Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture

Anoop Bhagyanath, Tripti Jain and Klaus Schneider

Embedded Systems Group
University of Kaiserslautern

MBMV, 2016

Outline

Motivation

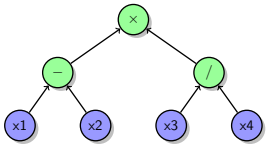
Instruction Level Parallelism (ILP)

Superscalar and VLIW Machines

- ILP restricted due to limited number of registers
  - instruction format encoding
  - register file wiring
- Compiler spills variables to main memory
- Number of instructions packed into a VLIW word

Instruction Level Parallelism (ILP)

Expression tree



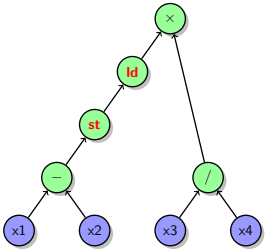
3 steps

VLIW code(2 regs)

x1	x2
−	x3
st	
x4	
ld	/
×	

6 steps

Superscalar code(2 regs)



5 steps

Exposed Datapath Architectures

- Compiler also controls the data transport
  - **bypass registers**
- Examples include Raw, TRIPS, Wavescalar, Flexcore, TTAs etc
- Code generators still rely on efficient register mappings
  - register bypassing utilized at later stages

Our Contribution

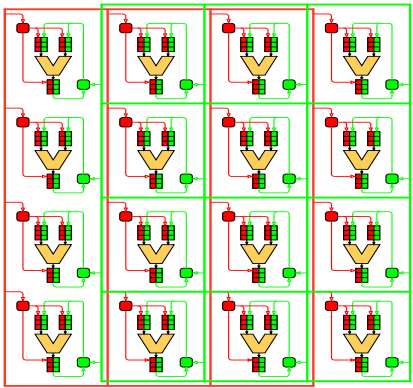
Synchronous Control Asynchronous Dataflow (**SCAD**)

- Exposed datapath architecture

Queue-based code generation

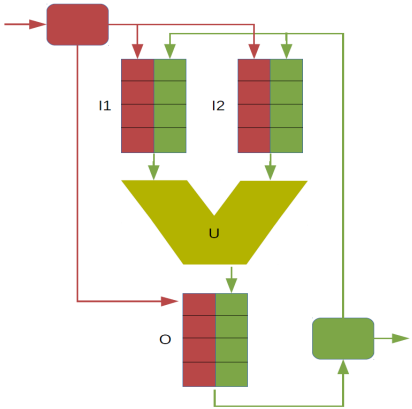
- Eliminate register usage improving effective ILP

SCAD Organization



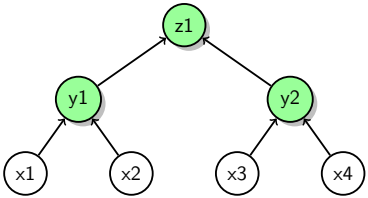
- Grid of processing elements
- FIFO buffers (queues) at inputs and outputs of processing units
- Move instructions $out \rightarrow in$
- Move instruction bus (MIB)
- Data transport network (DTN)
- Application-specific
  - Any arbitrary functionality
  - Interconnect choice

SCAD Organization

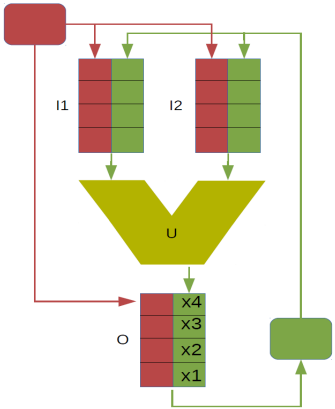


- each slot in queue *(adr,val)*
- Move instruction *out* $\rightarrow$ *in*
- Synchronous control via move instruction bus (MIB)
- Processing unit fires if enough data available
- Asynchronous dataflow via Data transport network (DTN)

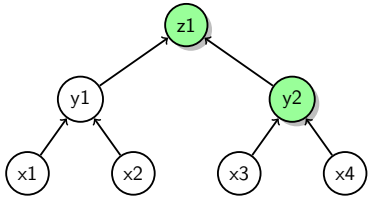
SCAD Functionality



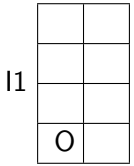
Executed `x1,x2,x3,x4`

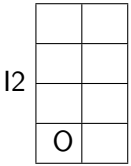


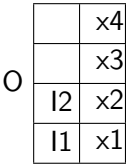
SCAD Functionality



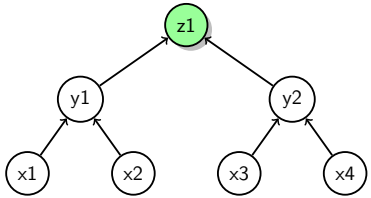
Decoded y1







SCAD Functionality



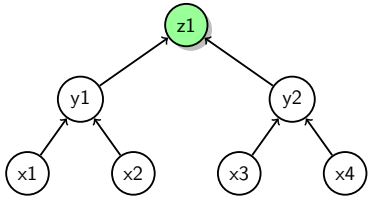
Decoded y2

I1
O
O

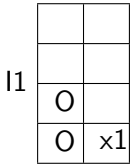
I2
O
O

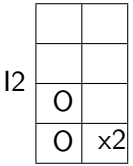
O	
I2	x4
I1	x3
I2	x2
I1	x1

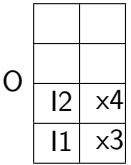
SCAD Functionality



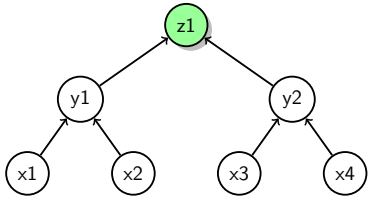
Data transported to execute y1



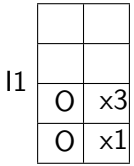


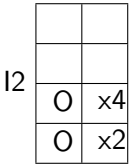


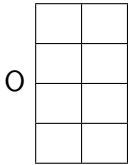
SCAD Functionality



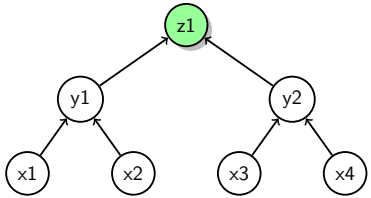
Data transported to execute y2



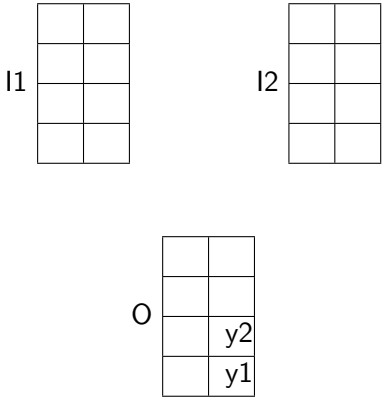




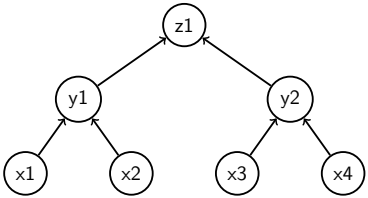
SCAD Functionality



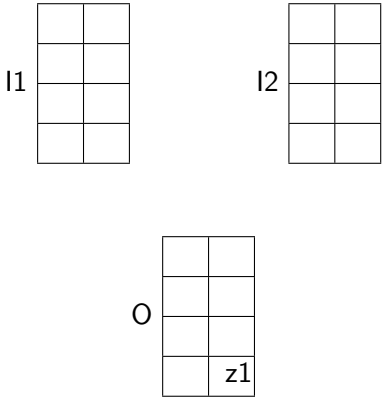
Executed `y1,y2`



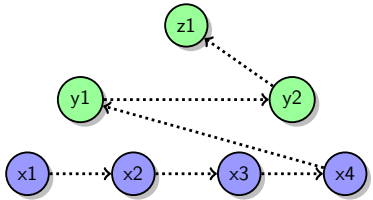
SCAD Functionality



Executed z1



Breadth-First Traversal



No registers used
Queue depth not limited
Queues scale better

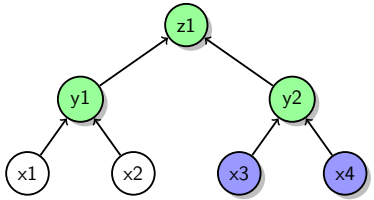
node ordering
x1
x2
x3
x4
y1
y2
z1

Depth-First Traversal

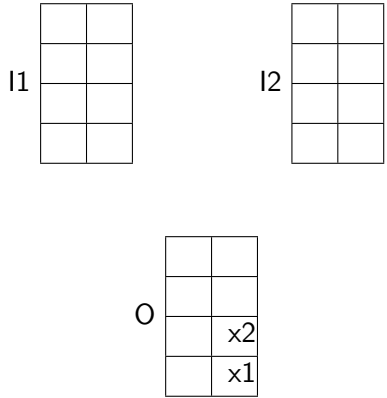
Current Compilers

- Order nodes of 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

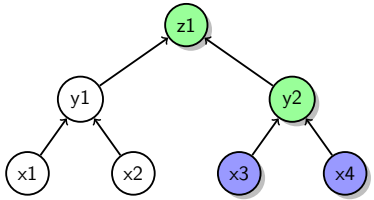
Depth-First Traversal in SCAD



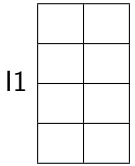
Executed `x1,x2`

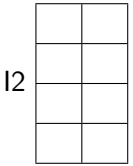


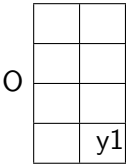
Depth-First Traversal in SCAD



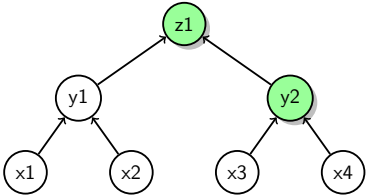
Executed y1





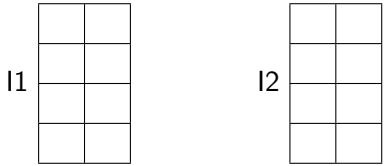


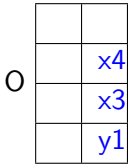
Depth-First Traversal in SCAD



Executed `x3,x4`

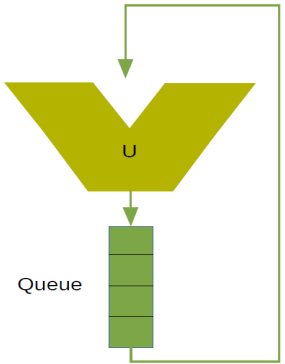
Wrong Order!



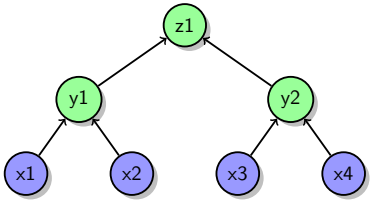


Queue Machine

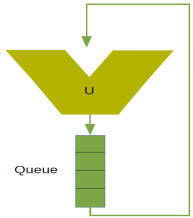
- Similar to the more familiar Stack Machine
- One queue to hold
  - operands for execution
  - result of execution
- Turing complete

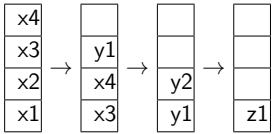


Code Generation for Queue Machine

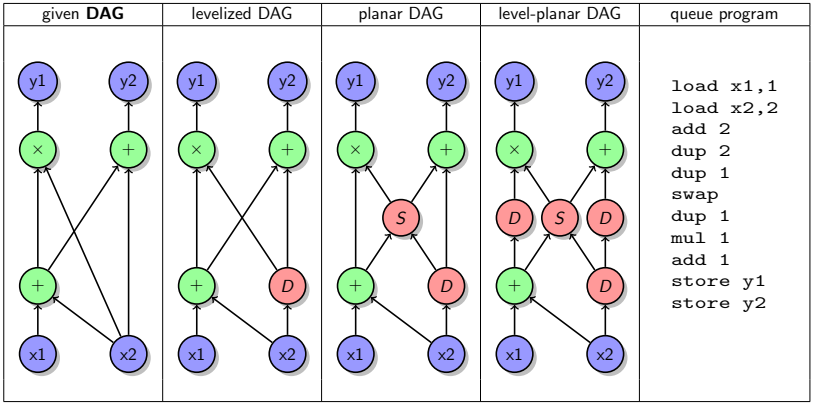


Expression **Tree**





Code Generation for Queue Machine

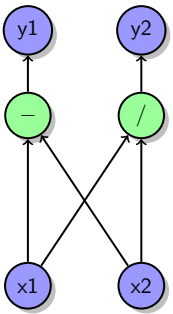
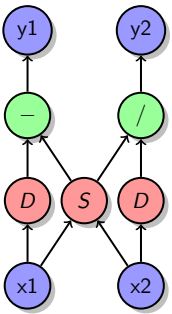


Queue code to SCAD code

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]
⋮	⋮

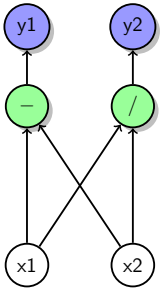
Example DAG on Queue machine

given DAG	level-planar DAG	queue program

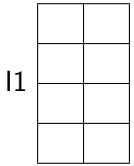
		`load x1,2`
		`load x2,2`
		`dup 1`
		`swap`
		`dup 1`
		`sub 1`
		`div 1`
		`store y1`
		`store y2`

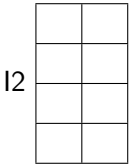
2 *Dup*,1 *Swap*

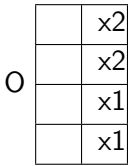
Example DAG on SCAD machine



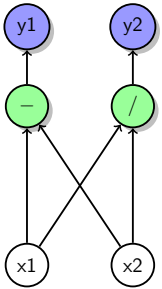
Executed `x1,x2`



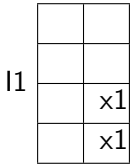


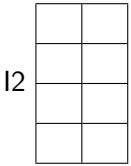


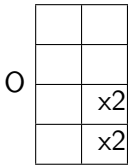
Example DAG on SCAD machine



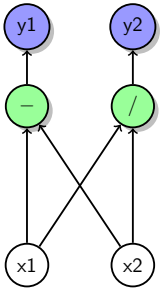
Transported `x1,x1`



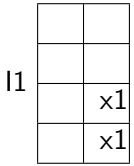


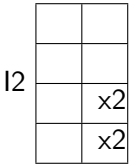


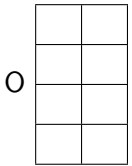
Example DAG on SCAD machine



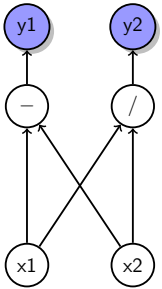
Transported `x2,x2`







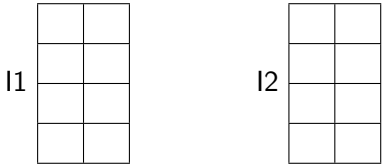
Example DAG on SCAD machine

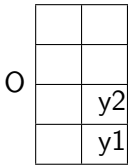


Executed − and /

No Swap and Dup used!

Derived SCAD code is not optimal





Conclusion

Summary

SCAD architecture

- exposed datapath

Queue-based code generation

- eliminates register usage
- improves effective ILP
- but non-optimal

Conclusion

Future Work

- Optimal SCAD code generation for DAGs
- Buffer size analysis
- Performance, cost and timing-predictability comparison

Thank You!

Questions?