Exploring the Potential of Instruction-Level Parallelism of Exposed Datapath Architectures with Buffered Processing Units

Anoop Bhagyanath and Klaus Schneider

Embedded Systems Chair
University of Kaiserslautern

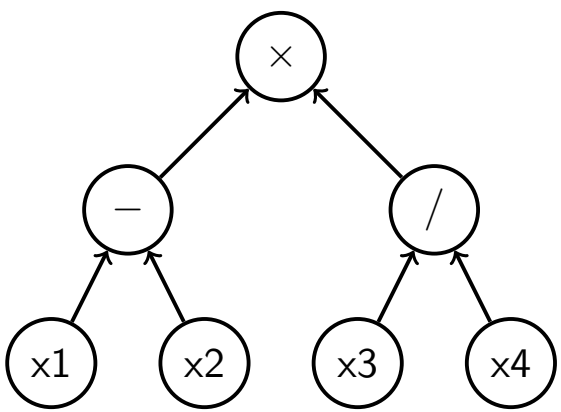
ACSD 2017

Outline

Motivation

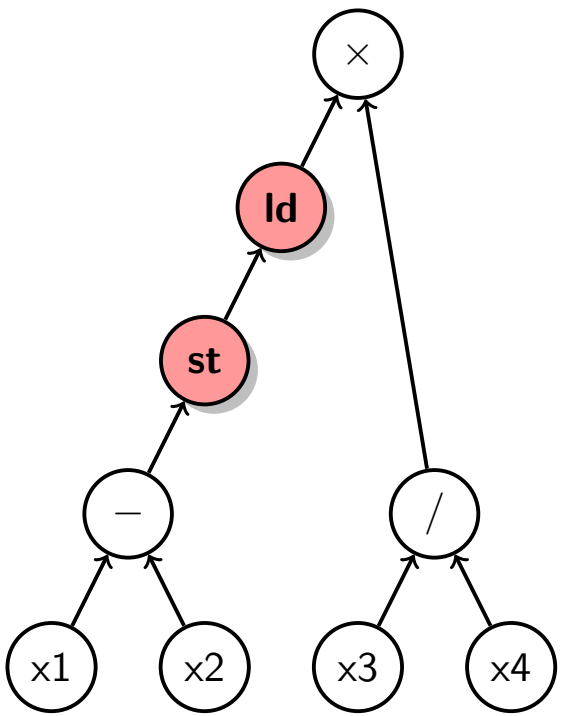
Instruction Level Parallelism (ILP)

expression tree



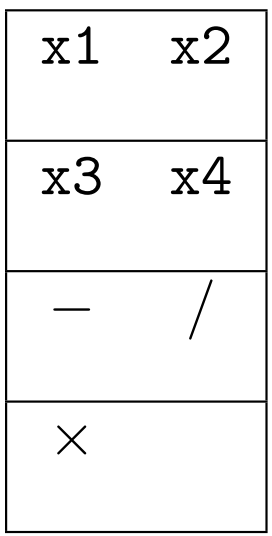
3 steps

dataflow graph (2 Regs)



5 steps

VLIW (4R 2W ports)

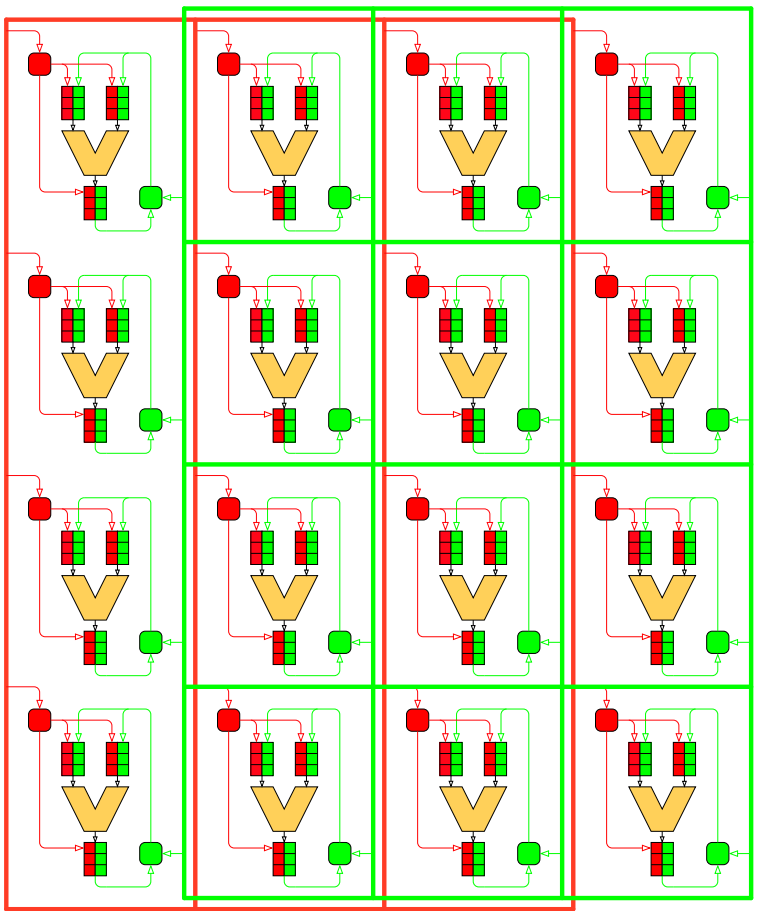


4 steps

Conventional Architectures

- ILP restricted due to limited number of registers and ports in register file
  - compiler spills variables to main memory
  - number of instructions packed into a VLIW word
- increasing number of registers is difficult
  - instruction format encoding
  - register file wiring

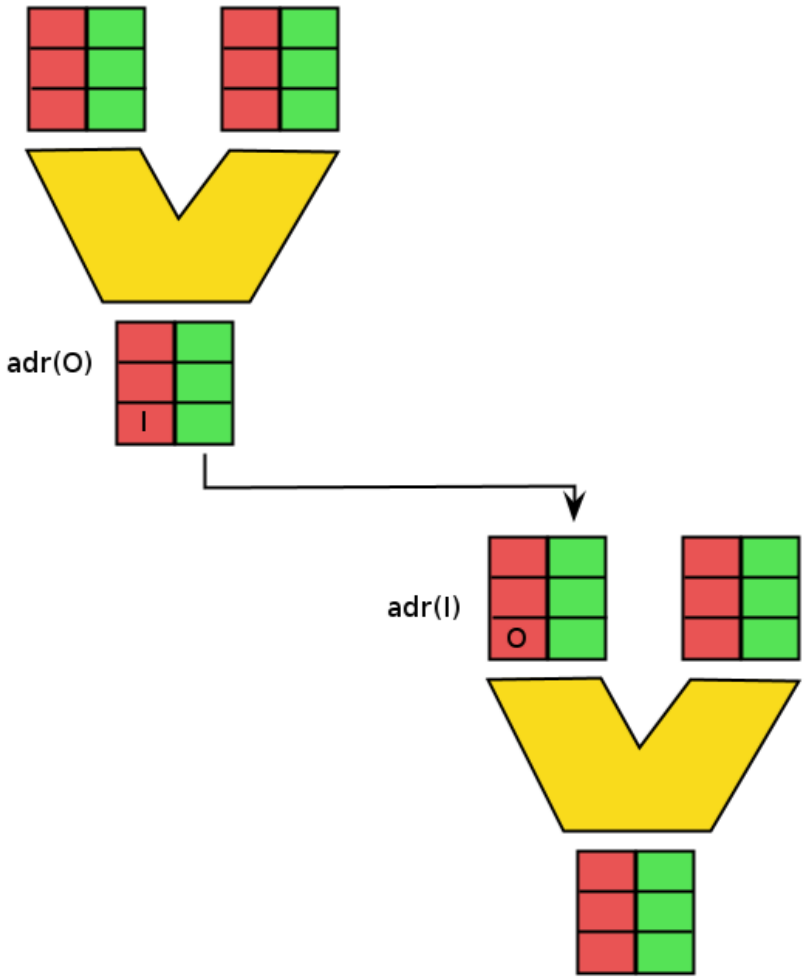
Exposed Datapath Architectures



Sync (SCAD) Control Async Dataflow

- grid of processing units
- FIFO buffers (queues) at inputs and outputs of PUs
- compiler also moves values from one PU to another: **bypass registers**
- although bypassing is used, the code generators still use register mappings
- examples: TTA, MOVE-PRO, TRIPS, Wavescalar, STA, Flexcore etc

SCAD Architecture

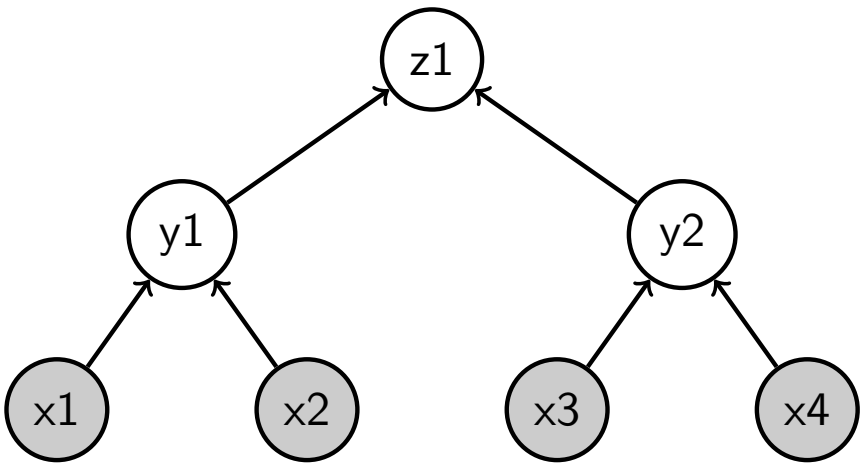


move instruction $O \rightarrow I$

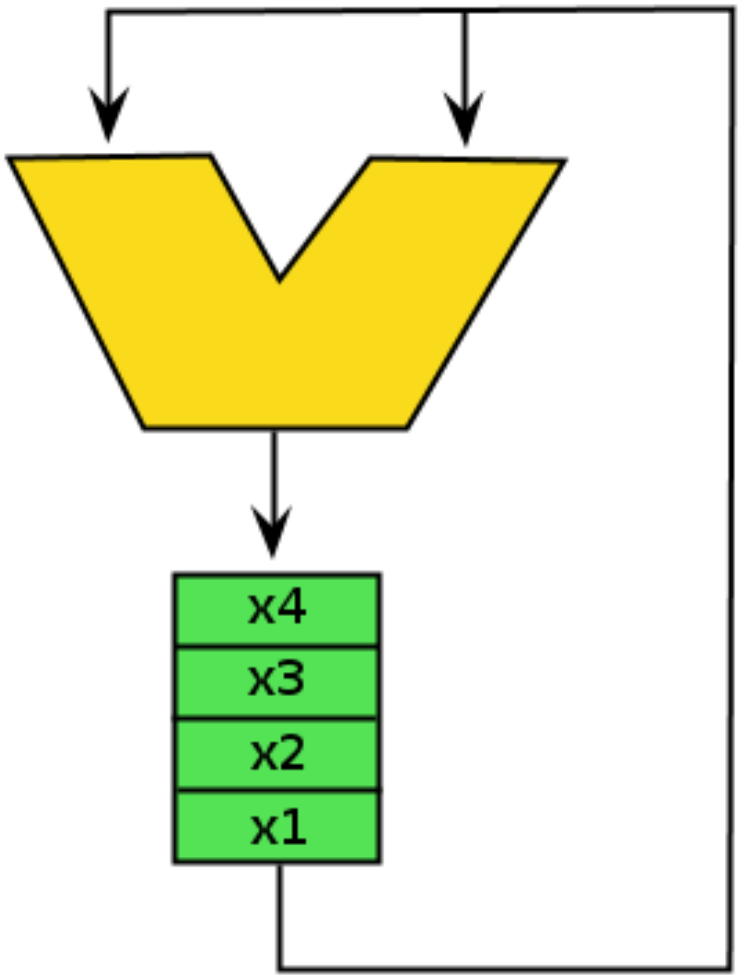
- move instructions $O \rightarrow I$
- move instruction bus (MIB) fills address slots
- PU fires if enough data available at input buffer heads
- data transport network (DTN) fills data slots
- application-specific
  - any arbitrary functionality in PUs
  - interconnect choice

Queue-based Code Generation

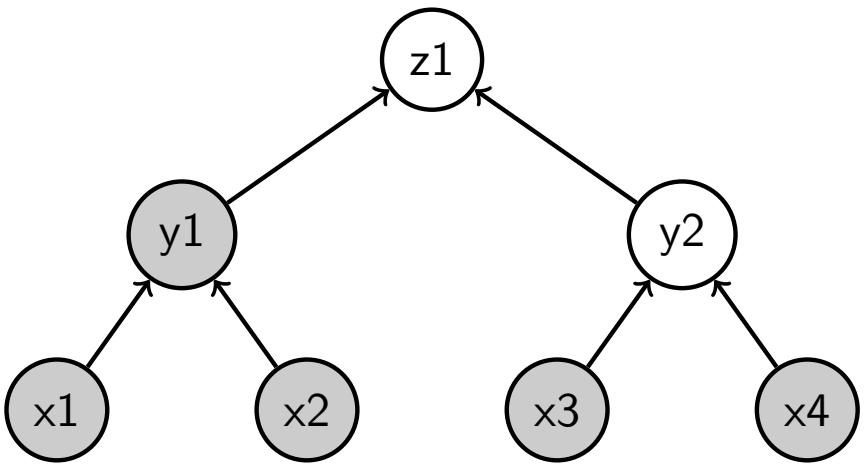
Code Generation for Queue Machine



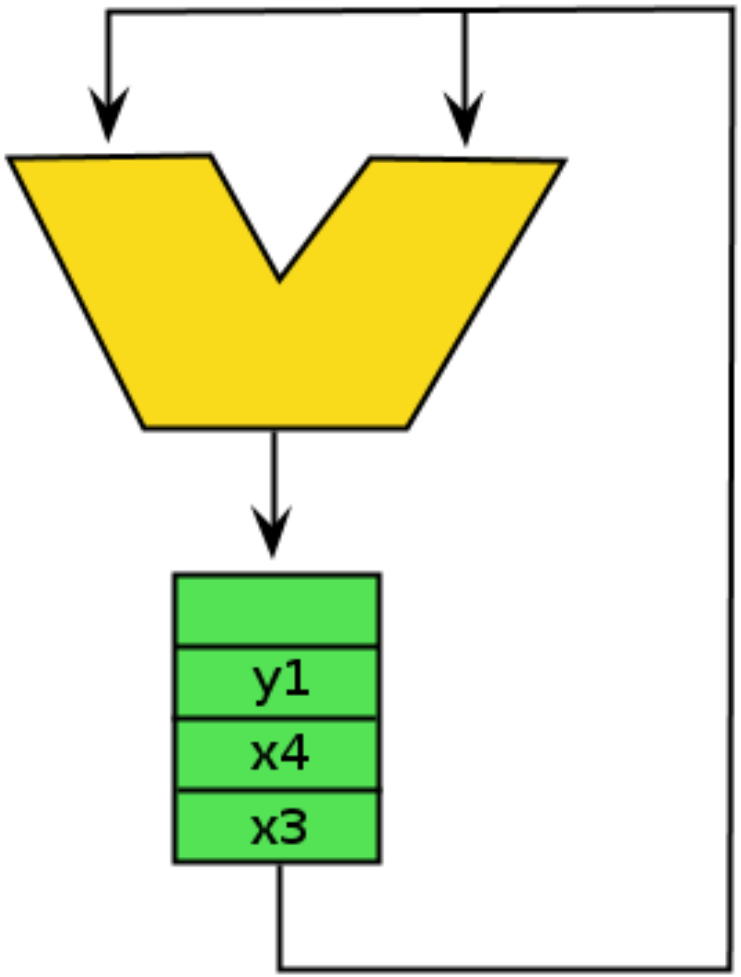
executed `x1,x2,x3,x4`



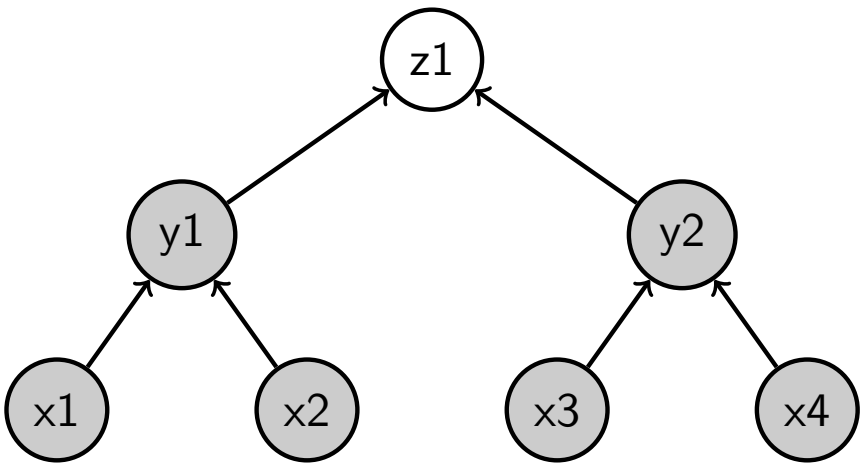
Code Generation for Queue Machine



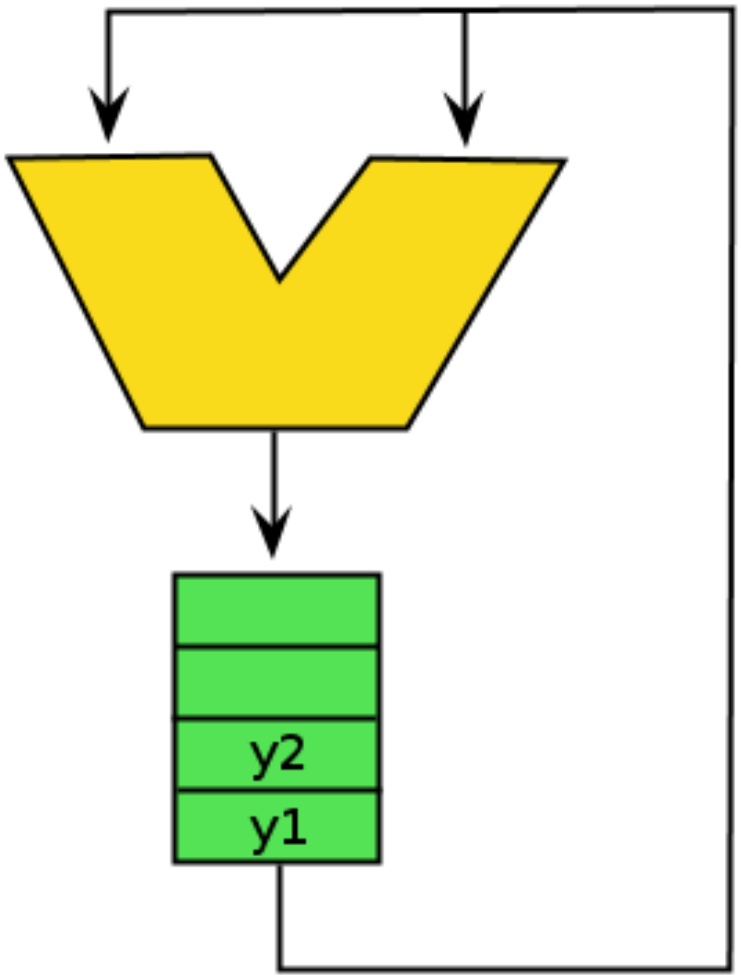
executed y1



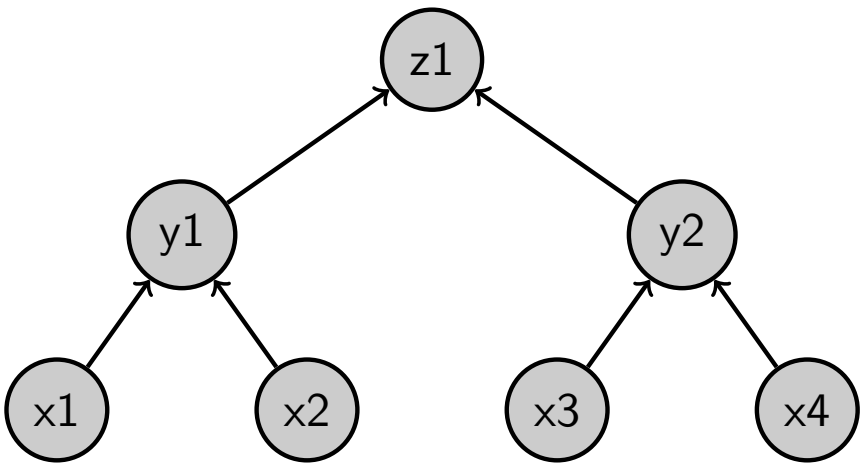
Code Generation for Queue Machine



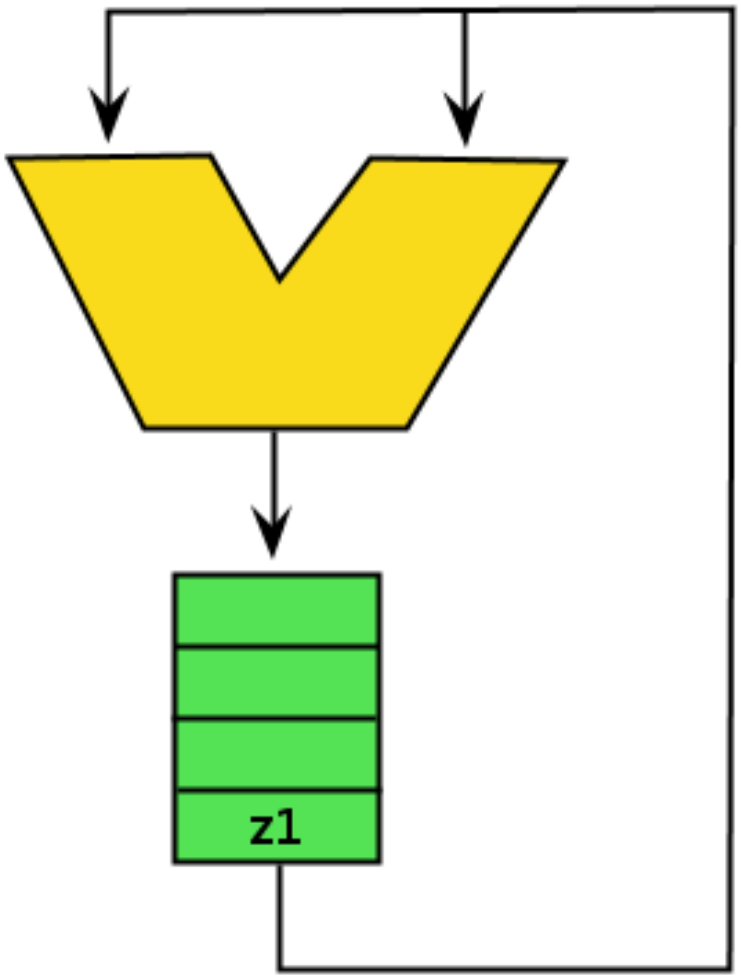
executed y2



Code Generation for Queue Machine

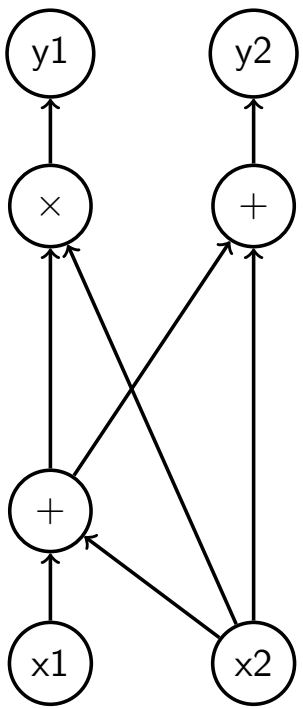


executed z1



Computation Overhead for Basic Blocks

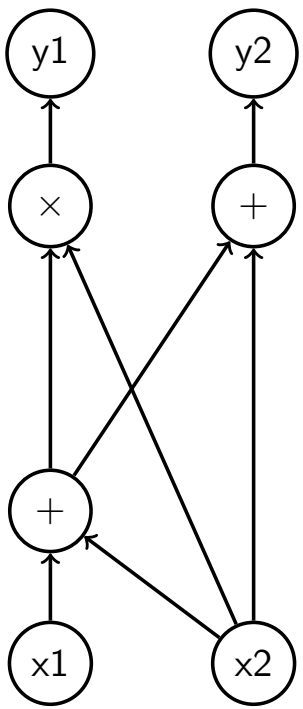
given **DAG**	levelized DAG	planar DAG	level-planar DAG	queue program

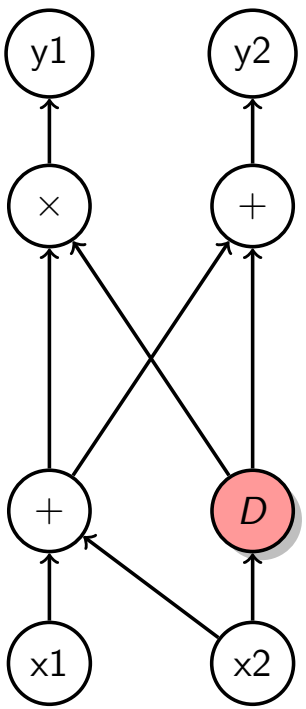



```
load x1,1
load x2,2
add 2
dup 2
dup 1
swap
dup 1
mul 1
add 1
store y1
store y2
```

Computation Overhead for Basic Blocks

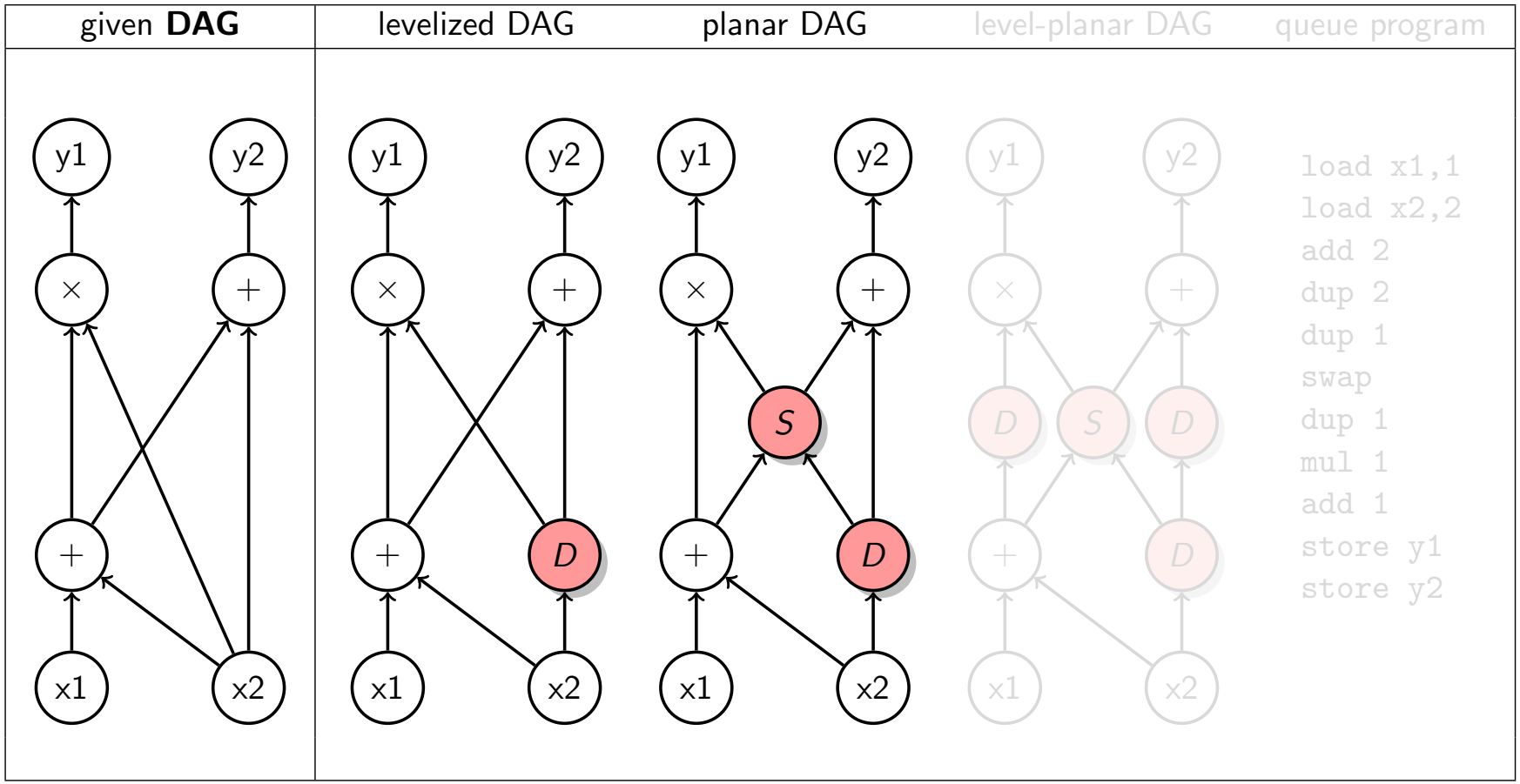
given **DAG**	levelized DAG	planar DAG	level-planar DAG	queue program



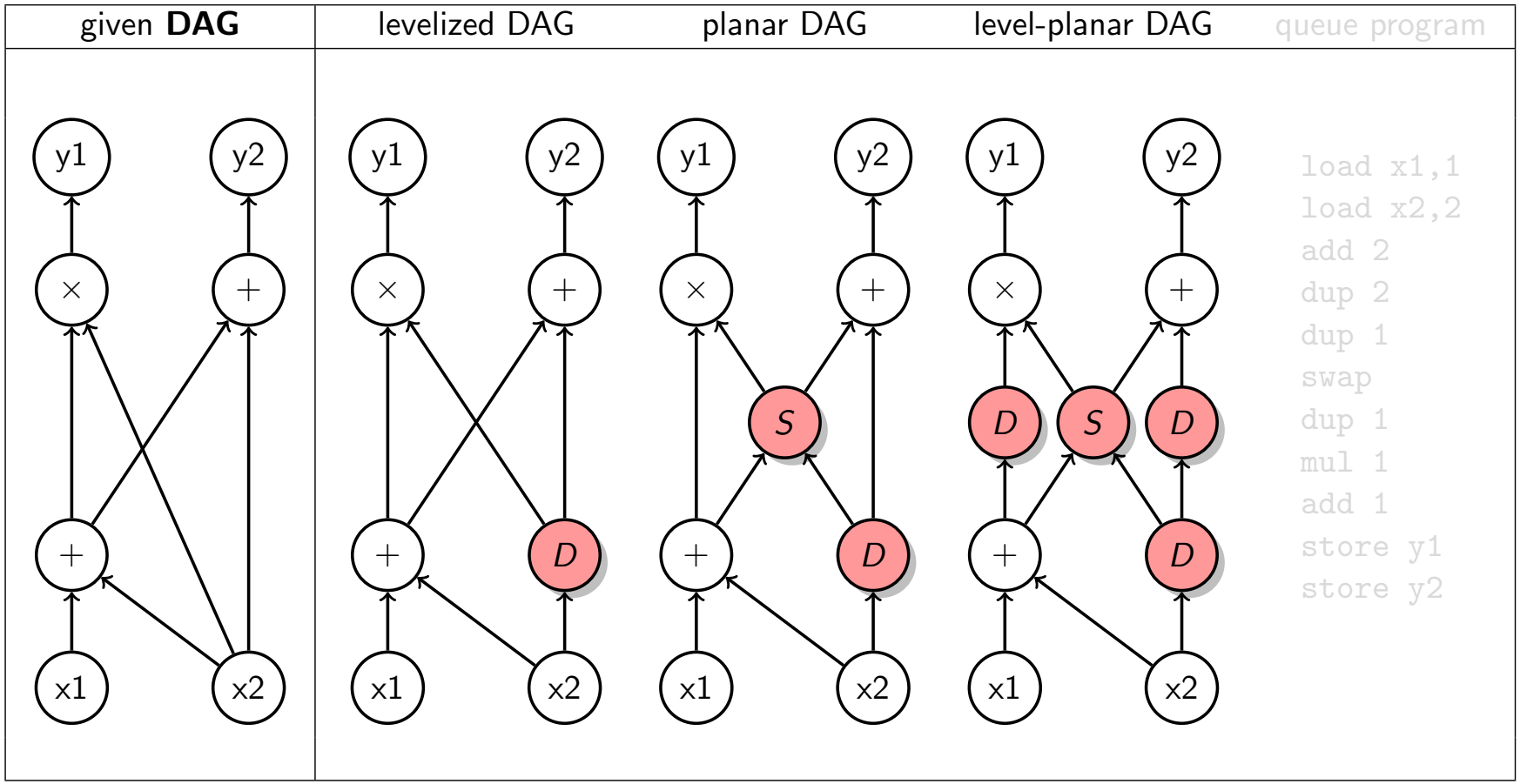



```
load x1,1
load x2,2
add 2
dup 2
dup 1
swap
dup 1
mul 1
add 1
store y1
store y2
```

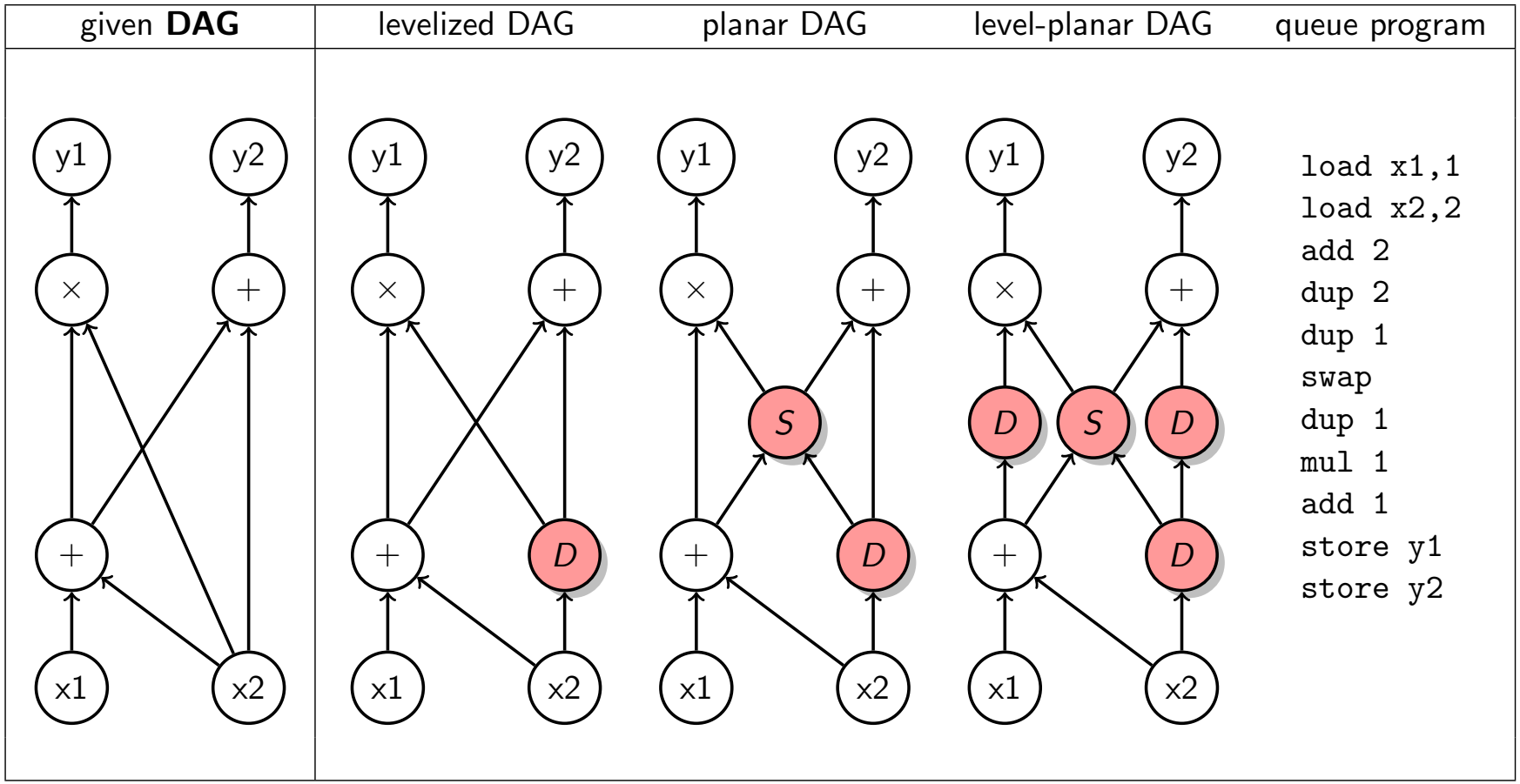
Computation Overhead for Basic Blocks



Computation Overhead for Basic Blocks



Computation Overhead for Basic Blocks

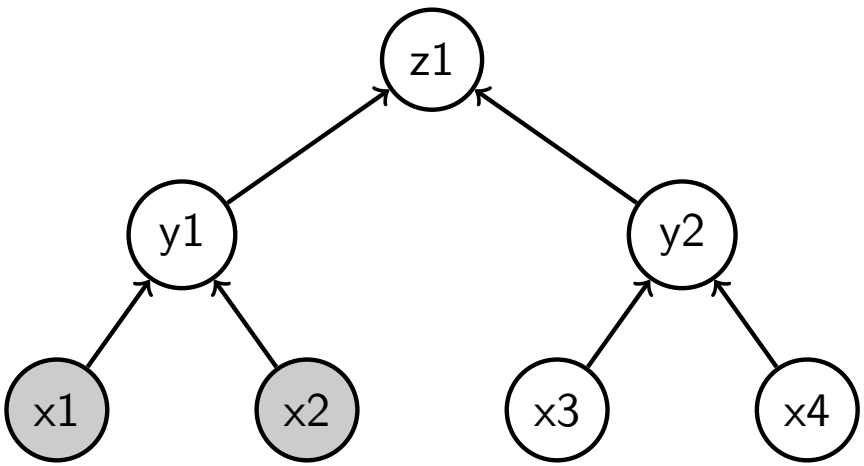


Depth-First Traversal

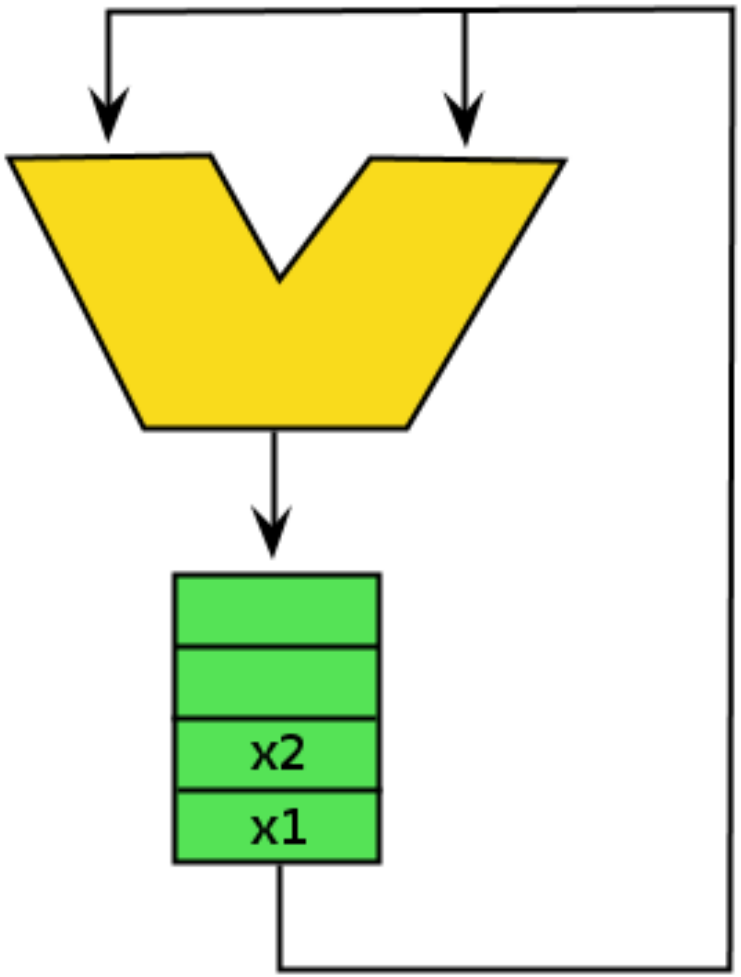
Current Compilers

- order nodes by depth-first traversal
- minimize register usage
- optimal code for expression trees
  - Sethi-Ullmann algorithm
  - polynomial time
- optimal code for directed-acyclic graphs (DAG)
  - proved to be NP-Complete

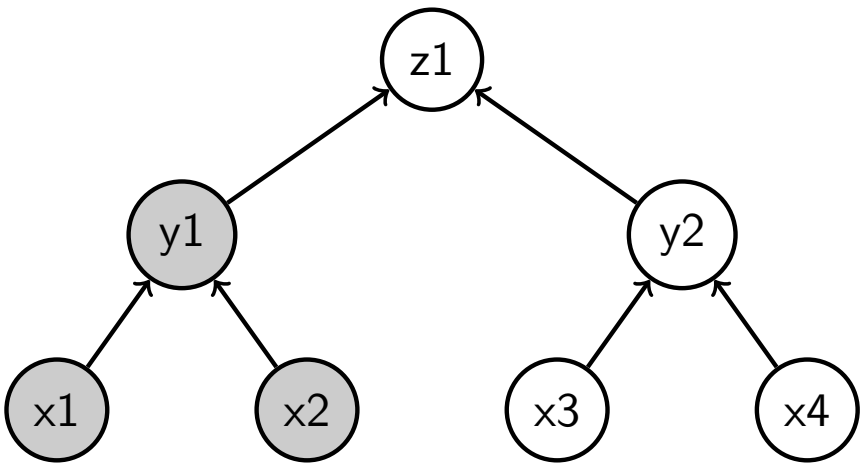
DFT in Queue Machine



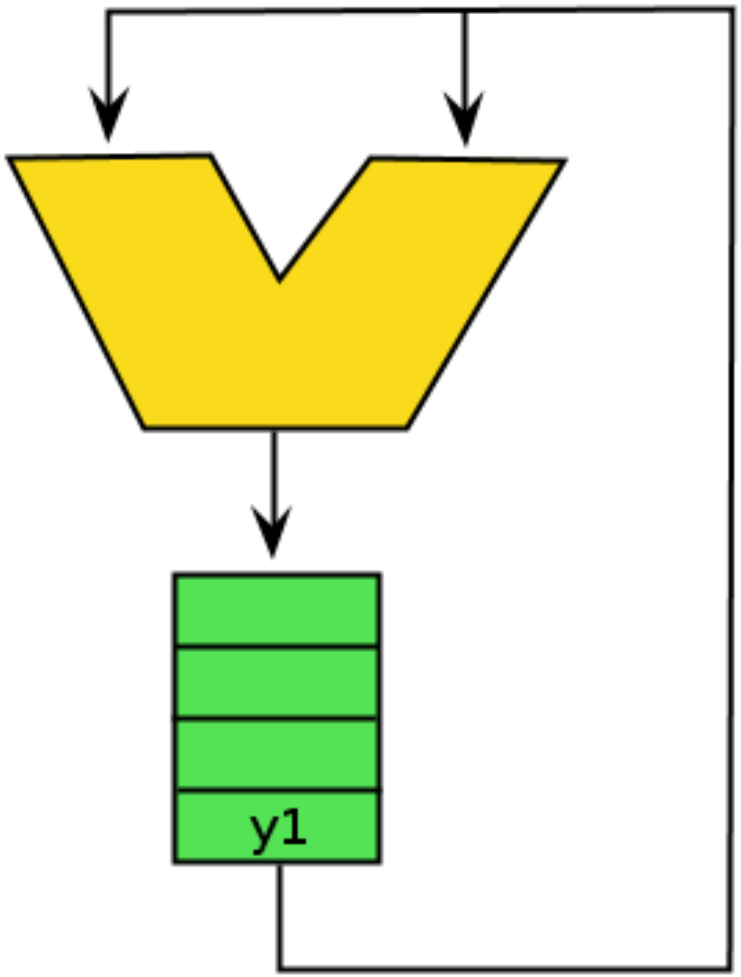
executed `x1,x2`



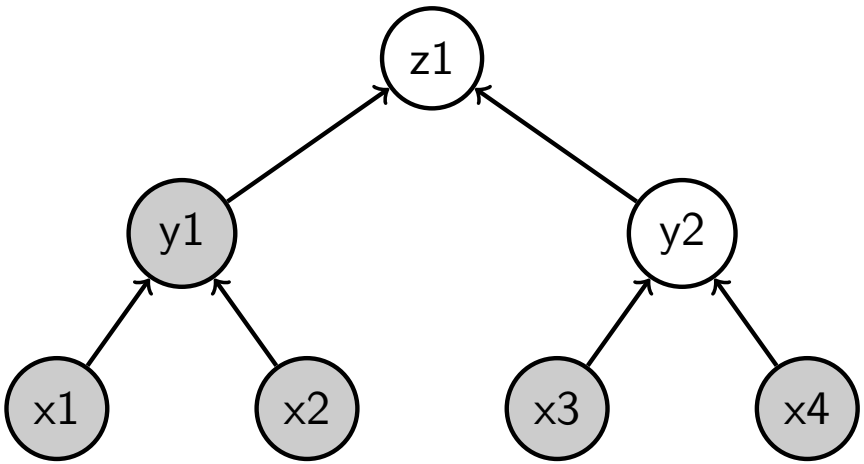
DFT in Queue Machine



executed y1

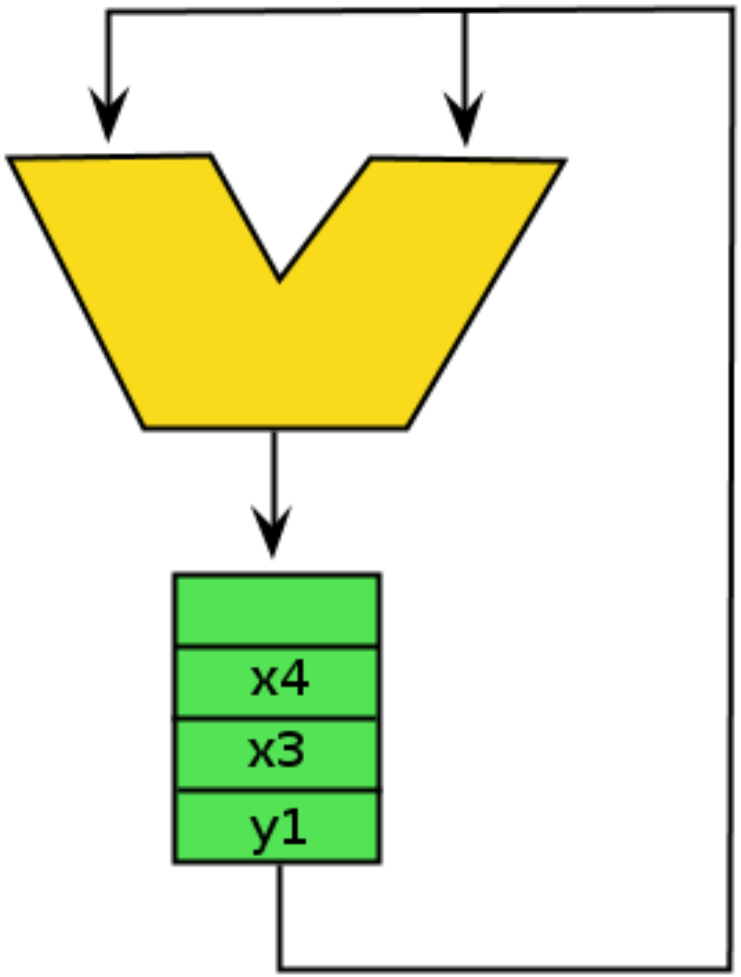


DFT in Queue Machine



executed `x3,x4`

wrong order to execute y2!

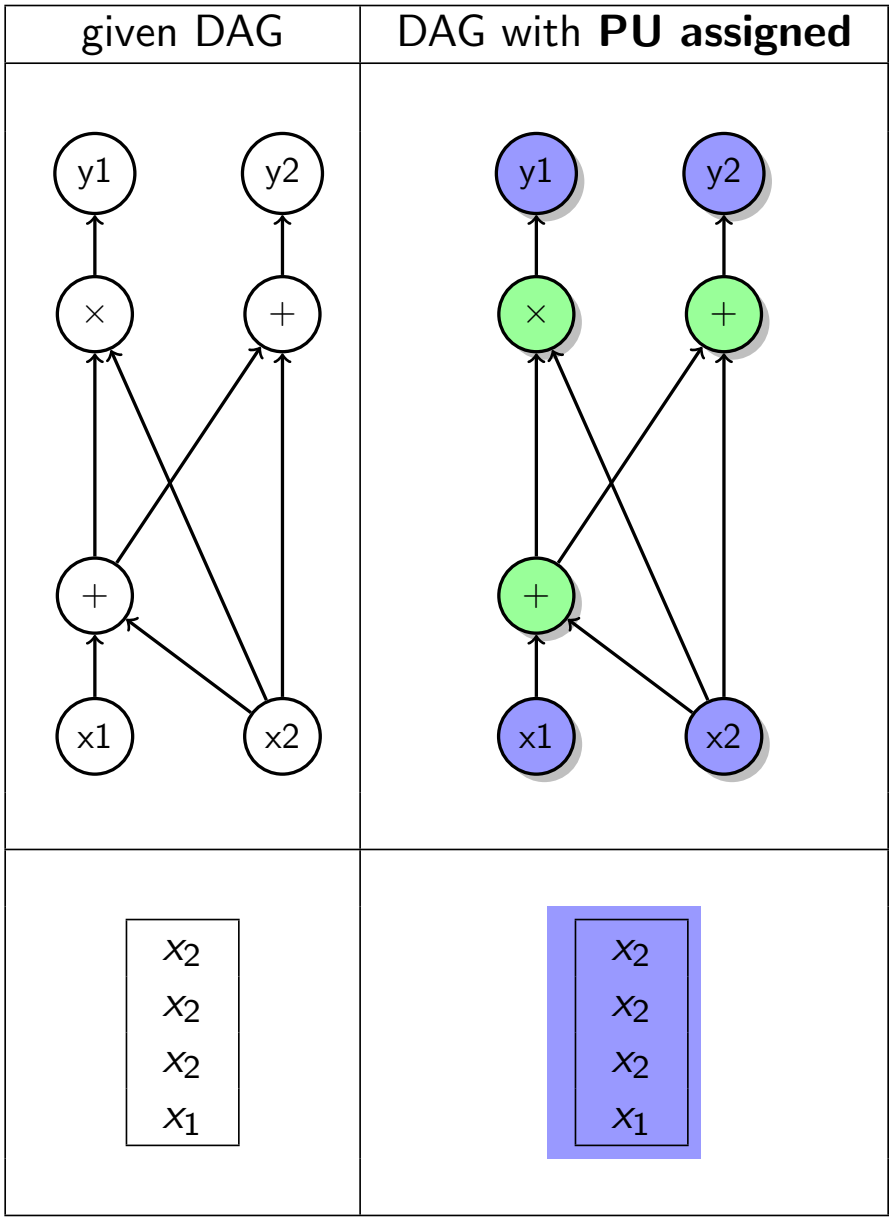


Queue code to SCAD code *(MBMV)*

Queue Instruction	Corresponding SCAD Move Instructions
⋮	⋮
(load x1,1)	[x1->inp1; load->opc; 1->cps]
⋮	⋮
(add 1)	[out->inp1; out->inp2; add->opc; 1->cps]
⋮	⋮
(dup 2)	[out->inp1; dup->opc; 2->cps]
⋮	⋮
(swap)	[out->inp1; out->inp2; swap->opc]
⋮	⋮
(store y1)	[y1->inp1; out->inp2; store->opc]
⋮	⋮

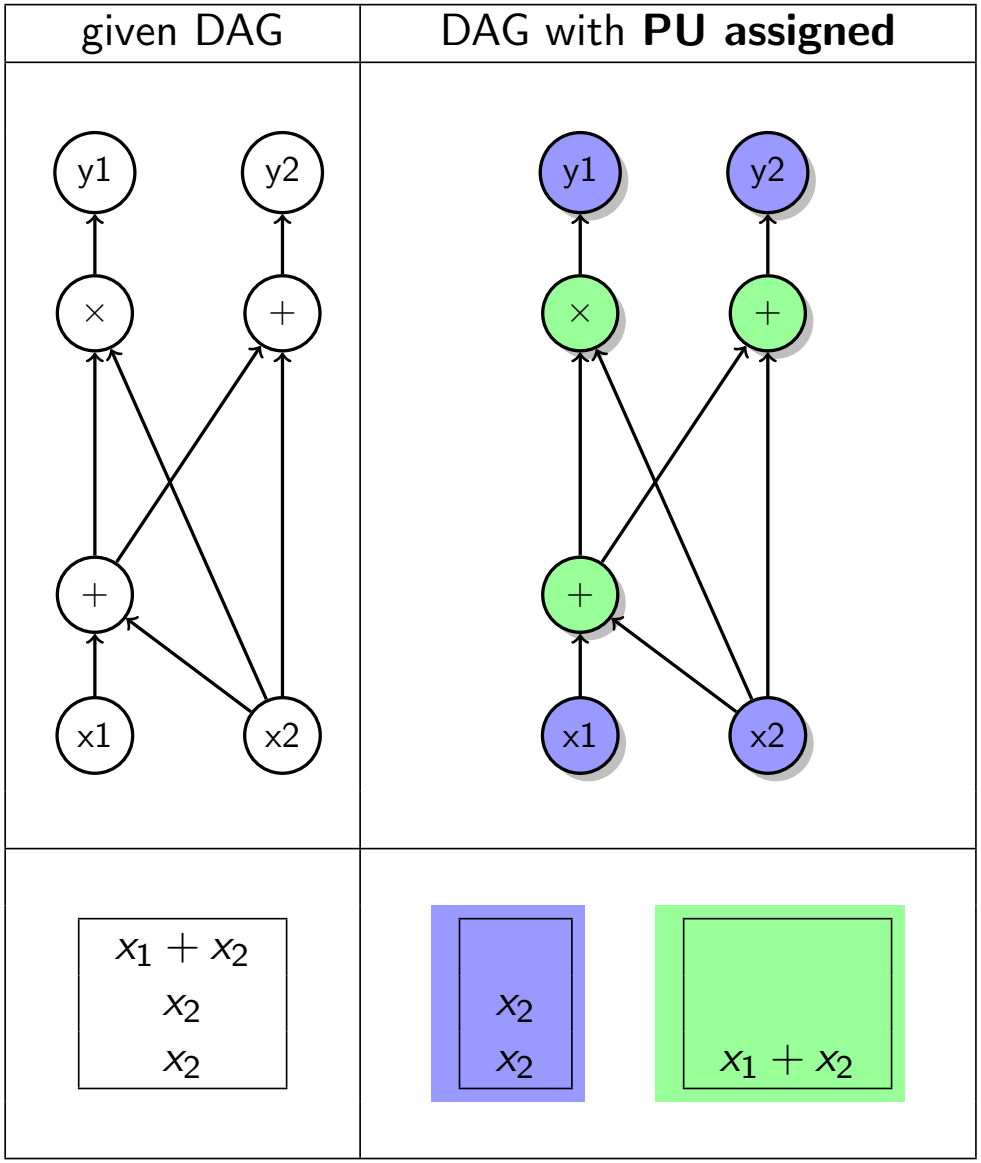
is the SCAD code optimal ?

Reduced Computation Overhead for SCAD



- Queue machine: One queue
  - single total order of all nodes

Reduced Computation Overhead for SCAD



- SCAD machine: Multiple queues
  - multiple partial orders of nodes
  - less computation overhead
- SAT based SCAD code generation *(MEMOCODE)*
  - resource-optimal
  - at most 4 PUs for up to 15 instruction basic blocks

Mapping to SMT

Problem statement

Given a basic block (in three-address SSA code), a SCAD machine with p universal PUs and 1 load-store unit, a desired execution time t:

- determine if the basic block can be executed on the SCAD machine in time t without any computation overhead.

Relations

α_{ij}	variable x_i is assigned to PU j
$\theta_{i,j}$	variable x_i is scheduled in timeslot j

Mapping to SMT

Problem statement

Given a basic block (in three-address SSA code), a SCAD machine with p universal PUs and 1 load-store unit, a desired execution time t:

- determine if the basic block can be executed on the SCAD machine in time t without any computation overhead.

Relations

α_{ij}	variable x_i is assigned to PU j
$\theta_{i,j}$	variable x_i is scheduled in timeslot j

Constraints

Binary values

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{p} 0 \leq \alpha_{i,j} \leq 1 \text{ and } \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{t-1} 0 \leq \theta_{i,j} \leq 1 \quad (1)$$

Schedule exactly once

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{t-1} \theta_{i,j} = 1 \quad (2)$$

Unique PU assignment

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{p} \alpha_{i,j} = 1 \quad (3)$$

Constraints

Binary values

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{p} 0 \leq \alpha_{i,j} \leq 1 \text{ and } \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{t-1} 0 \leq \theta_{i,j} \leq 1 \quad (1)$$

Schedule exactly once

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{t-1} \theta_{i,j} = 1 \quad (2)$$

Unique PU assignment

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{p} \alpha_{i,j} = 1 \quad (3)$$

Constraints

Binary values

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{p} 0 \leq \alpha_{i,j} \leq 1 \text{ and } \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{t-1} 0 \leq \theta_{i,j} \leq 1 \quad (1)$$

Schedule exactly once

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{t-1} \theta_{i,j} = 1 \quad (2)$$

Unique PU assignment

$$\bigwedge_{i=0}^{n-1} \sum_{j=0}^{p} \alpha_{i,j} = 1 \quad (3)$$

Constraints ...

Data dependency

For each node x_i, τ_i is the time slot in which the node is scheduled

$$\tau_i = \sum_{j=0}^{t-1} j \times \theta_{i,j}$$

For every instruction $x_t = x_l \odot x_r$,

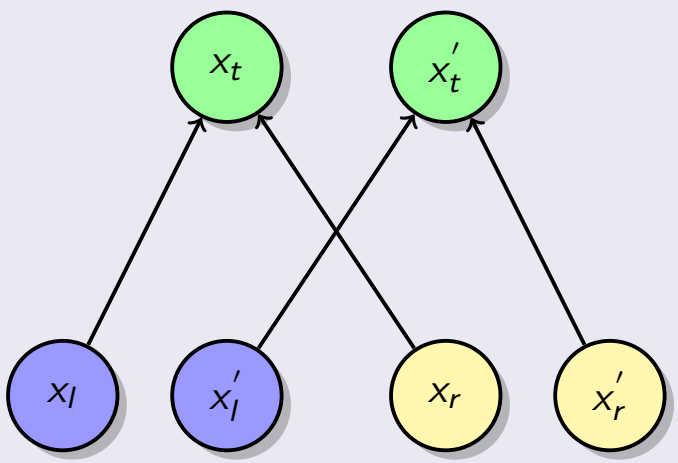
$$\begin{aligned} \tau_t - \tau_l &\geq \delta_l \\ \tau_t - \tau_r &\geq \delta_r \end{aligned} \tag{4}$$

Constraints ...

Ordering variables in buffers

for every pair of instructions:

$$x_t = x_l \odot x_r$$
$$x_t^{'} = x_l^{'} \odot x_r^{'}$$

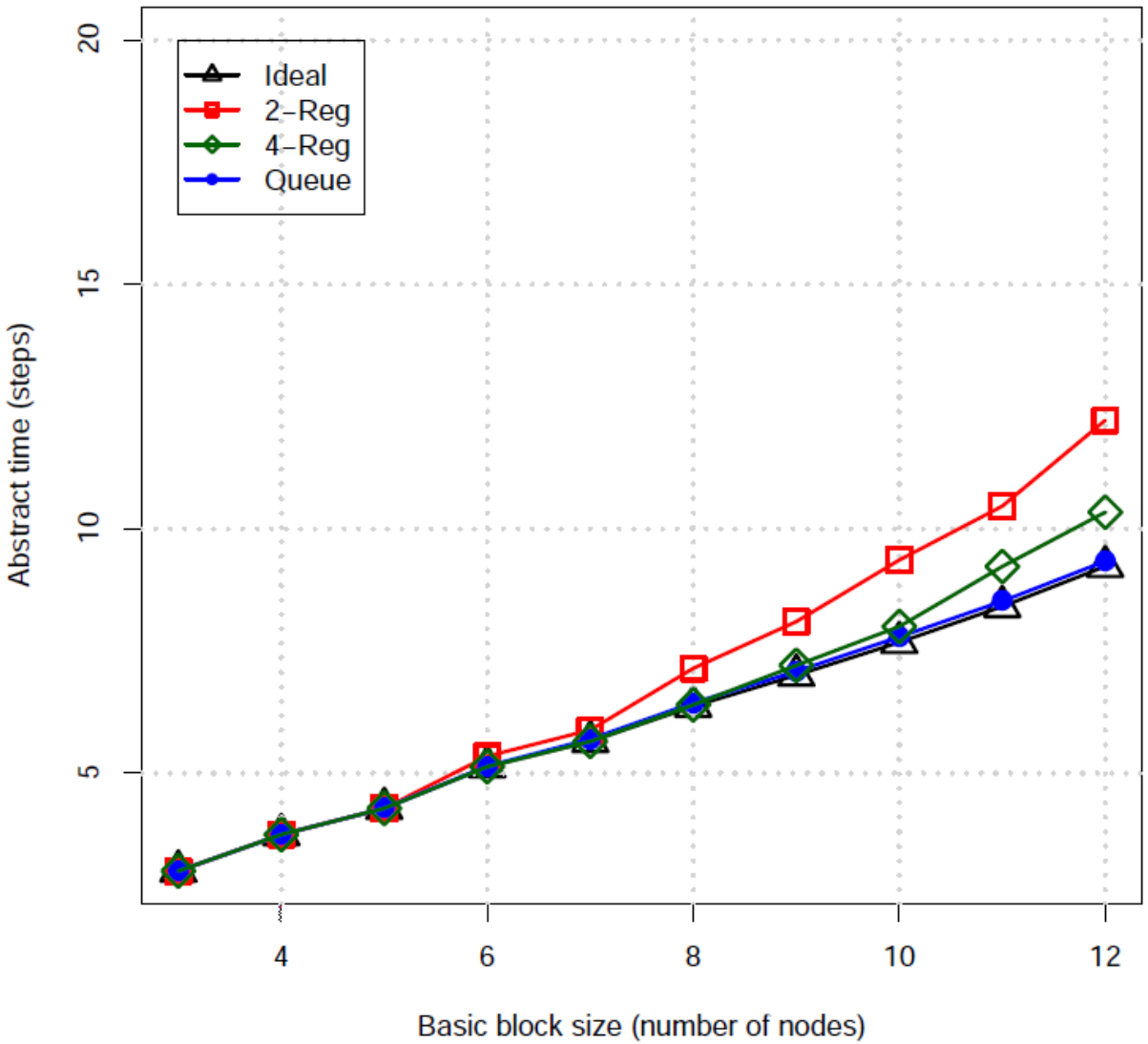


Buffer constraint

$$\beta_{t,t'} \implies \left(\begin{array}{l} \left(\tau_t < \tau_{t'}\right) \wedge \\ \left(\beta_{l,l'} \implies \tau_l \leq \tau_{l'}\right) \wedge \\ \left(\beta_{r,r'} \implies \tau_r \leq \tau_{r'}\right) \end{array} \right) \vee \left(\begin{array}{l} \left(\tau_{t'} < \tau_t\right) \wedge \\ \left(\beta_{l,l'} \implies \tau_{l'} \leq \tau_l\right) \wedge \\ \left(\beta_{r,r'} \implies \tau_{r'} \leq \tau_r\right) \end{array} \right) \quad (5)$$

Preliminary Results

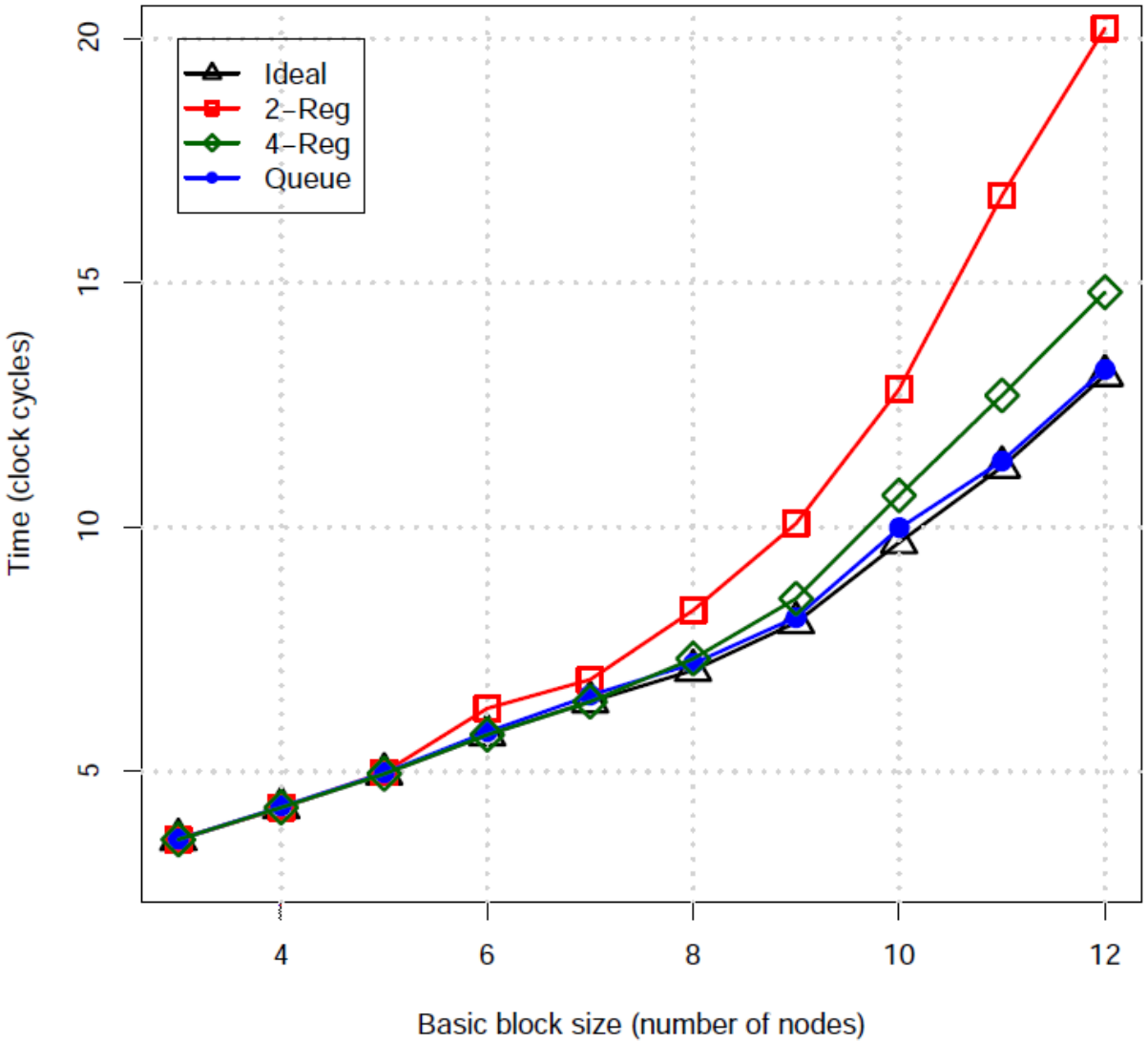
Performance



unit latency for all nodes in the basic block

Preliminary Results ...

Performance



90% cache hit probability
1 cycle: hit
10 cycles: miss

Future Work

- analyze buffer-sizes
- hardness of optimal code generation problem
- efficient heuristics

Thank You!

Questions?