1
avg.	17.3	13.1	22.0	15.0
max.	161	160	220	345

MaxLive of Stage Scheduling Heuristics



Lower-Bound: schedule independent, ignore resource constraints [Huff, 93]

Additional MaxLive of Stage Scheduling Heuristics



Relative to: Optimal Stage Scheduler

Conclusions

Results for Stage Scheduling:

- optimal stage scheduling decreases register requirements by 24.9%
- best heuristic (3-UP + RSS) decreases register requirements by 24.6%
  - on average 99% of the optimal decrease
  - complexity: O($\boldsymbol{e}^2$)
- good heuristic (UP) decreases register requirements by 24.1%
  - on average 97% of the optimal decrease
  - complexity: O($\boldsymbol{e}$)
- simple heuristic (SS) decreases register requirements by 19.7%
  - on average 82% of the optimal decrease
  - complexity: O($\boldsymbol{e}$)

Conclusions:

- simple, efficient heuristics to reduce the register requirements
- no steady-state performance penalty

Acknowledgments

The authors would like to thank Santosh Abraham, Sadun Anik, Richard Johnson, Vinod Kathail, Michael Schlansker, and Bob Rau for their many useful suggestions and for providing the input data set.