Code Generation for **S**ynchronous **C**ontrol **A**synchronous **D**ataflow (**SCAD**) Architectures

Anoop Bhagyanath

Embedded Systems Group
Technische Universität Kaiserslautern

31 January 2020

Outline

Outline

Processor Trend

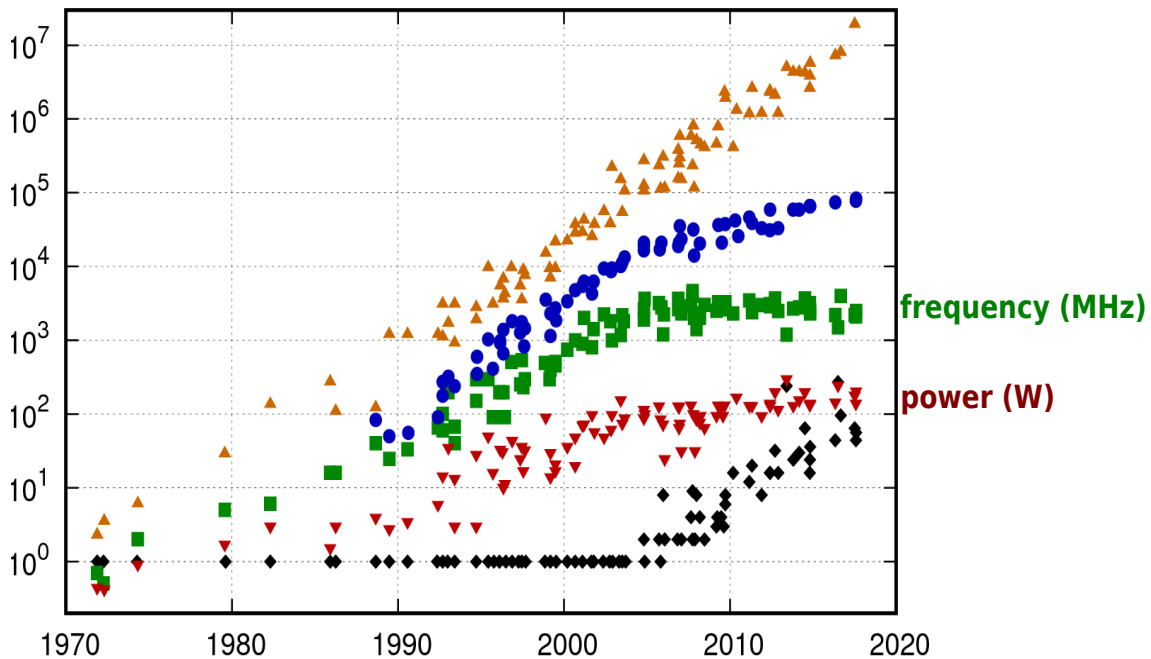
frequency wall (2005 on)

$\rightsquigarrow$ increasing power dissipation

transistor density grows

$\rightsquigarrow$ silicon technology

transistors to performance



Processor Trend

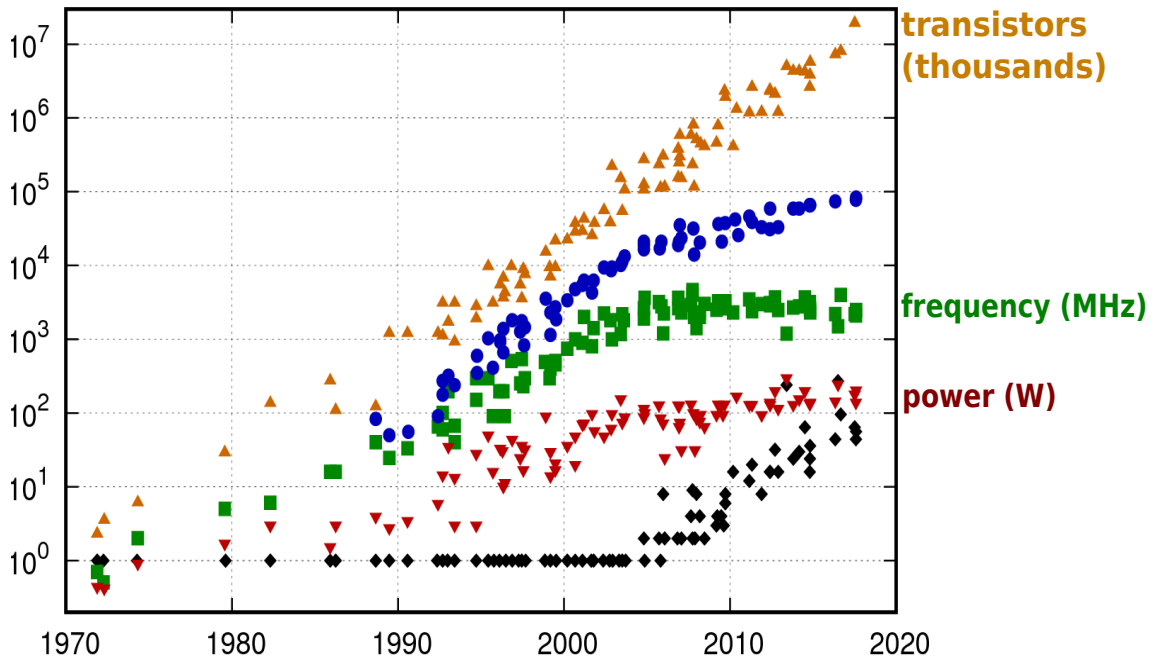
frequency wall (2005 on)

⇝ increasing power dissipation

transistor density grows

⇝ silicon technology

transistors to performance



Processor Trend

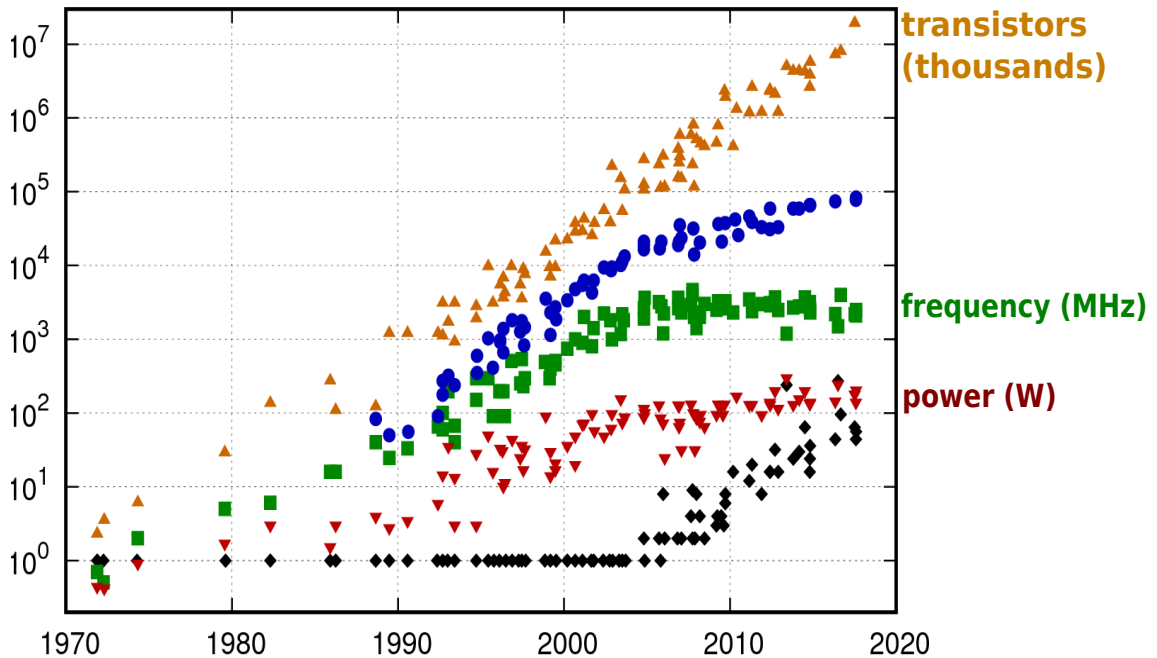
frequency wall (2005 on)

⇝ increasing power dissipation

transistor density grows

⇝ silicon technology

transistors to performance



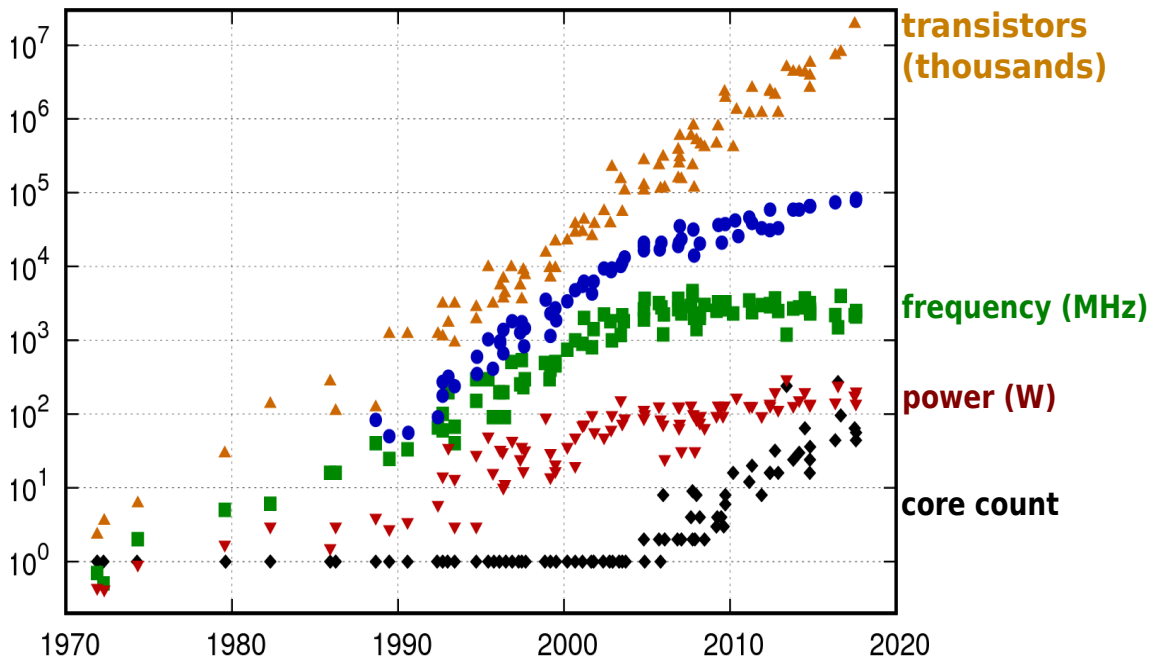
Parallel Computing

thread-level parallelism

- ⇝ programming difficulty
- ⇝ timing analysis difficulty in embedded systems

instr-level parallelism (ILP)

- ⇝ various techniques such as OOO execution, loop unrolling etc
- ⇝ **still limited utilization**



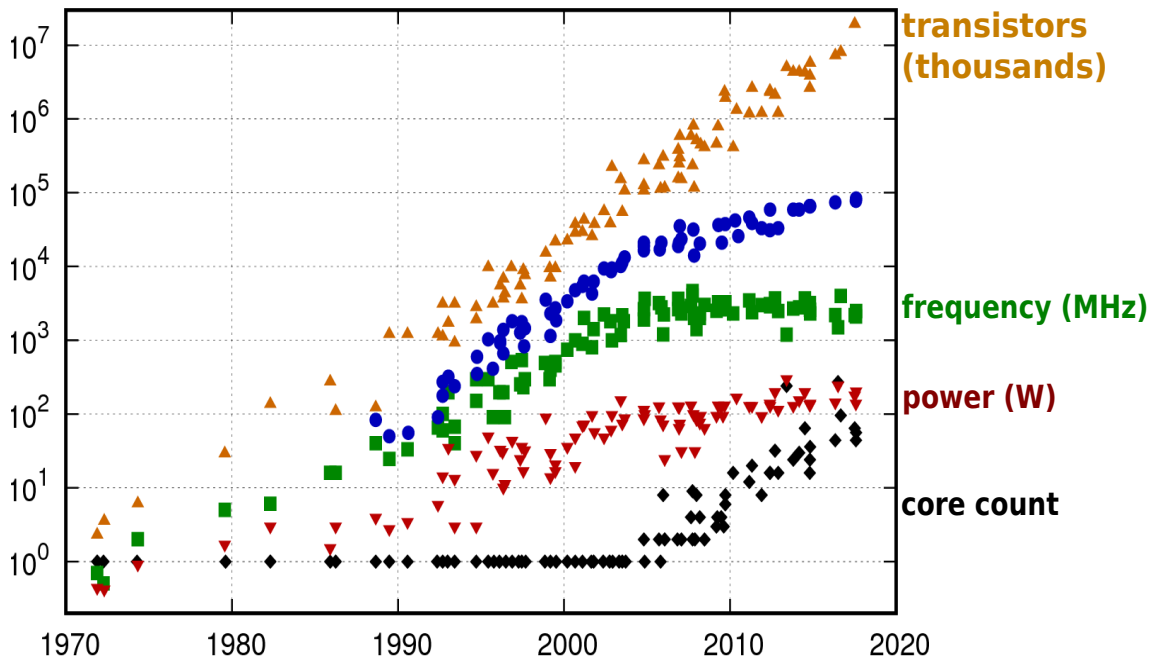
Parallel Computing

thread-level parallelism

- ⇝ programming difficulty
- ⇝ timing analysis difficulty in embedded systems

instr-level parallelism (ILP)

- ⇝ various techniques such as OOO execution, loop unrolling etc
- ⇝ still limited utilization



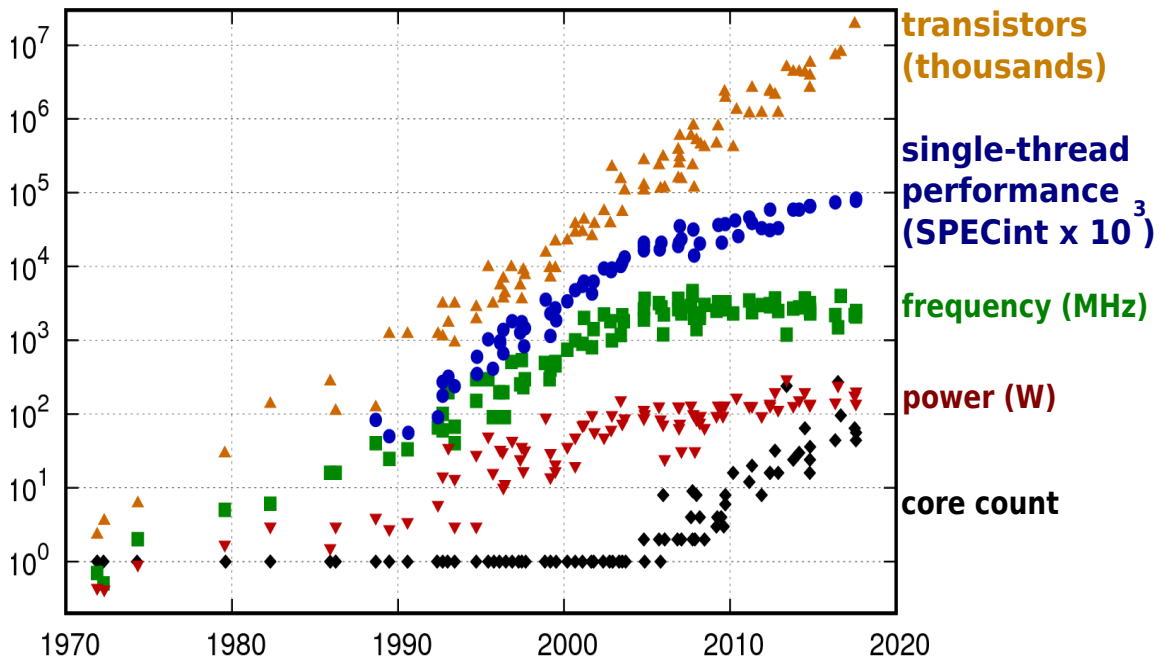
Parallel Computing

thread-level parallelism

- ⇝ programming difficulty
- ⇝ timing analysis difficulty in embedded systems

instr-level parallelism (ILP)

- ⇝ various techniques such as OOO execution, loop unrolling etc
- ⇝ **still limited utilization**



Commercial Architectures

memory wall

- $\rightsquigarrow$ memory hierarchy $\rightsquigarrow$ registers closest to processor
- $\rightsquigarrow$ register-register architectures

ILP bottleneck $\rightsquigarrow$ irrespective of number of processing units (PU)

register file

- $\rightsquigarrow$ limited registers $\rightsquigarrow$ scale with $O(n^2)$
- $\rightsquigarrow$ limited register file ports

  area and power scale with k^3, access time scale with $k^{3/2}$

Commercial Architectures

memory wall

- $\rightsquigarrow$ memory hierarchy $\rightsquigarrow$ registers closest to processor
- $\rightsquigarrow$ register-register architectures

ILP bottleneck $\rightsquigarrow$ irrespective of number of processing units (PU)

register file

- $\rightsquigarrow$ limited registers $\rightsquigarrow$ scale with $O(n^2)$
- $\rightsquigarrow$ limited register file ports
  - area and power scale with k^3, access time scale with $k^{3/2}$

Commercial Architectures

memory wall

- $\rightsquigarrow$ memory hierarchy $\rightsquigarrow$ registers closest to processor
- $\rightsquigarrow$ register-register architectures

ILP bottleneck $\rightsquigarrow$ irrespective of number of processing units (PU)

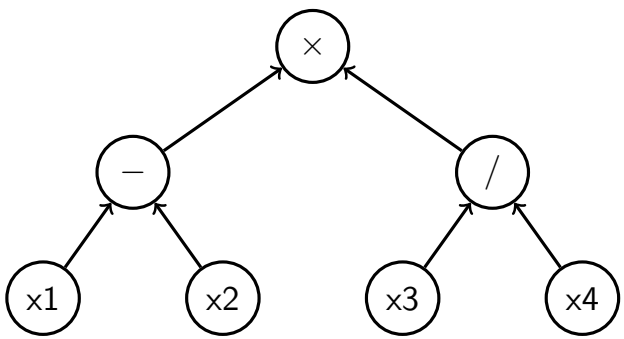
register file

- $\rightsquigarrow$ limited registers $\rightsquigarrow$ scale with $O(n^2)$
- $\rightsquigarrow$ limited register file ports

  area and power scale with k^3, access time scale with $k^{3/2}$

Limited ILP Utilization: An Example

expression tree



3 steps

2W port reg file

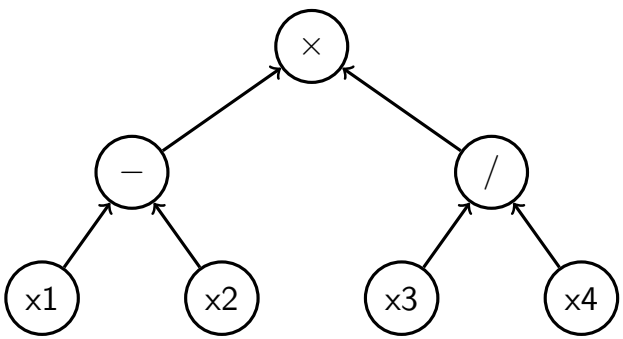
4 steps

2 regs

5 steps

Limited ILP Utilization: An Example

expression tree



3 steps

2W port reg file

x1	x2
x3	x4
−	/
×	

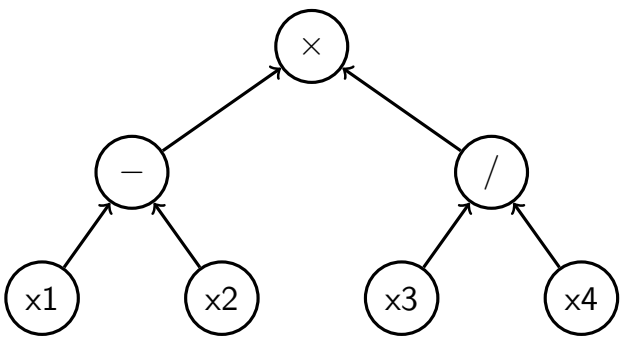
4 steps

2 regs

5 steps

Limited ILP Utilization: An Example

expression tree



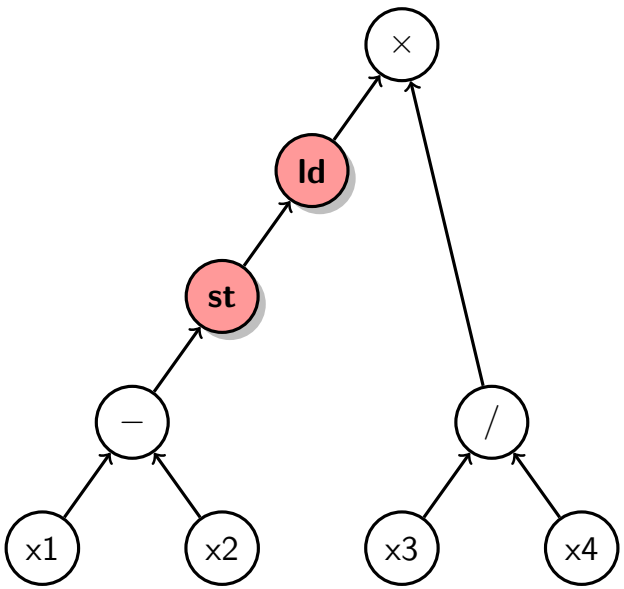
3 steps

2W port reg file

x1	x2
x3	x4
−	/
×	

4 steps

2 regs



5 steps

State of the Art

communicate values directly from producer PU to consumer PU

$\rightsquigarrow$ **direct instruction communication**

exposed datapath architectures

$\rightsquigarrow$ compiler move values directly between PUs

TTA $\rightsquigarrow$ registers at PU output and inputs

$\rightsquigarrow$ still require central register file

TRIPS $\rightsquigarrow$ instruction buffers at PU inputs

$\rightsquigarrow$ block oriented

$\rightsquigarrow$ register file for communicating across blocks

RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc

$\rightsquigarrow$ **still use registers due to**

$\rightsquigarrow$ **execution paradigm and/or code generator drawback**

State of the Art

communicate values directly from producer PU to consumer PU

⇝ **direct instruction communication**

exposed datapath architectures

⇝ compiler move values directly between PUs

TTA ⇝ registers at PU output and inputs

⇝ still require central register file

TRIPS ⇝ instruction buffers at PU inputs

⇝ block oriented

⇝ register file for communicating across blocks

RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc

⇝ **still use registers due to**

⇝ **execution paradigm and/or code generator drawback**

State of the Art

communicate values directly from producer PU to consumer PU

⇝ **direct instruction communication**

exposed datapath architectures

⇝ compiler move values directly between PUs

TTA ⇝ registers at PU output and inputs

⇝ still require central register file

TRIPS ⇝ instruction buffers at PU inputs

⇝ block oriented

⇝ register file for communicating across blocks

RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc

⇝ **still use registers due to**

⇝ **execution paradigm and/or code generator drawback**

State of the Art

communicate values directly from producer PU to consumer PU

- ⇝ **direct instruction communication**

exposed datapath architectures

- ⇝ compiler move values directly between PUs
  - TTA ⇝ registers at PU output and inputs
    - ⇝ still require central register file
  - TRIPS ⇝ instruction buffers at PU inputs
    - ⇝ block oriented
    - ⇝ register file for communicating across blocks
  - RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc
- ⇝ **still use registers due to**
  - ⇝ **execution paradigm and/or code generator drawback**

State of the Art

communicate values directly from producer PU to consumer PU

- ⇝ **direct instruction communication**

exposed datapath architectures

- ⇝ compiler move values directly between PUs
  - TTA ⇝ registers at PU output and inputs
    - ⇝ still require central register file
  - TRIPS ⇝ instruction buffers at PU inputs
    - ⇝ block oriented
    - ⇝ register file for communicating across blocks
  - RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc
- ⇝ **still use registers due to**
  - ⇝ **execution paradigm and/or code generator drawback**

State of the Art

communicate values directly from producer PU to consumer PU

- ⇝ **direct instruction communication**

exposed datapath architectures

- ⇝ compiler move values directly between PUs
  - TTA ⇝ registers at PU output and inputs
    - ⇝ still require central register file
  - TRIPS ⇝ instruction buffers at PU inputs
    - ⇝ block oriented
    - ⇝ register file for communicating across blocks
  - RAW, STA, MOVE-PRO, FlexCore, DySER, Tartan etc
- ⇝ **still use registers due to**
  - ⇝ **execution paradigm and/or code generator drawback**

Outline

SCAD Architecture

FIFO buffers

- $\rightsquigarrow$ scale with $O(n)$

move instructions

- $\rightsquigarrow$ *out buf* $\rightarrow$ *inp buf*
- $\rightsquigarrow$ direct instr communication

buffers connected to control unit

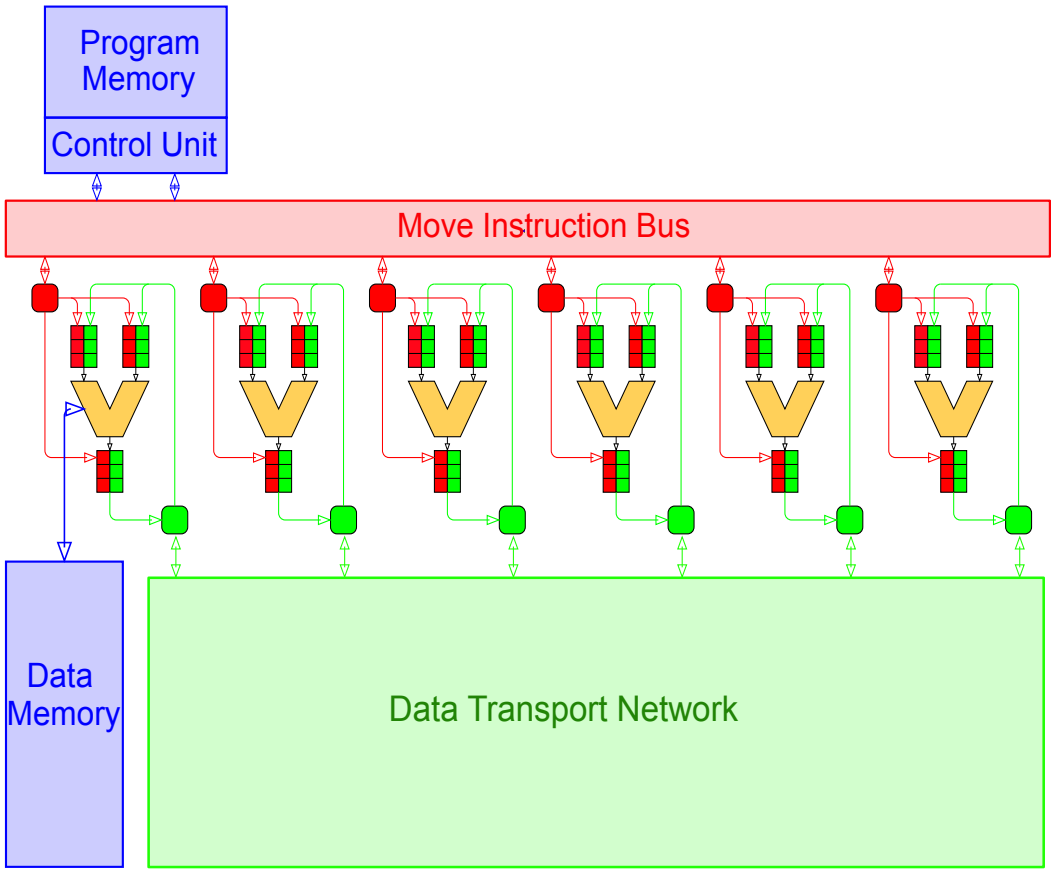
- $\rightsquigarrow$ move instruction bus

outputs connected to inputs

- $\rightsquigarrow$ data transport network

application-specific PUs

- $\rightsquigarrow$ embedded systems



SCAD Architecture

FIFO buffers

- ⇝ scale with $O(n)$

move instructions

- ⇝ *out buf* → *inp buf*
- ⇝ direct instr communication

buffers connected to control unit

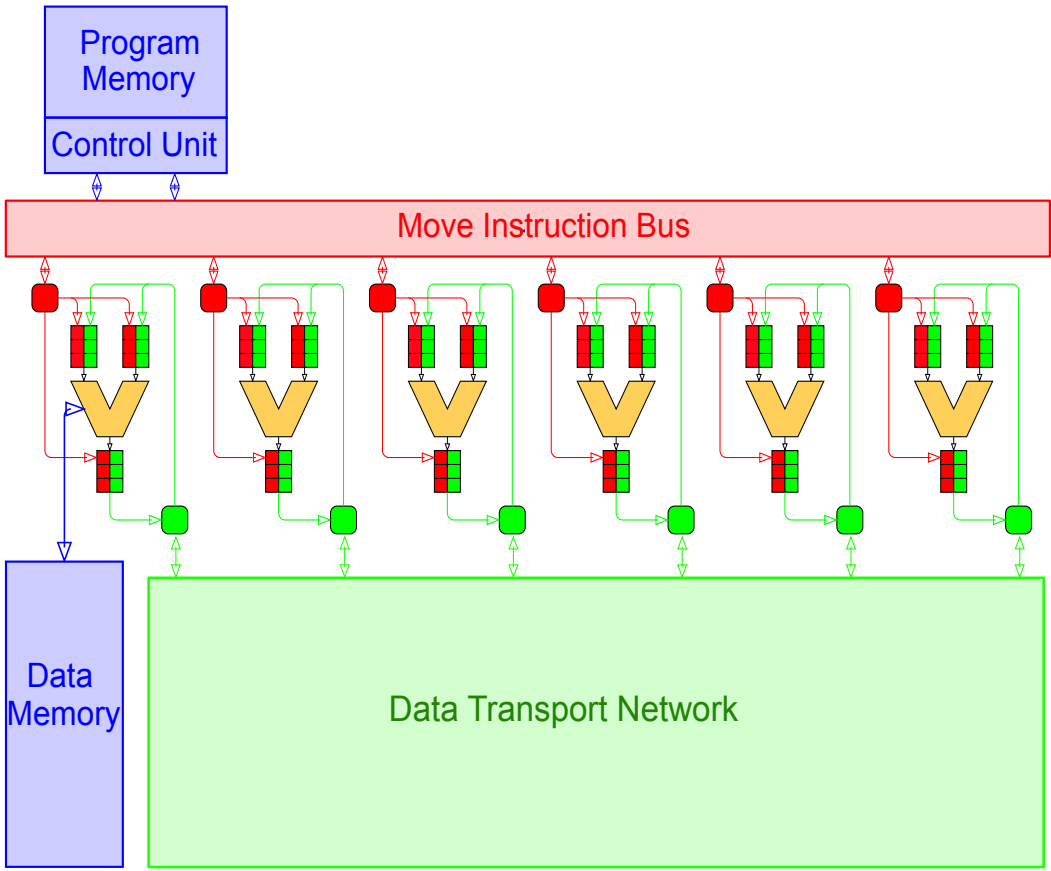
- ⇝ move instruction bus

outputs connected to inputs

- ⇝ data transport network

application-specific PUs

- ⇝ embedded systems



SCAD Architecture

FIFO buffers

- ⇝ scale with $O(n)$

move instructions

- ⇝ *out buf* → *inp buf*
- ⇝ direct instr communication

buffers connected to control unit

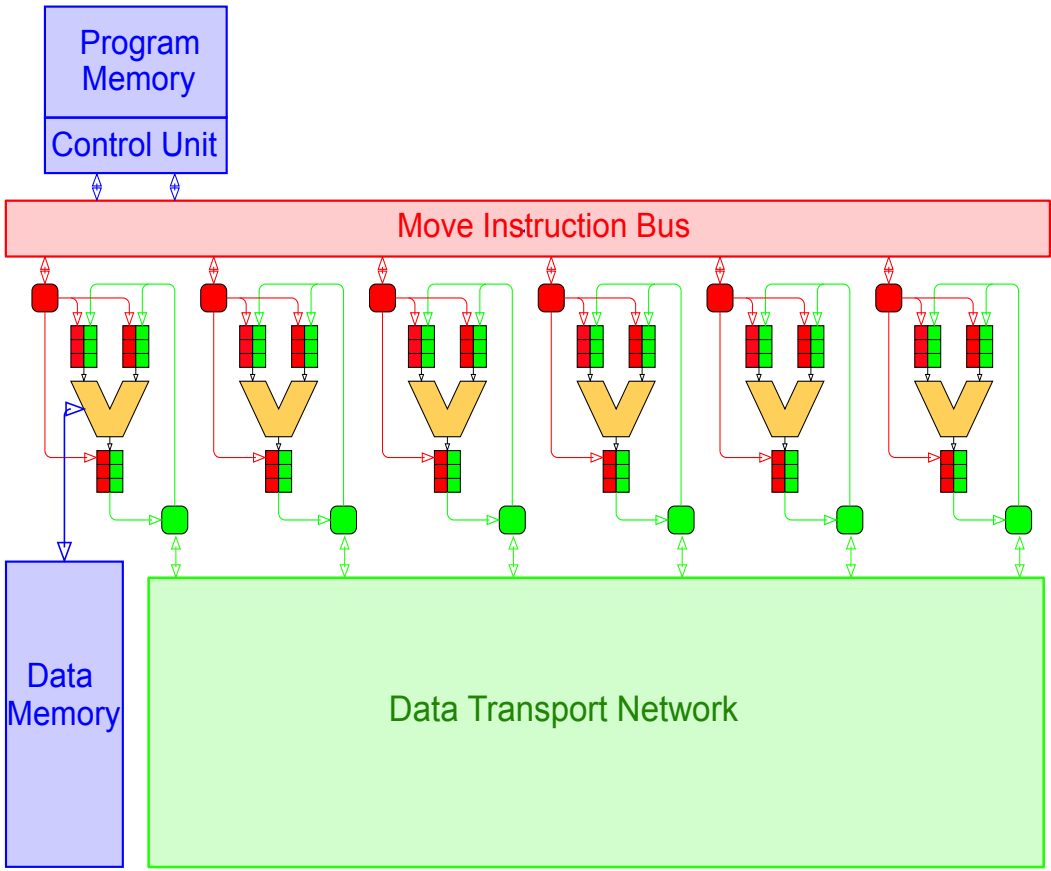
- ⇝ move instruction bus

outputs connected to inputs

- ⇝ data transport network

application-specific PUs

- ⇝ embedded systems



SCAD Architecture

FIFO buffers

- ⇝ scale with $O(n)$

move instructions

- ⇝ *out buf* → *inp buf*
- ⇝ direct instr communication

buffers connected to control unit

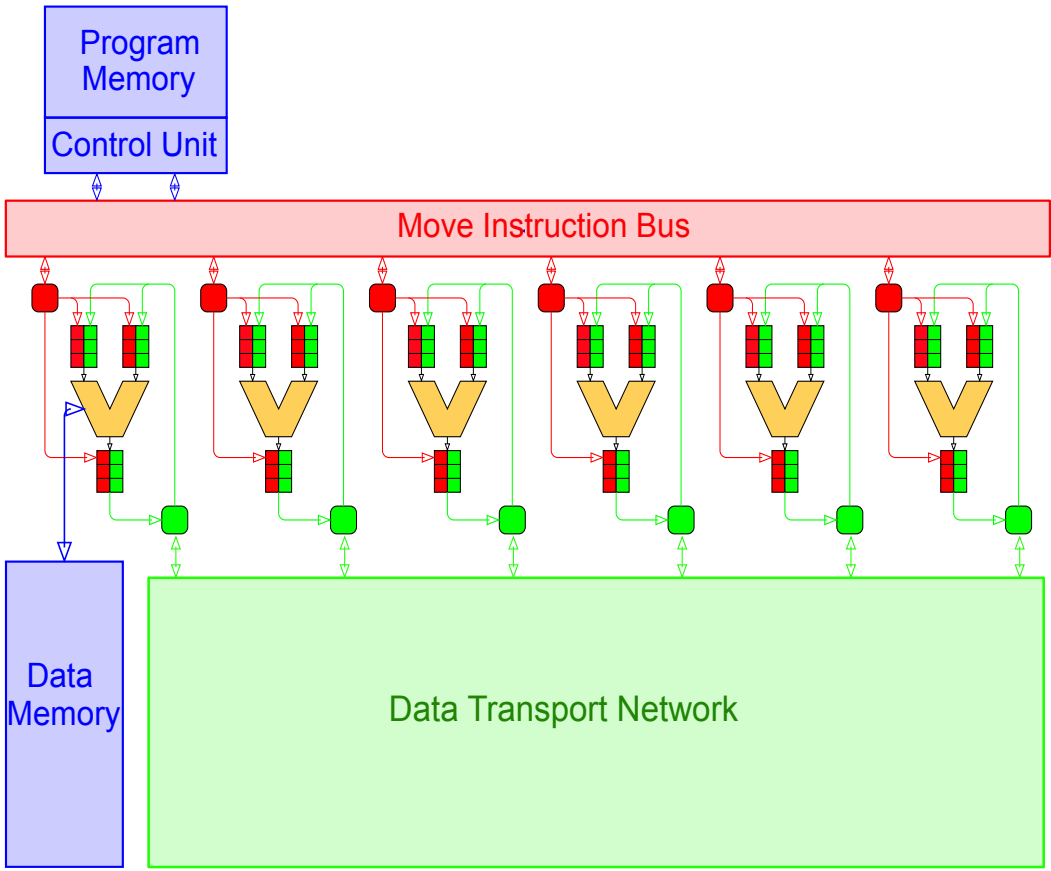
- ⇝ move instruction bus

outputs connected to inputs

- ⇝ data transport network

application-specific PUs

- ⇝ embedded systems



SCAD Architecture

FIFO buffers

- ⇝ scale with $O(n)$

move instructions

- ⇝ *out buf* → *inp buf*
- ⇝ direct instr communication

buffers connected to control unit

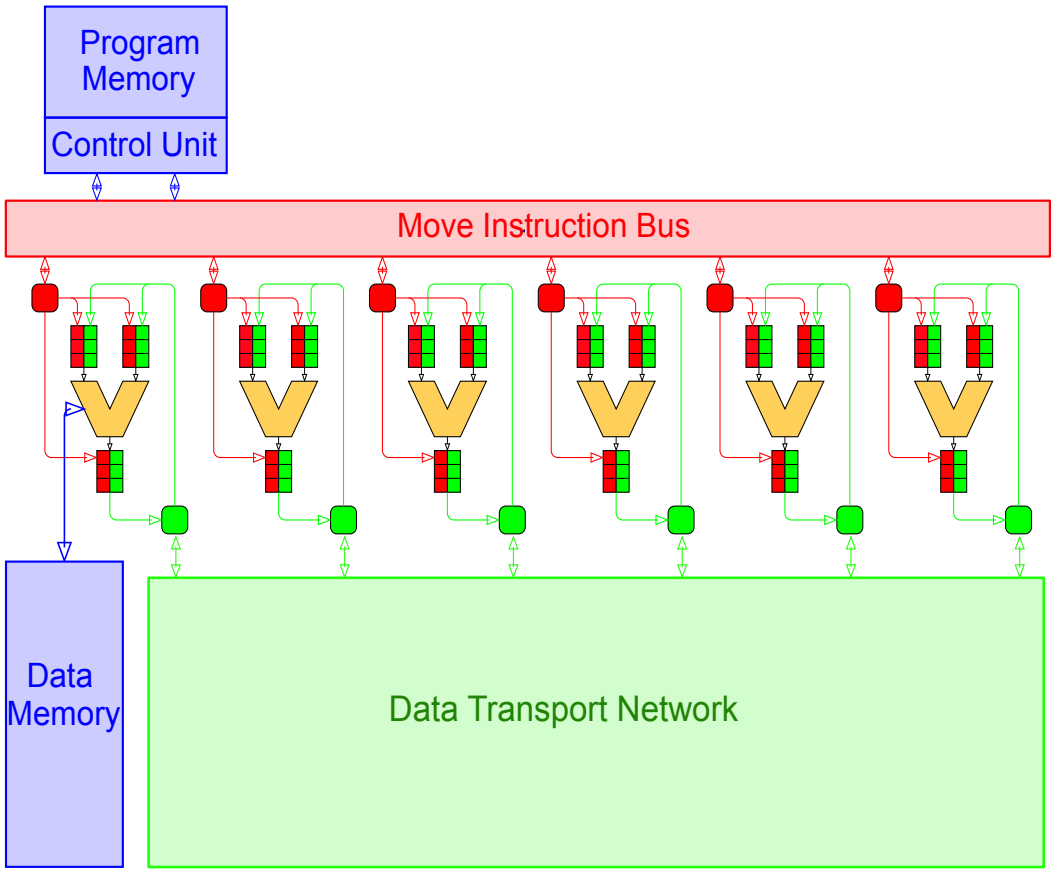
- ⇝ move instruction bus

outputs connected to inputs

- ⇝ data transport network

application-specific PUs

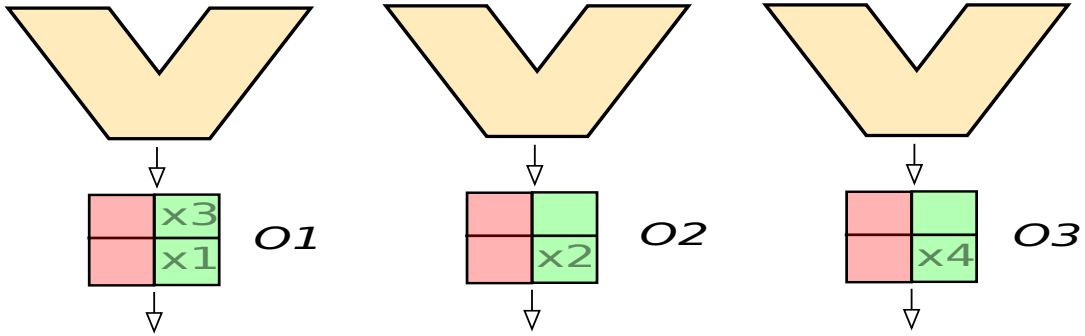
- ⇝ embedded systems



Execution in SCAD

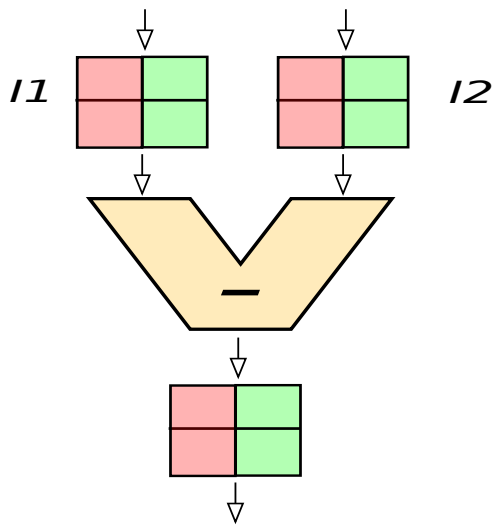
program

$\rightsquigarrow$ $y1 = x1 - x2$
$y2 = x3 - x4$



move program

$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$

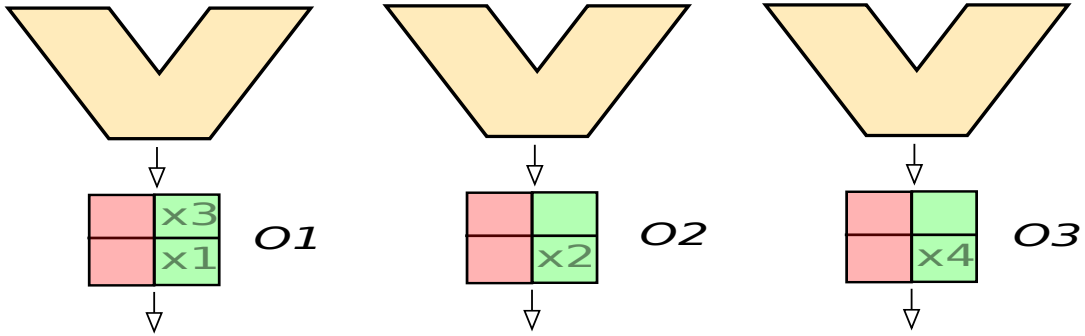


Sync (sequential) Control Word (parallel) Data Flow

Execution in SCAD

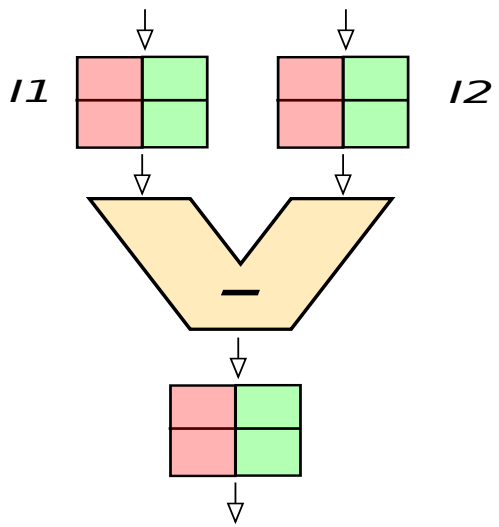
program

$\rightsquigarrow$ $y1 = x1 - x2$
$y2 = x3 - x4$



move program

$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$

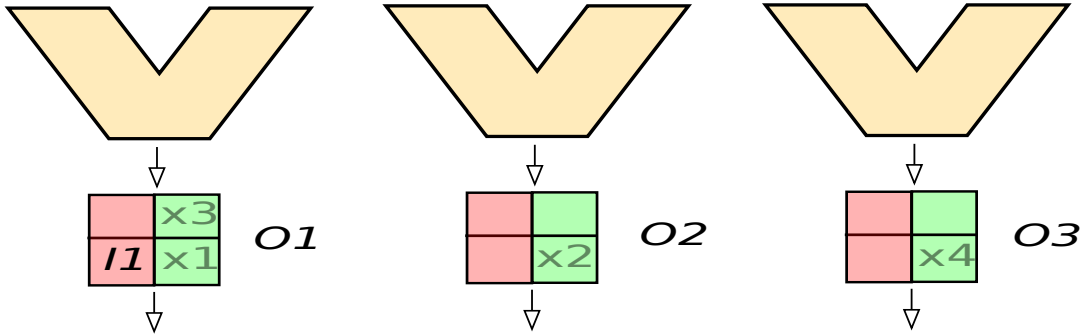


Sync (sequential) Control Word (parallel) Data Flow

Execution in SCAD

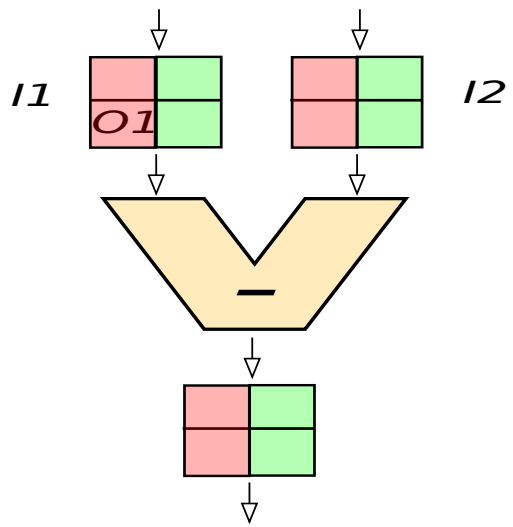
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

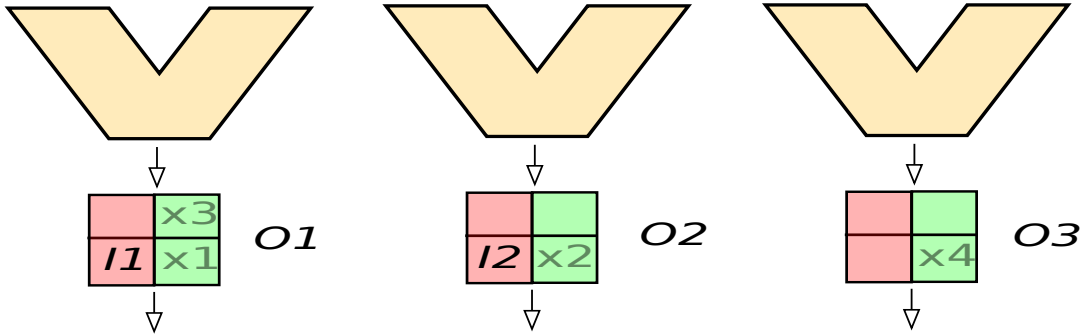
$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$



Execution in SCAD

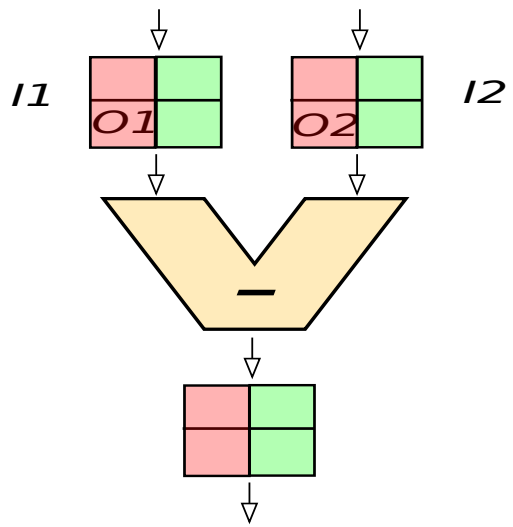
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$

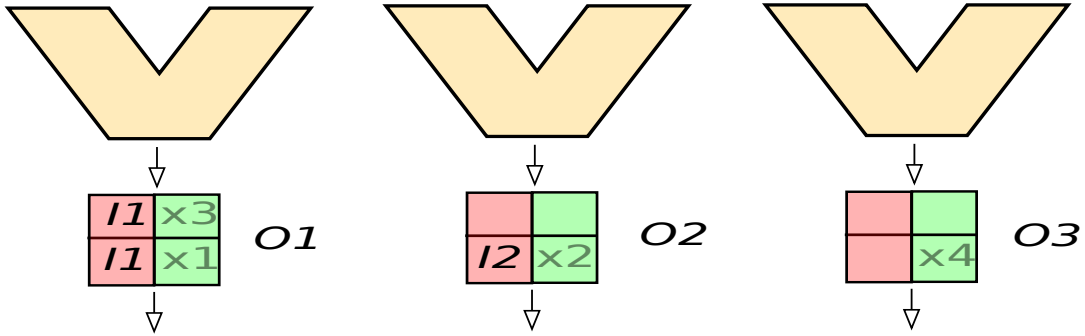


Sync (sequential) Control Work (parallel) Data Flow

Execution in SCAD

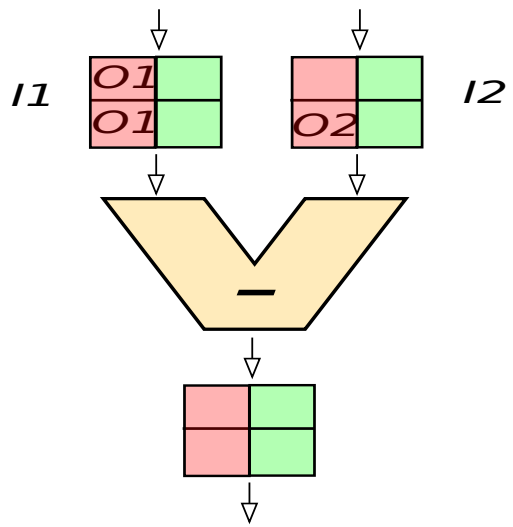
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

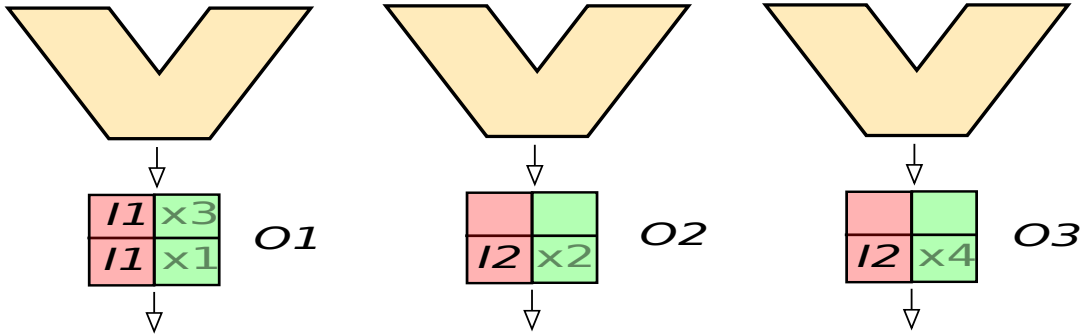
$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$



Execution in SCAD

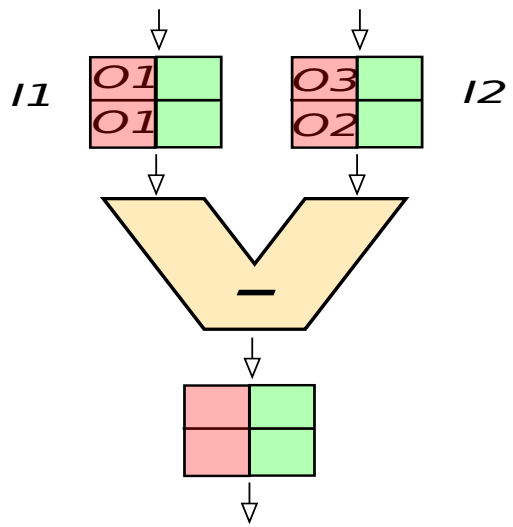
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

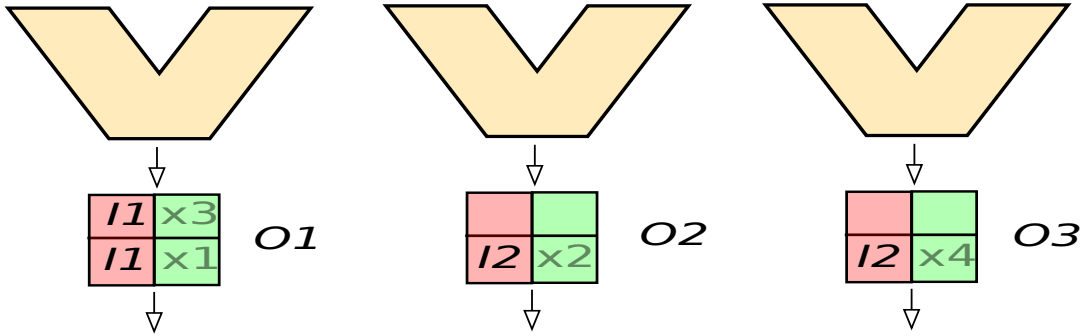
$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$



Execution in SCAD

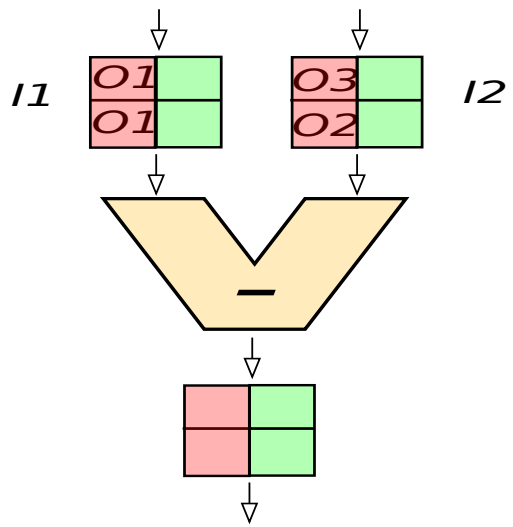
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$

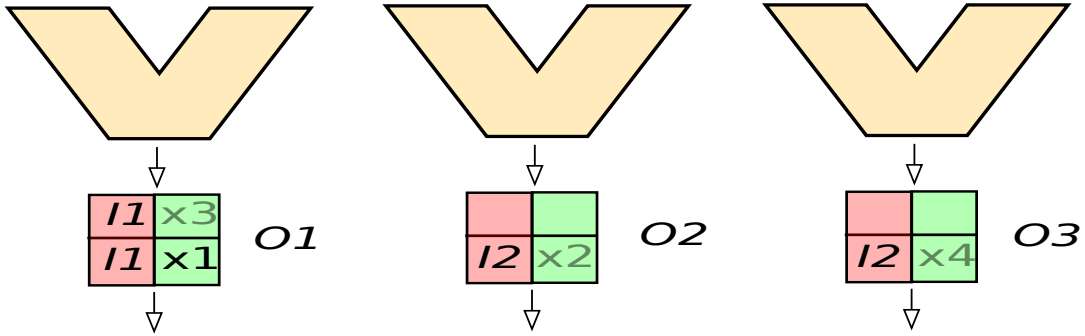


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

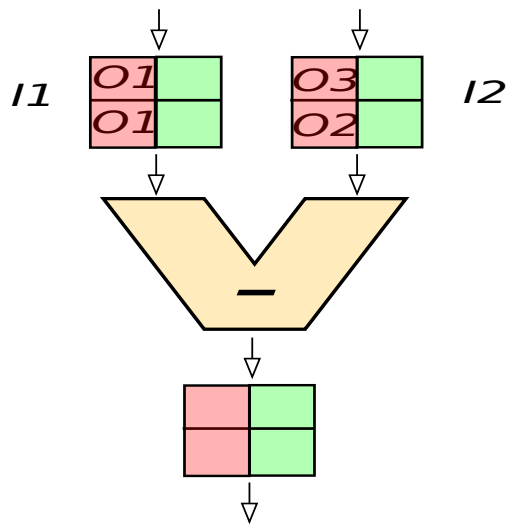
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$
$O2 \rightarrow I2$
$O1 \rightarrow I1$
$O3 \rightarrow I2$



Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

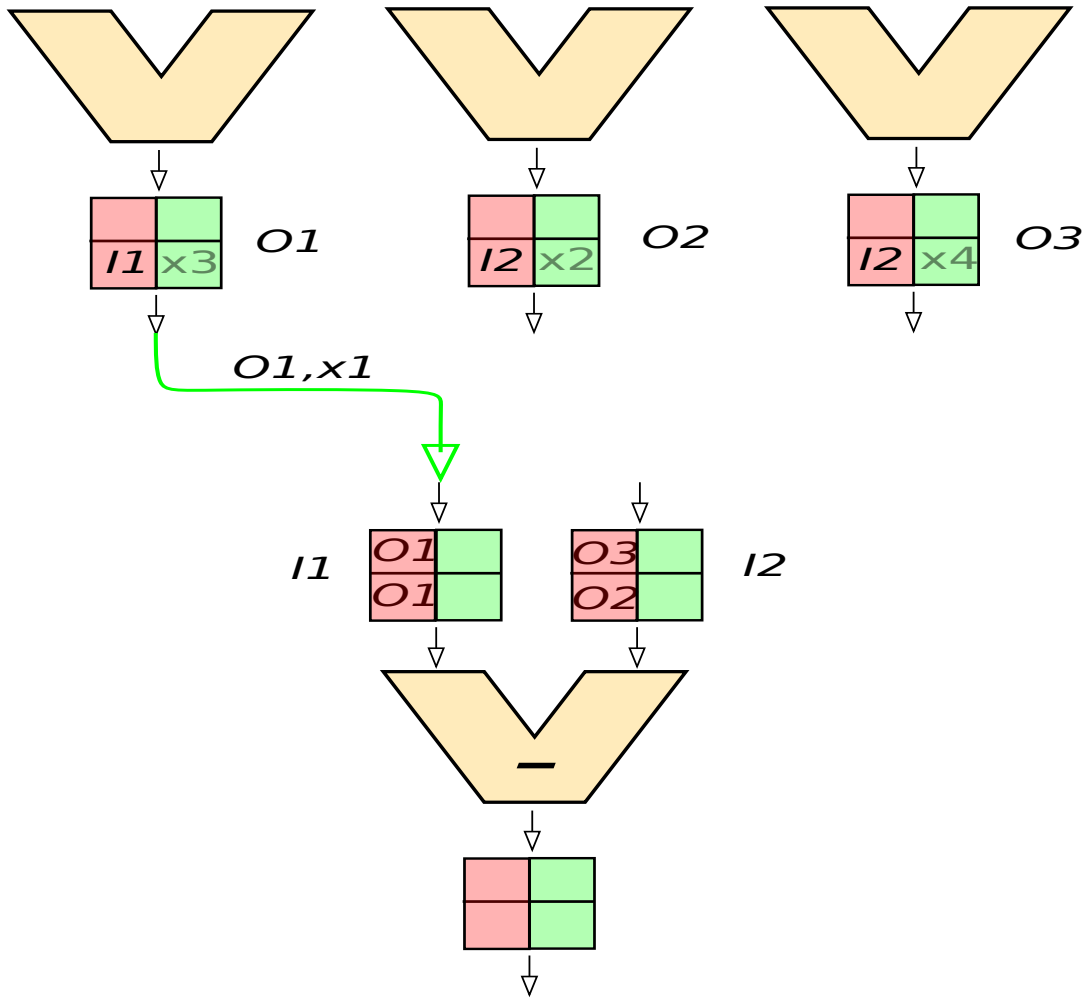
Execution in SCAD

program

$$\rightsquigarrow \begin{aligned} y1 &= x1 - x2 \\ y2 &= x3 - x4 \end{aligned}$$

move program

$$\rightsquigarrow \begin{aligned} O1 &\rightarrow I1 \\ O2 &\rightarrow I2 \\ O1 &\rightarrow I1 \\ O3 &\rightarrow I2 \end{aligned}$$

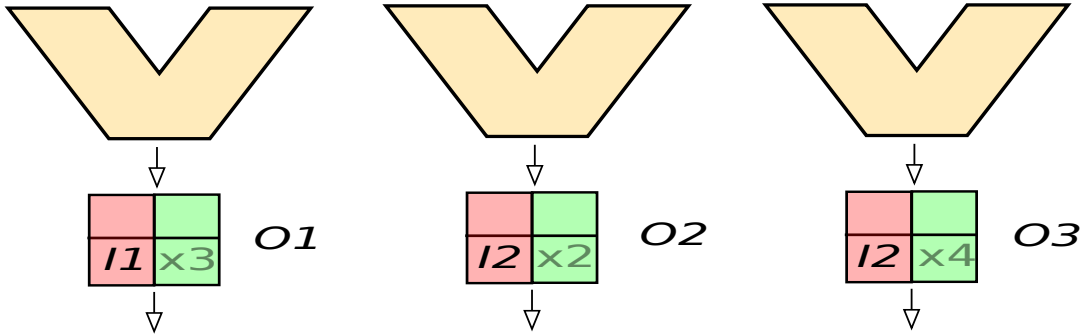


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

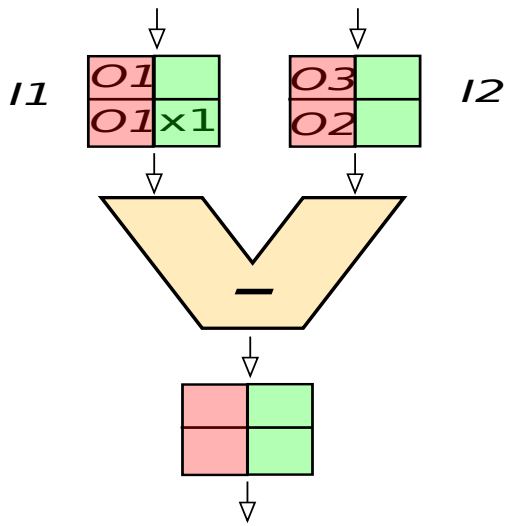
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2*

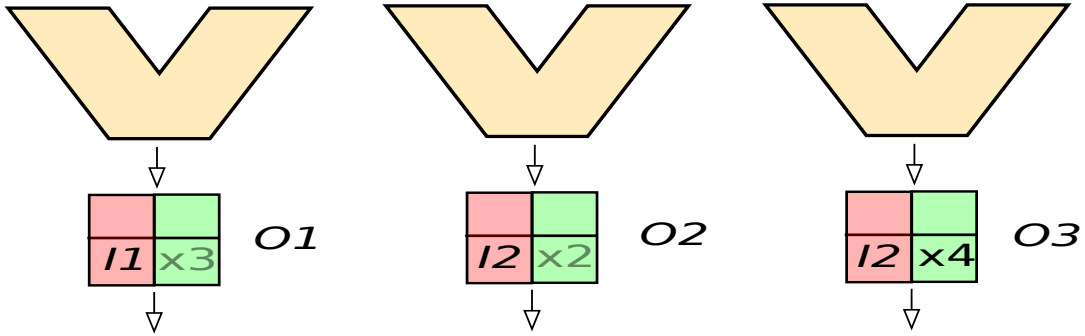


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

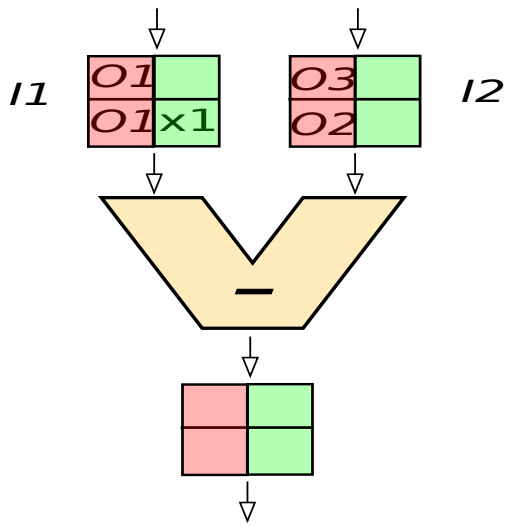
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2*



Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

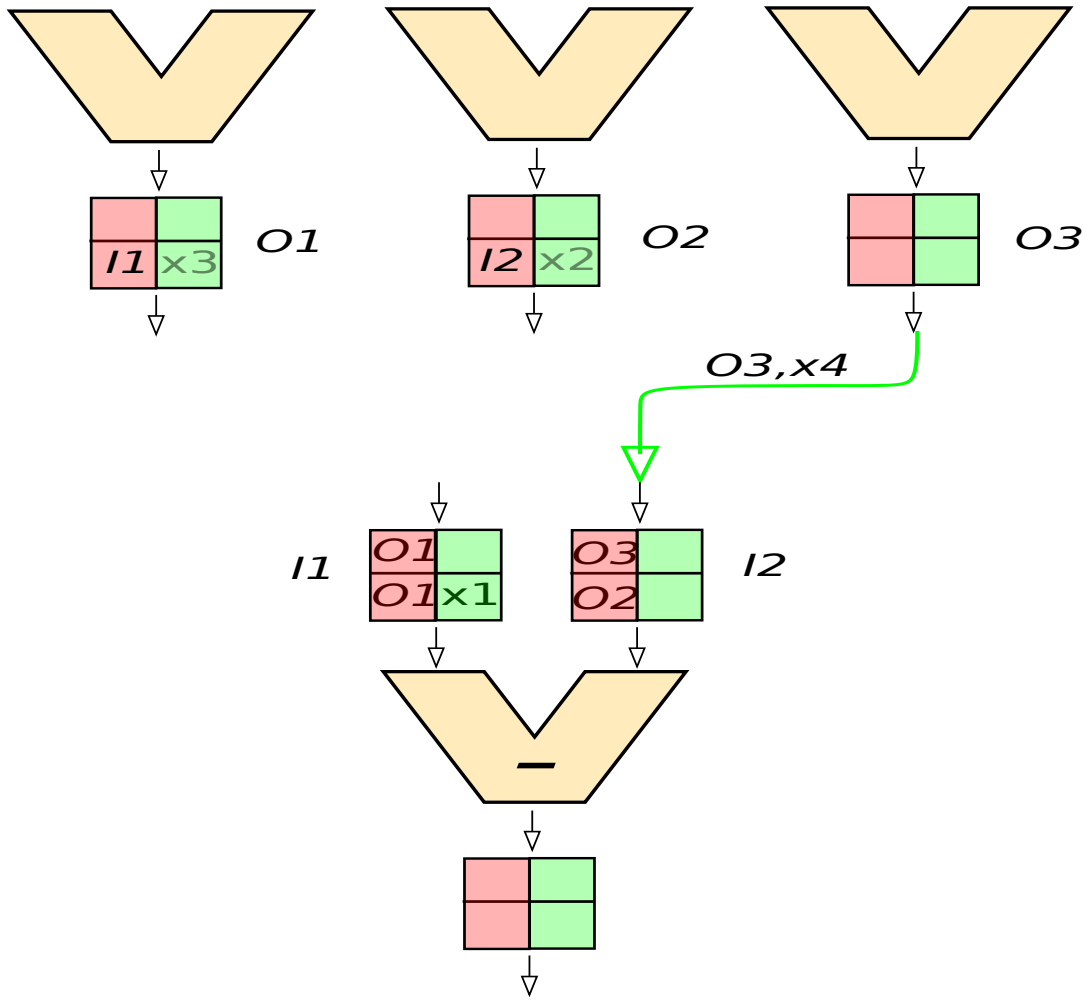
Execution in SCAD

program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4

move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2*

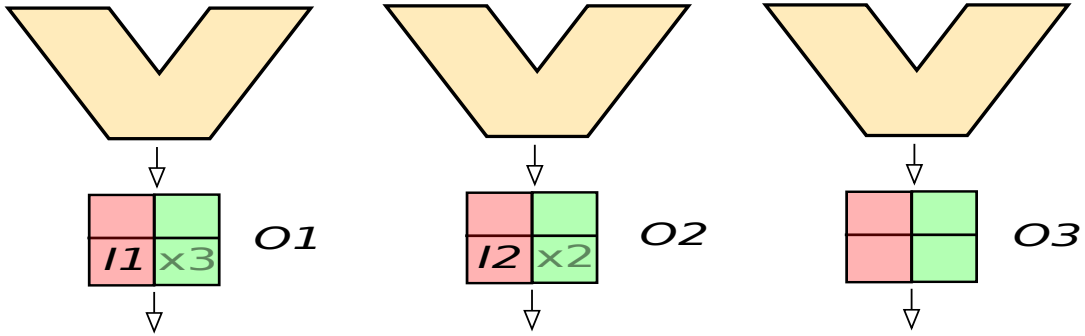


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

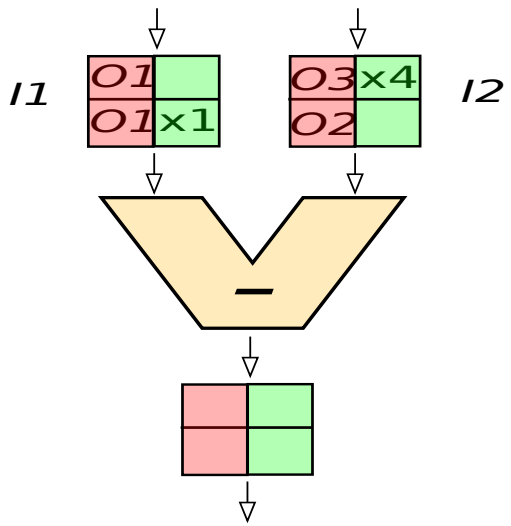
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2* ✓

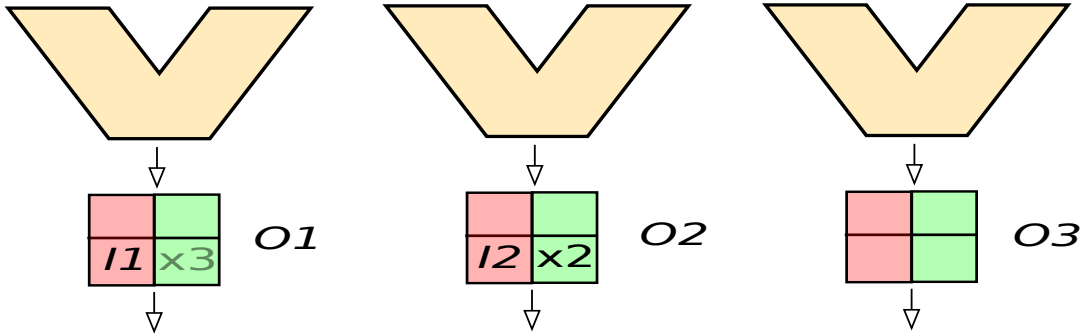


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

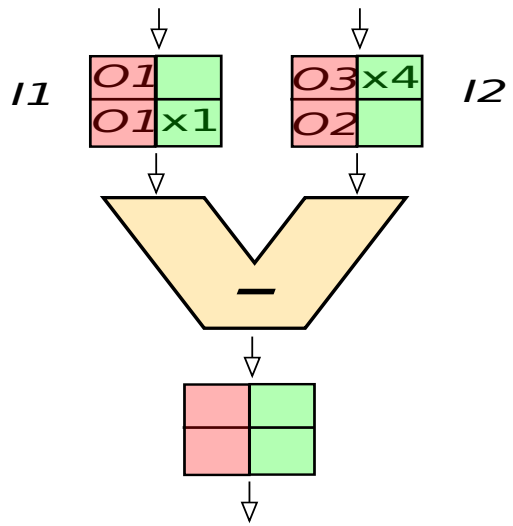
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2* ✓



Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

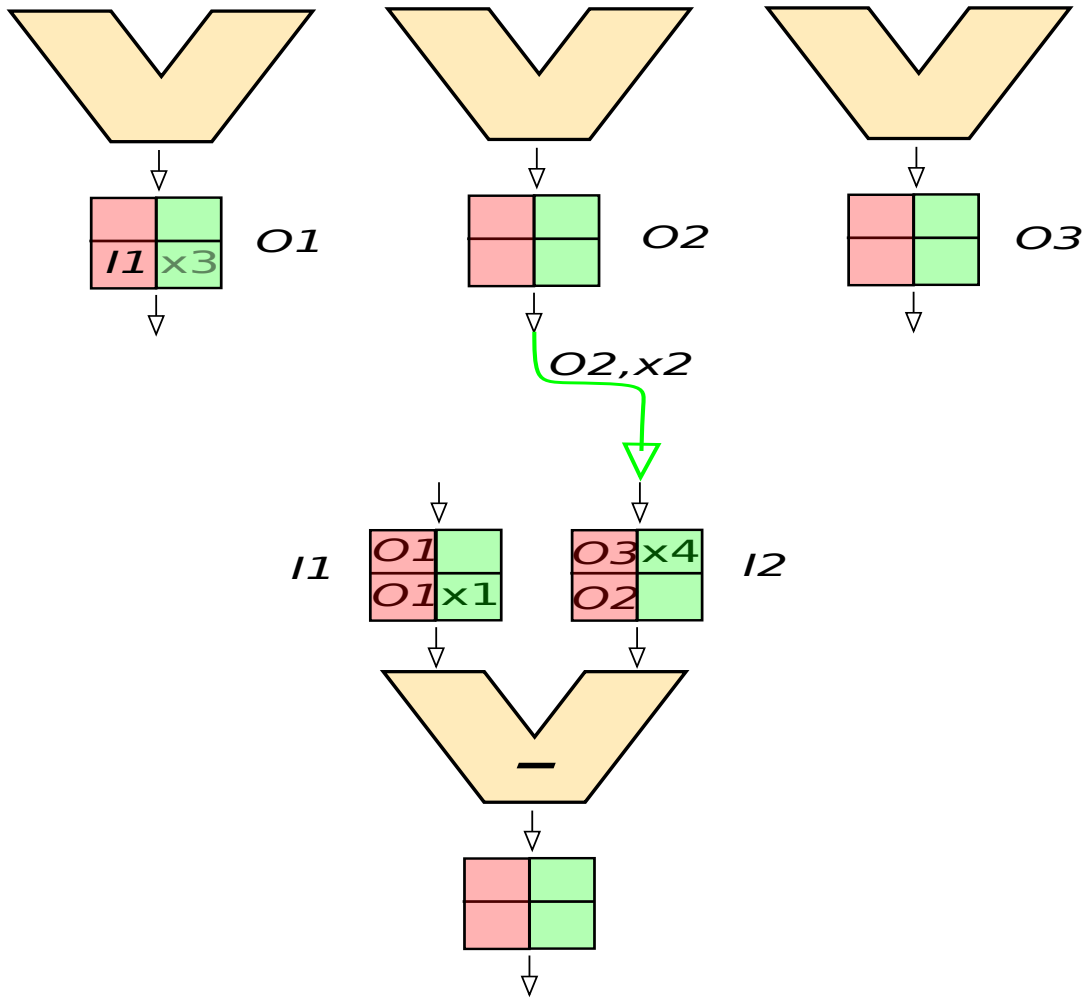
Execution in SCAD

program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4

move program

$\rightsquigarrow$ *O1* → *I1* ✓
O2 → *I2*
O1 → *I1*
O3 → *I2* ✓

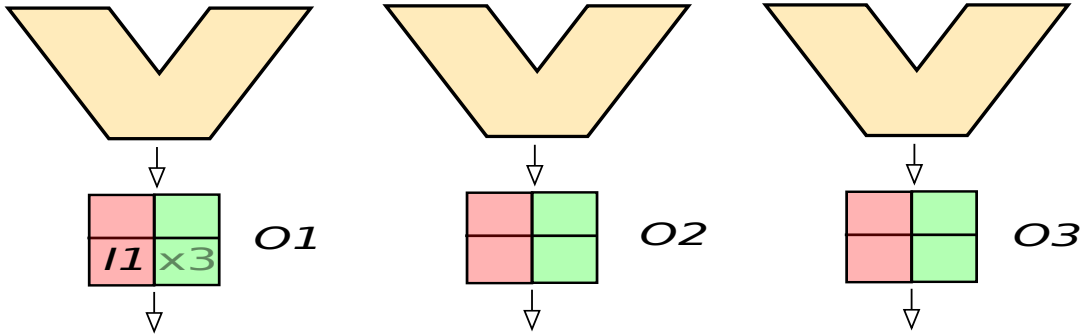


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

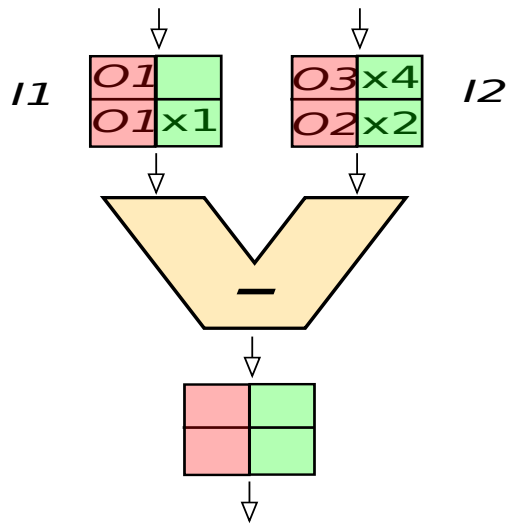
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$
$O3 \rightarrow I2$ ✓

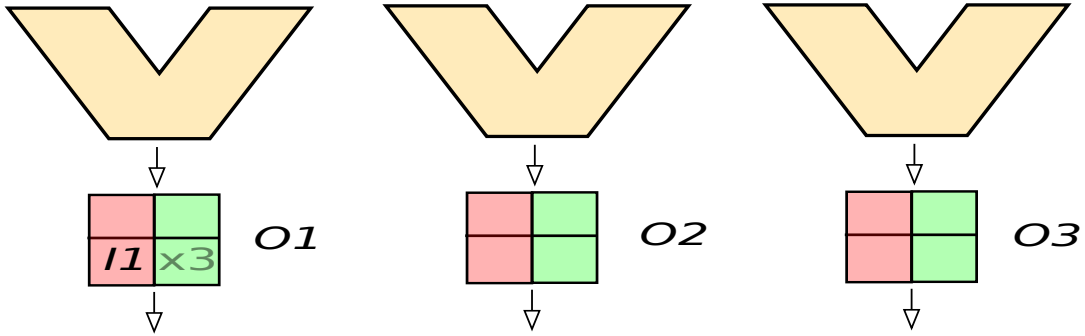


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

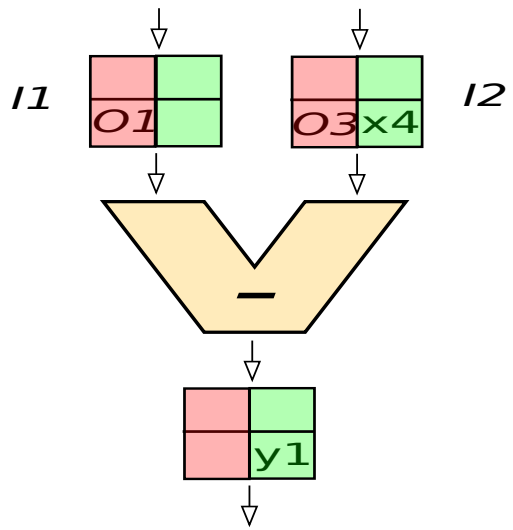
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$
$O3 \rightarrow I2$ ✓

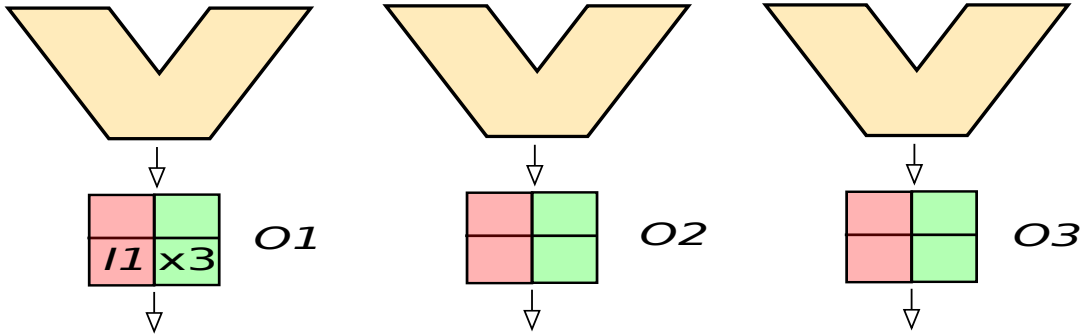


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

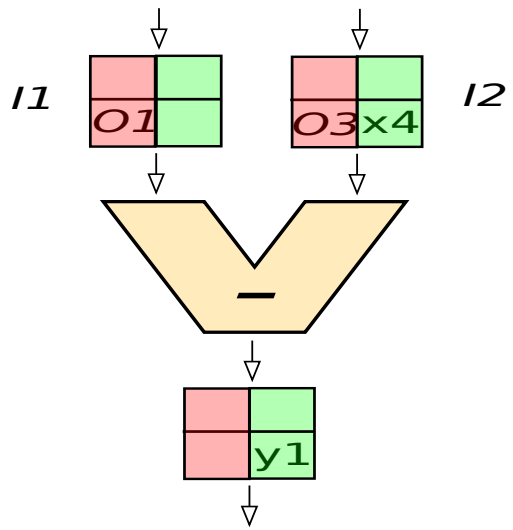
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$
$O3 \rightarrow I2$ ✓



Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

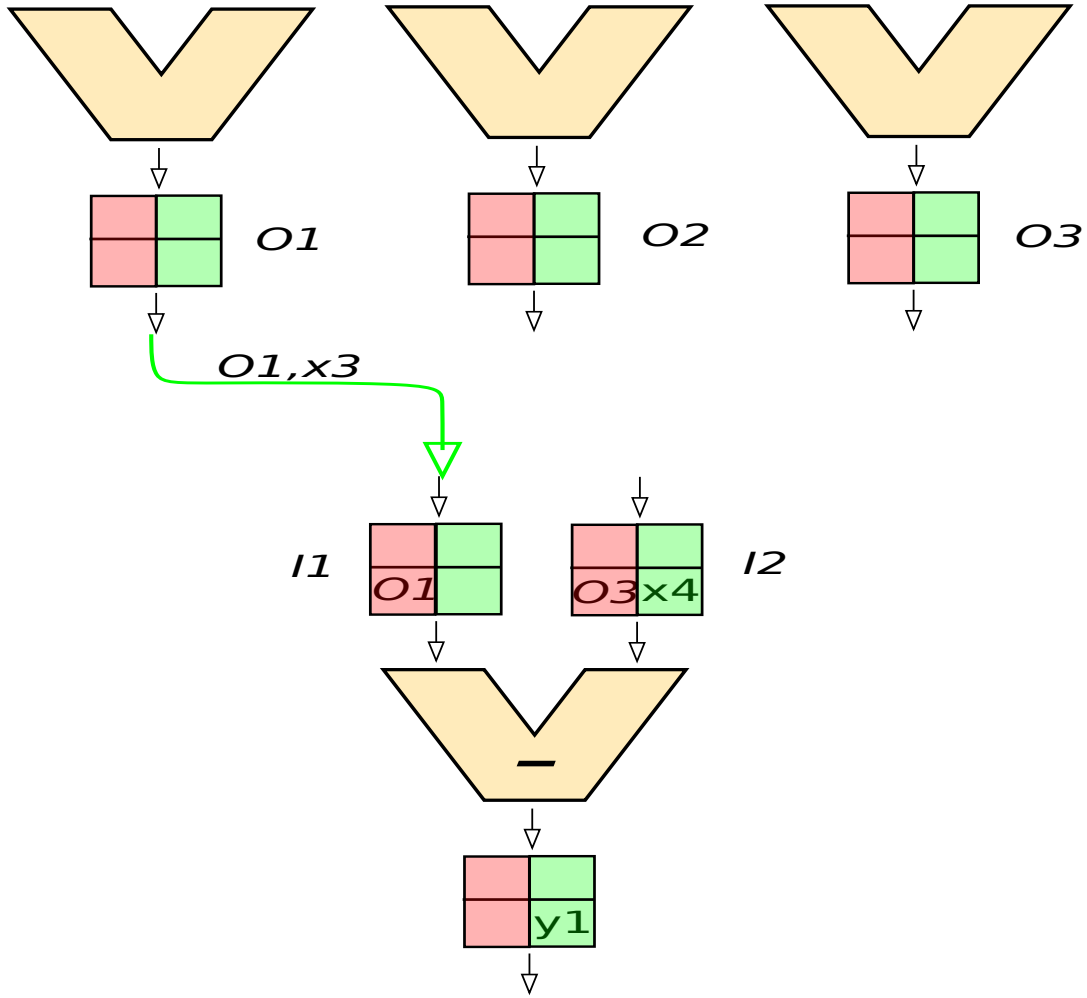
Execution in SCAD

program

$\rightsquigarrow$ $y1 = x1 - x2$
$y2 = x3 - x4$

move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$
$O3 \rightarrow I2$ ✓

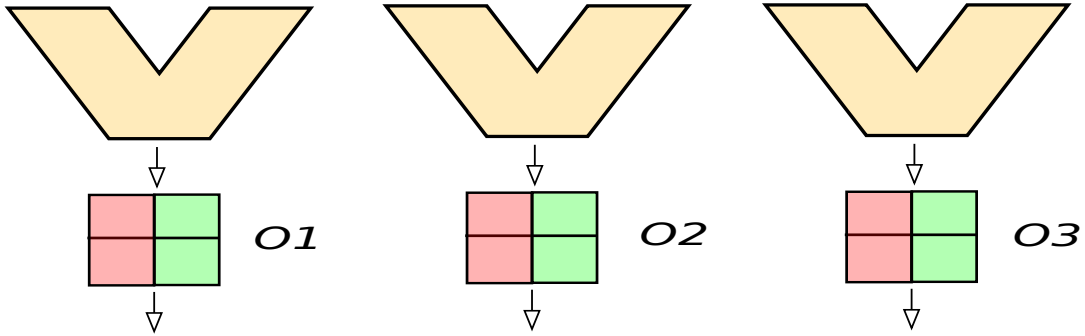


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

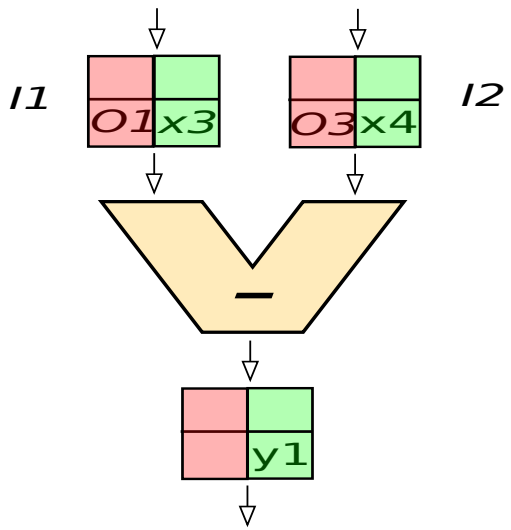
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$ ✓
$O3 \rightarrow I2$ ✓

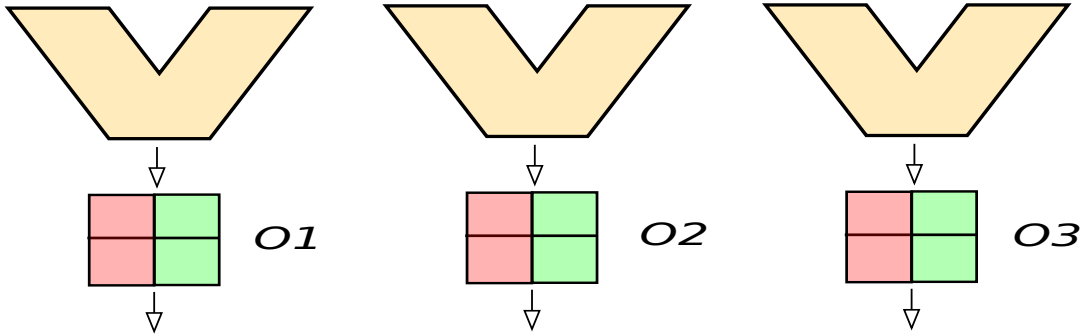


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

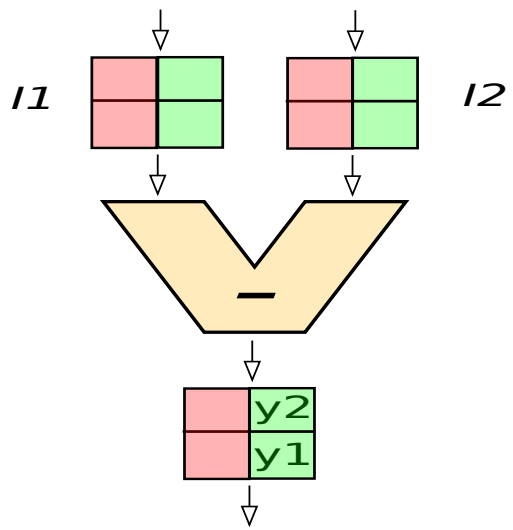
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$ ✓
$O3 \rightarrow I2$ ✓

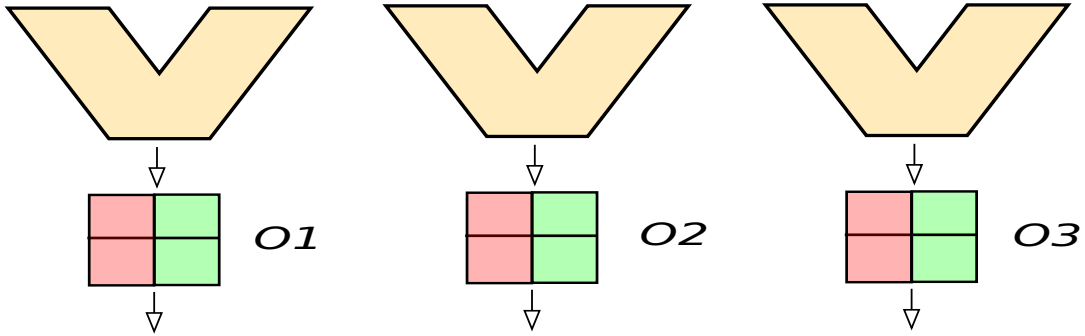


Sync (sequential) **C**ontrol Async (parallel) **D**ataflow

Execution in SCAD

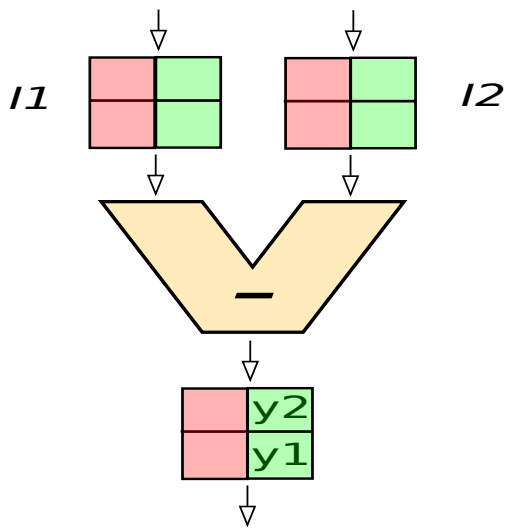
program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4



move program

$\rightsquigarrow$ $O1 \rightarrow I1$ ✓
$O2 \rightarrow I2$ ✓
$O1 \rightarrow I1$ ✓
$O3 \rightarrow I2$ ✓



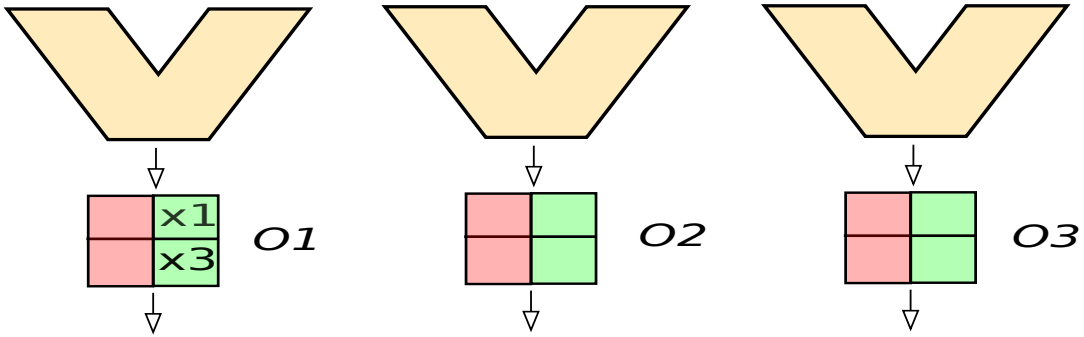
Sync (sequential) **C**ontrol **A**sync (parallel) **D**ataflow

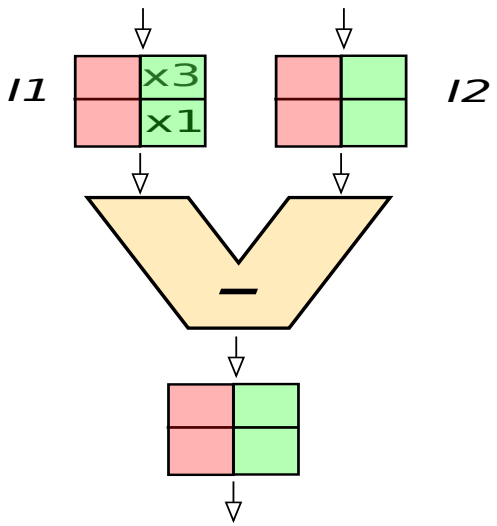
Execution in SCAD

program

$$\rightsquigarrow \begin{aligned} y1 &= x1 - x2 \\ y2 &= x3 - x4 \end{aligned}$$

move program ?



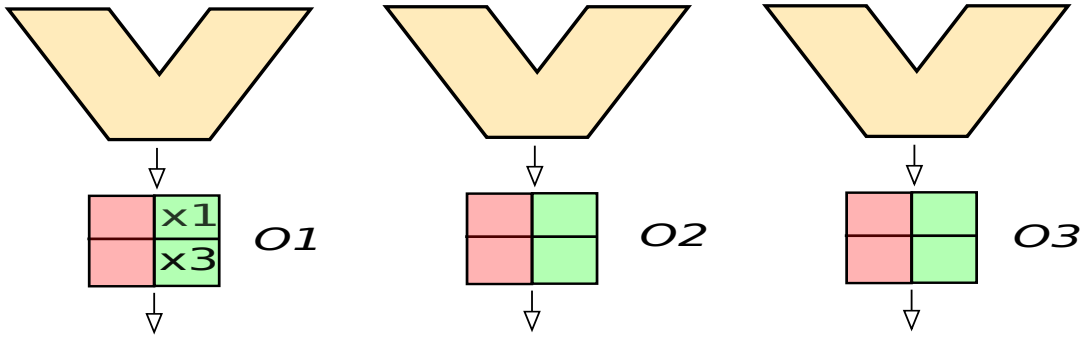


Sync (sequential) **C**ontrol **A**sync (parallel) **D**ataflow

Execution in SCAD

program

$\rightsquigarrow$ y1 = x1 − x2
y2 = x3 − x4

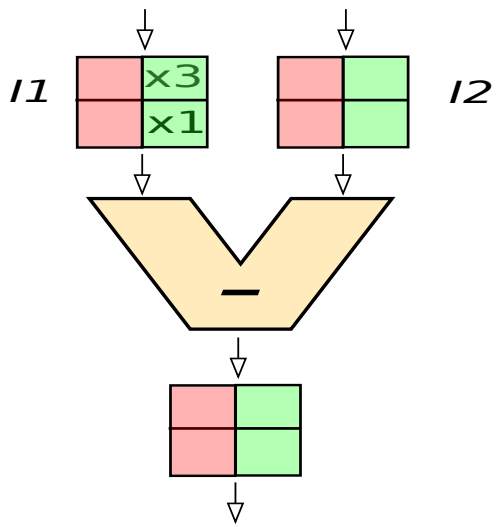


move program ?

compilation for SCAD

$\rightsquigarrow$ **(1) variable ordering**

$\rightsquigarrow$ **(2) PU assignment**



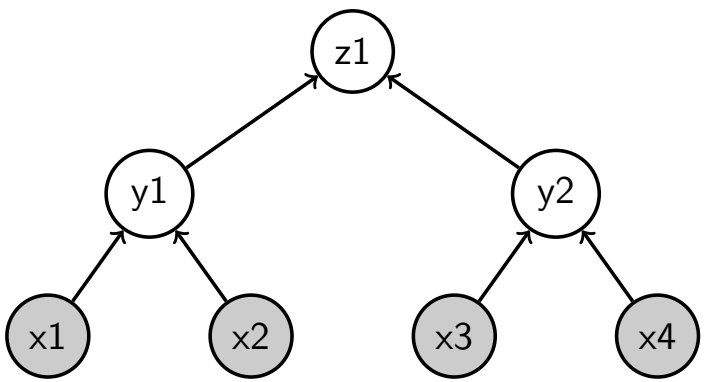
Sync (sequential) **C**ontrol **A**sync (parallel) **D**ataflow

Code Generation Idea: Expression Tree

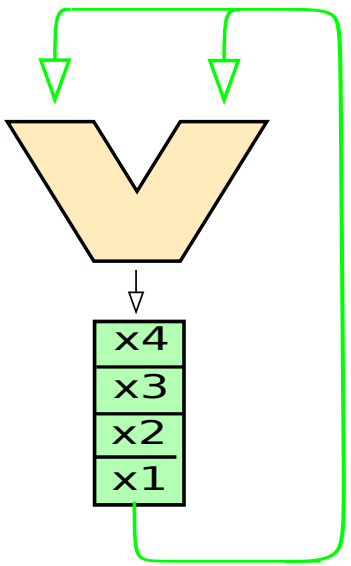
program

$$y1 = x1 - x2$$
$$y2 = x3 - x4$$
$$z1 = y1 + y2$$

breadth-first ordering

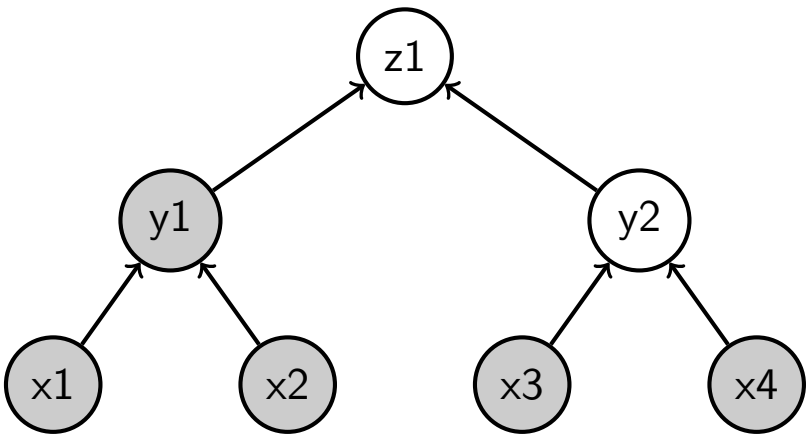


executed `x1,x2,x3,x4`

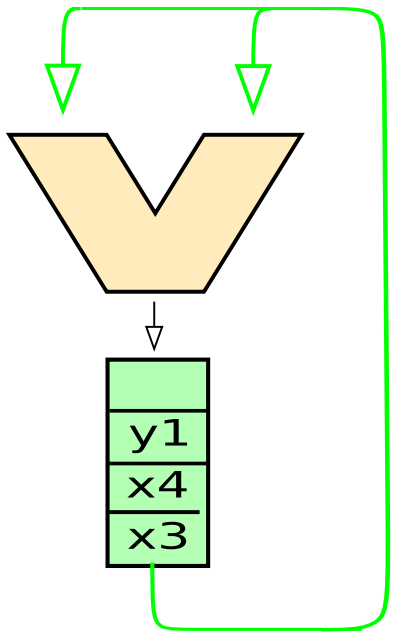


Code Generation Idea: Expression Tree

breadth-first ordering

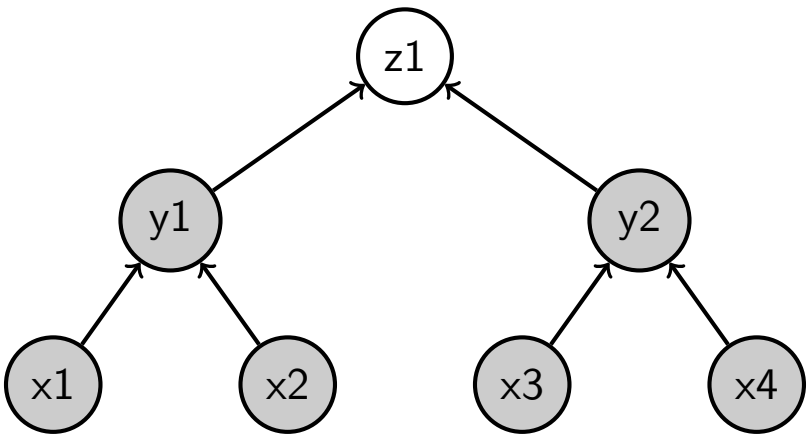


executed y1

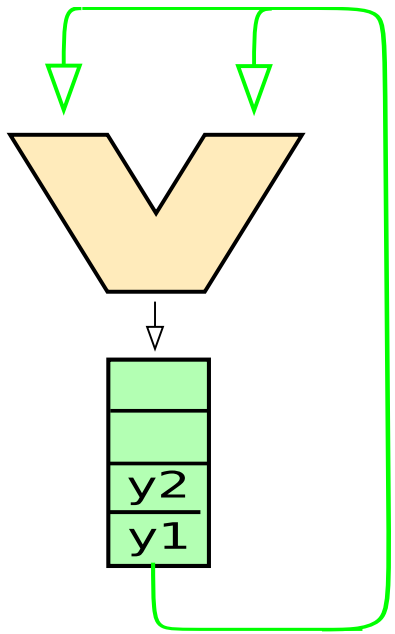


Code Generation Idea: Expression Tree

breadth-first ordering

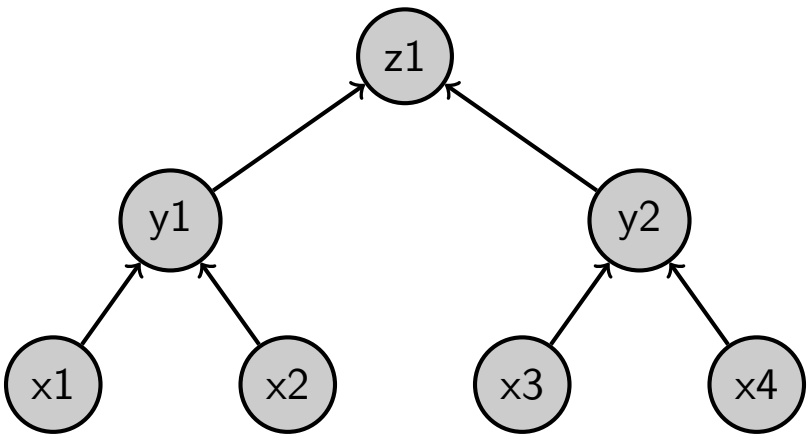


executed y2

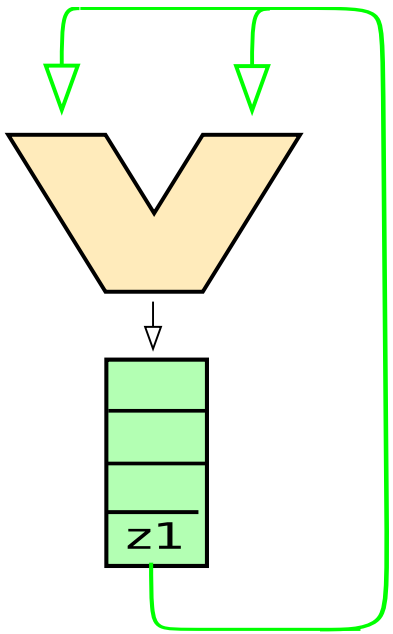


Code Generation Idea: Expression Tree

breadth-first ordering



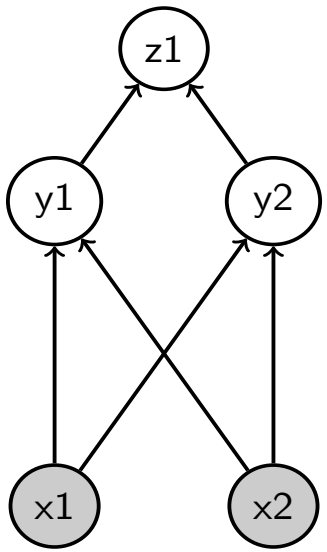
executed z1

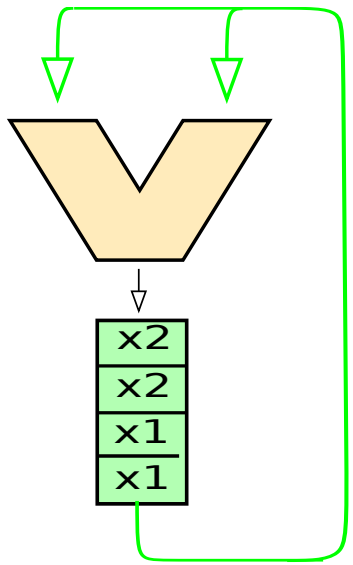


Code Generation Idea: Expression DAG

program

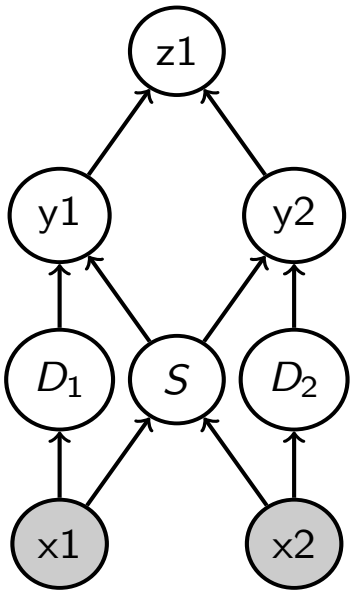
$$y1 = x1 \;/\; x2$$
$$y2 = x2 - x1$$
$$z1 = y1 + y2$$

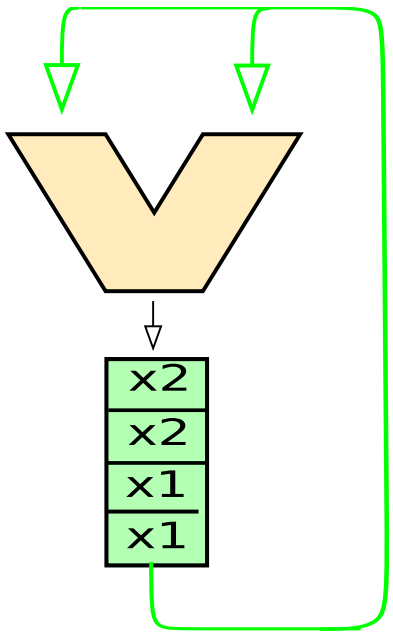




executed `x1,x2`

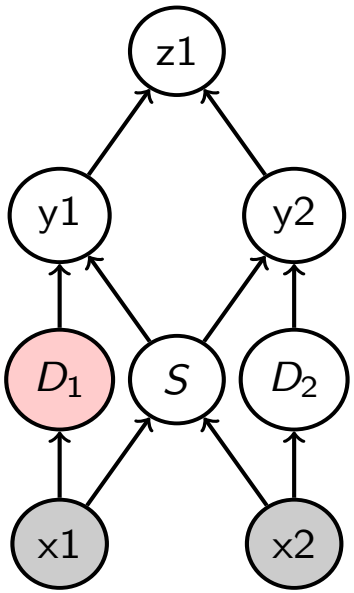
Code Generation Idea: Expression DAG

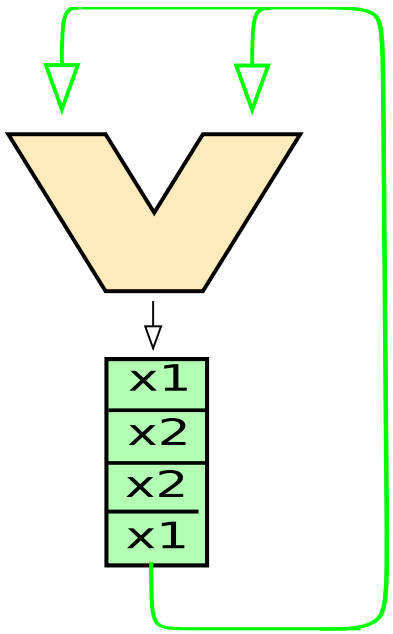




reordering overhead !

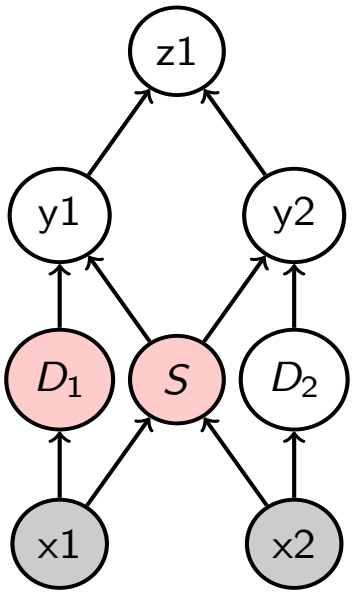
Code Generation Idea: Expression DAG

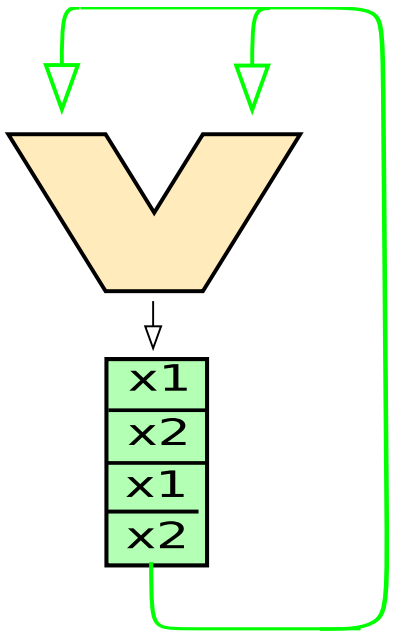




executed D_1 duplicate

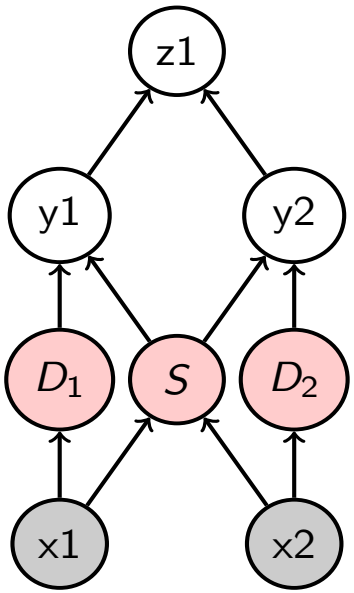
Code Generation Idea: Expression DAG

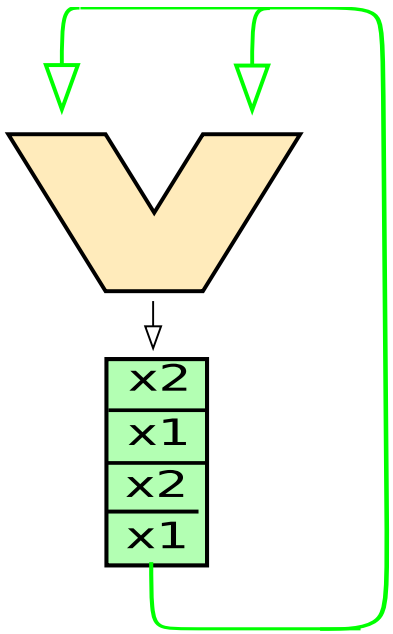




executed S swap

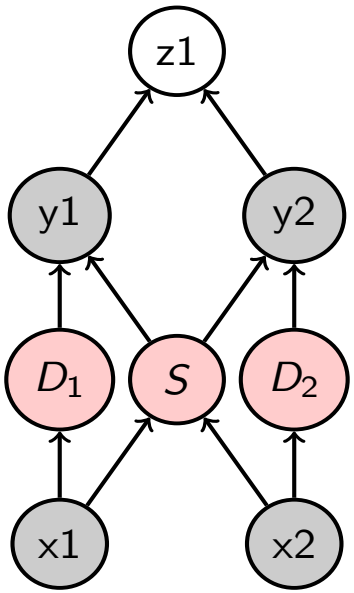
Code Generation Idea: Expression DAG

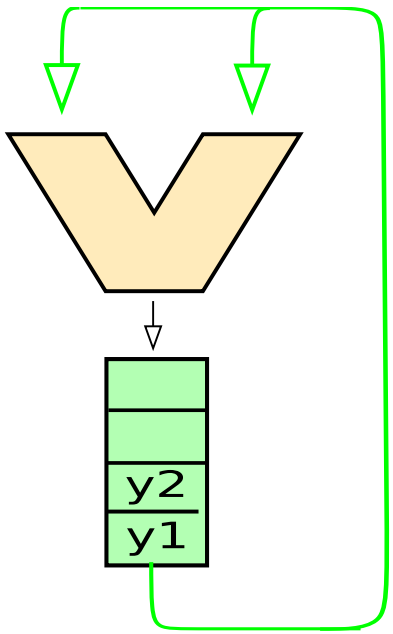




executed D_2 duplicate

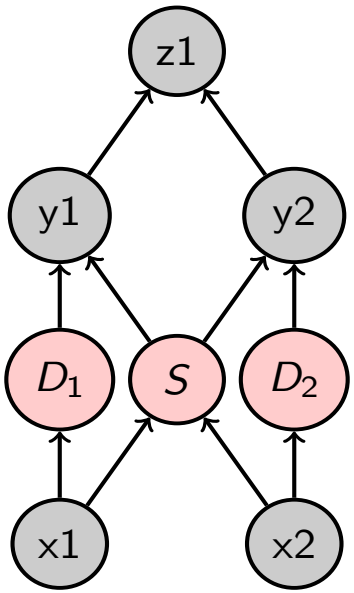
Code Generation Idea: Expression DAG

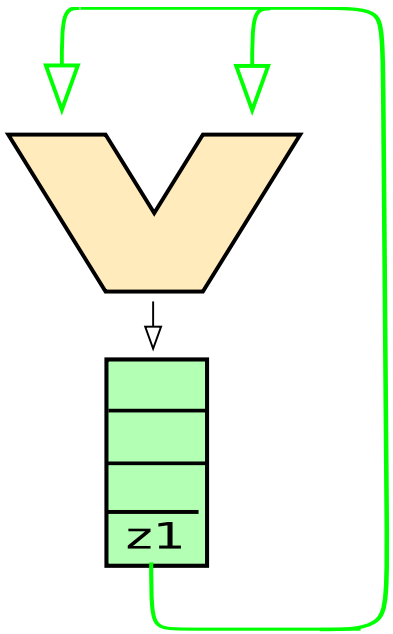




executed `y1,y2`

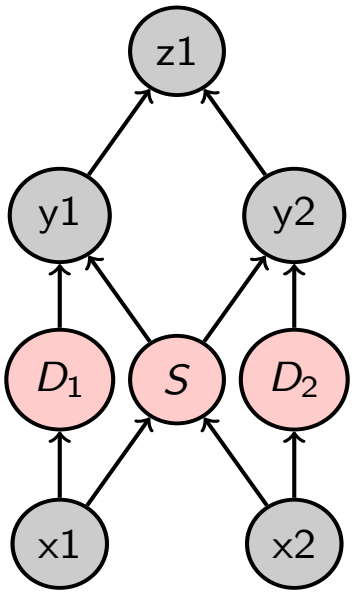
Code Generation Idea: Expression DAG

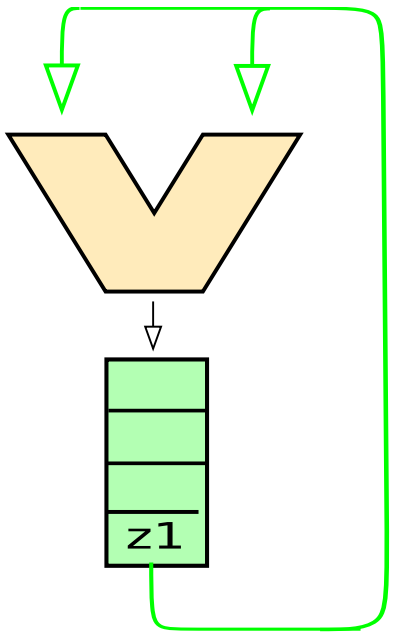




executed z1

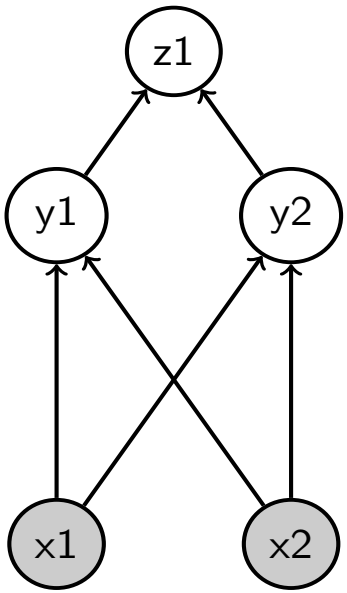
Code Generation Idea: Expression DAG

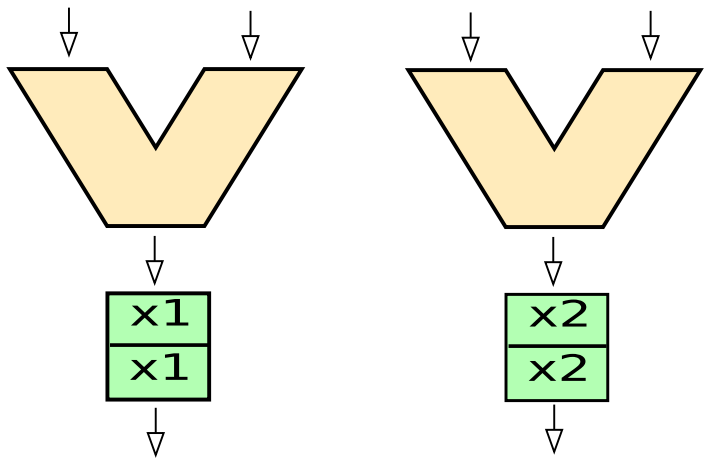




overhead $D_1, S, D_2 \rightsquigarrow$ performance loss

Overhead in Multiple PU SCAD

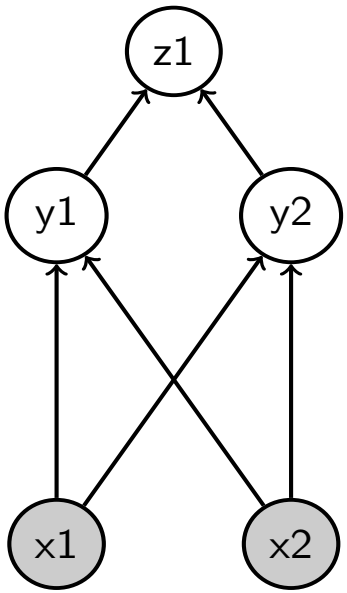


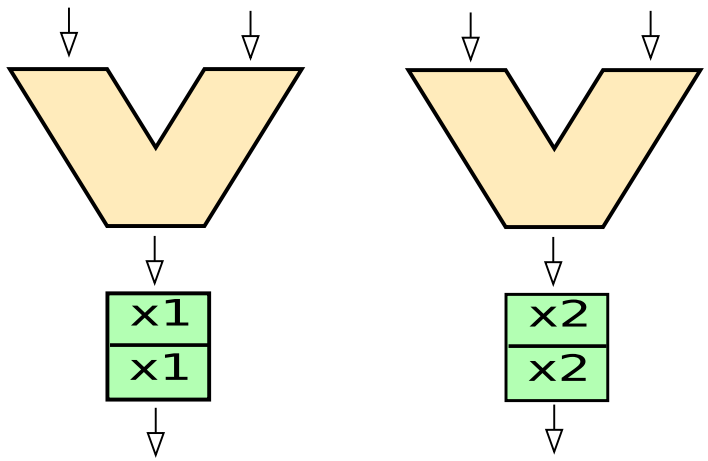


executed `x1,x2`

less access restrictions $\rightsquigarrow$ less overhead !

Overhead in Multiple PU SCAD





executed `x1,x2`

less access restrictions $\rightsquigarrow$ less overhead !

Outline

Optimal Code Generation

determine if a given program can be executed on a given SCAD machine with p PUs, **without any overhead**

$\rightsquigarrow$ NP-hard $\Leftarrow$ reduction from graph coloring

mapping to SAT

- relations

$\rightsquigarrow$ variable ordering $x_i \prec x_j$

x_i must be consumed before x_j

$\rightsquigarrow$ PU assignment $\alpha_{i,j}$

x_i is assigned to PU j

Optimal Code Generation

determine if a given program can be executed on a given SCAD machine with p PUs, **without any overhead**

$\rightsquigarrow$ NP-hard $\Leftarrow$ reduction from graph coloring

mapping to SAT

- relations

$\rightsquigarrow$ variable ordering $\mathrm{x}i \prec \mathrm{x}j$

$\mathrm{x}i$ must be consumed before $\mathrm{x}j$

$\rightsquigarrow$ PU assignment $\alpha_{i,j}$

$\mathrm{x}i$ is assigned to PU j

Optimal Code Generation

determine if a given program can be executed on a given SCAD machine with p PUs, **without any overhead**

$\rightsquigarrow$ NP-hard $\Leftarrow$ reduction from graph coloring

mapping to SAT

- relations

$\rightsquigarrow$ variable ordering $\mathrm{x}i \prec \mathrm{x}j$

$\mathrm{x}i$ must be consumed before $\mathrm{x}j$

$\rightsquigarrow$ PU assignment $\alpha_{i,j}$

$\mathrm{x}i$ is assigned to PU j

Optimal Code Generation

determine if a given program can be executed on a given SCAD machine with p PUs, **without any overhead**

$\rightsquigarrow$ NP-hard $\Leftarrow$ reduction from graph coloring

mapping to SAT

- relations

$\rightsquigarrow$ variable ordering $x_i \prec x_j$

x_i must be consumed before x_j

$\rightsquigarrow$ PU assignment $\alpha_{i,j}$

x_i is assigned to PU j

Optimal Code Generation

determine if a given program can be executed on a given SCAD machine with p PUs, **without any overhead**

$\rightsquigarrow$ NP-hard $\Leftarrow$ reduction from graph coloring

mapping to SAT

- relations

$\rightsquigarrow$ variable ordering $\mathrm{x}i \prec \mathrm{x}j$

$\mathrm{x}i$ must be consumed before $\mathrm{x}j$

$\rightsquigarrow$ PU assignment $\alpha_{i,j}$

$\mathrm{x}i$ is assigned to PU j

SAT Constraints - Idea

unique PU assignment constraint

data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order

SAT matching is complete

$\rightsquigarrow$ necessary and sufficient

SAT Constraints - Idea

unique PU assignment constraint

data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order

$\rightsquigarrow$ SAT-matching is complete

$\rightsquigarrow$ necessary and sufficient

SAT Constraints - Idea

unique PU assignment constraint

data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order

SAT mapping is complete

$\rightsquigarrow$ necessary and sufficient

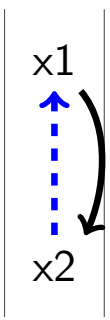
SAT Constraints - Idea

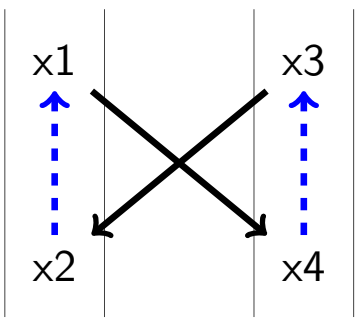
unique PU assignment constraint

data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order





SAT mapping is complete

$\rightsquigarrow$ necessary and sufficient

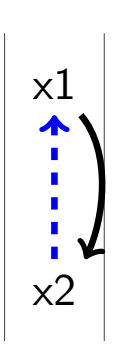
SAT Constraints - Idea

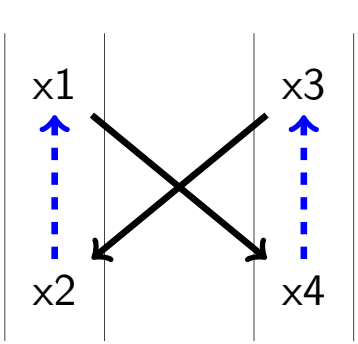
unique PU assignment constraint

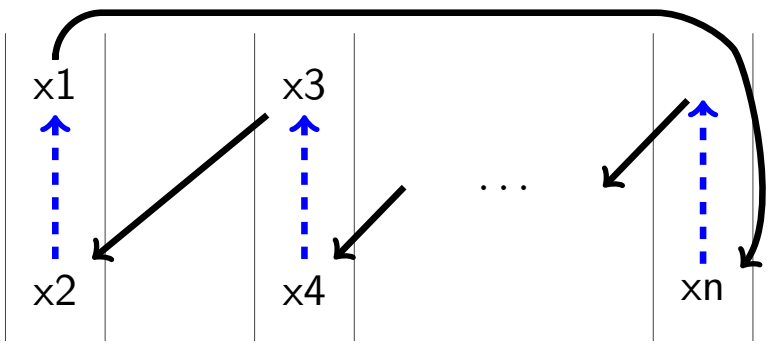
data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order







SAT mapping is complete

$\rightsquigarrow$ necessary and sufficient

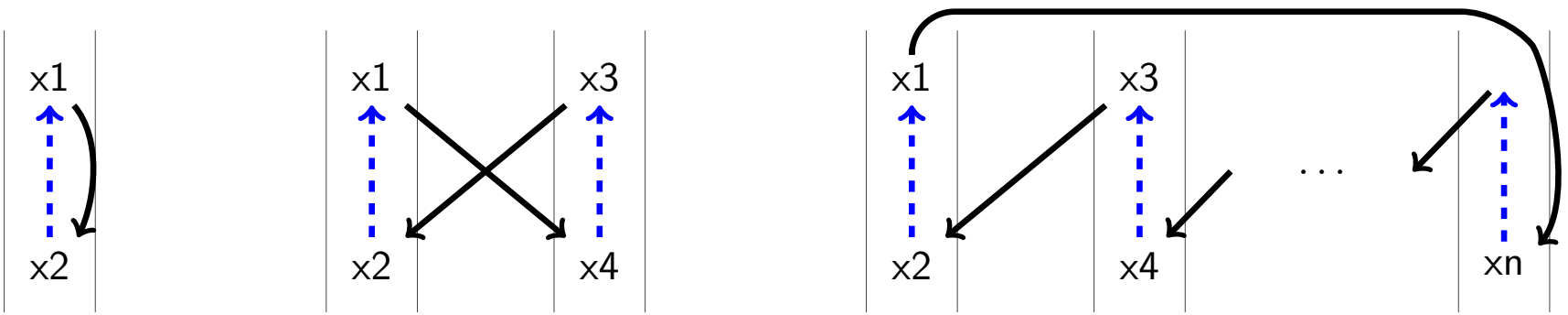
SAT Constraints - Idea

unique PU assignment constraint

data dependency constraint

buffer constraint

$\rightsquigarrow$ avoid cycles in consumption order



SAT mapping is complete

$\rightsquigarrow$ necessary and sufficient

Optimal Code Generation: An Example

program	constraints	solution	move program
x4 = x0 + x1 x5 = x2 ∗ x3 x6 = x4 / x5 x7 = x5 − x4 x8 = x6 + x7	$\left(\bigwedge_{x_i \in \mathcal{V}} \bigvee_{k=0}^{2} \alpha_{i,k}\right) \wedge$ $\left(\bigwedge_{x_i \in \mathcal{V}} \bigwedge_{k=0}^{2} \alpha_{i,k} \Rightarrow \bigwedge_{j=0, j \neq k}^{2} \neg \alpha_{i,j}\right)$ $\beta_{i,j} :\Leftrightarrow \bigvee_{k=0}^{2} \alpha_{i,k} \wedge \alpha_{j,k}$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \beta_{i,j} \Rightarrow \left(\begin{array}{c} x_i \prec x_j \Rightarrow \neg x_j \prec_d x_i \\ \wedge \\ x_j \prec x_i \Rightarrow \neg x_i \prec_d x_j \end{array} \right)$ $\bigwedge_{x_i, x_j, x_k \in \mathcal{V}} x_i \prec x_j \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i \in \mathcal{V}} \neg x_i \prec x_i$ $\gamma_{i,j,k} :\Leftrightarrow \beta_{j,k} \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \bigwedge_{x_k \in \mathcal{V}} \beta_{i,j} \Rightarrow \left(\begin{array}{c} \left(\begin{array}{c} x_i \prec x_j \\ \wedge \\ \gamma_{\mathsf{L}(i),\mathsf{L}(j),k} \wedge \gamma_{\mathsf{R}(i),\mathsf{R}(j),k} \end{array} \right) \\ \vee \\ \left(\begin{array}{c} x_j \prec x_i \\ \wedge \\ \gamma_{\mathsf{L}(j),\mathsf{L}(i),k} \wedge \gamma_{\mathsf{R}(j),\mathsf{R}(i),k} \end{array} \right) \end{array} \right)$	0 1 2 x3 x5 x4 x2 x6 x8 x0 x7 x1	3 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 2 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 0 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 1 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc lsu@out → pu0@in1 lsu@out → pu0@in0 (*,2) → pu0@opc pu0@out → pu0@in1 lsu@out → pu1@in0 lsu@out → pu1@in1 (+,2) → pu1@opc

Optimal Code Generation: An Example

program	constraints	solution	move program
x4 = x0 + x1 x5 = x2 * x3 x6 = x4 / x5 x7 = x5 − x4 x8 = x6 + x7	$\left(\bigwedge_{x_i \in \mathcal{V}} \bigvee_{k=0}^{2} \alpha_{i,k}\right) \wedge$ $\left(\bigwedge_{x_i \in \mathcal{V}} \bigwedge_{k=0}^{2} \alpha_{i,k} \Rightarrow \bigwedge_{j=0, j \neq k}^{2} \neg \alpha_{i,j}\right)$ --- $\beta_{i,j} :\Leftrightarrow \bigvee_{k=0}^{2} \alpha_{i,k} \wedge \alpha_{j,k}$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} x_i \prec x_j \Rightarrow \neg x_j \prec_d x_i \\ \wedge \\ x_j \prec x_i \Rightarrow \neg x_i \prec_d x_j \end{pmatrix}$ --- $\bigwedge_{x_i, x_j, x_k \in \mathcal{V}} x_i \prec x_j \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i \in \mathcal{V}} \neg x_i \prec x_i$ $\gamma_{i,j,k} :\Leftrightarrow \beta_{j,k} \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \bigwedge_{x_k \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} \begin{pmatrix} x_i \prec x_j \\ \wedge \\ \gamma_{\mathsf{L}(i),\mathsf{L}(j),k} \wedge \gamma_{\mathsf{R}(i),\mathsf{R}(j),k} \end{pmatrix} \\ \vee \\ \begin{pmatrix} x_j \prec x_i \\ \wedge \\ \gamma_{\mathsf{L}(j),\mathsf{L}(i),k} \wedge \gamma_{\mathsf{R}(j),\mathsf{R}(i),k} \end{pmatrix} \end{pmatrix}$	0 1 2 x3 x5 x4 x2 x6 x8 x0 x7 x1	3 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 2 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 0 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 1 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc lsu@out → pu0@in1 lsu@out → pu0@in0 (*,2) → pu0@opc pu0@out → pu0@in1 lsu@out → pu1@in0 lsu@out → pu1@in1 (+,2) → pu1@opc

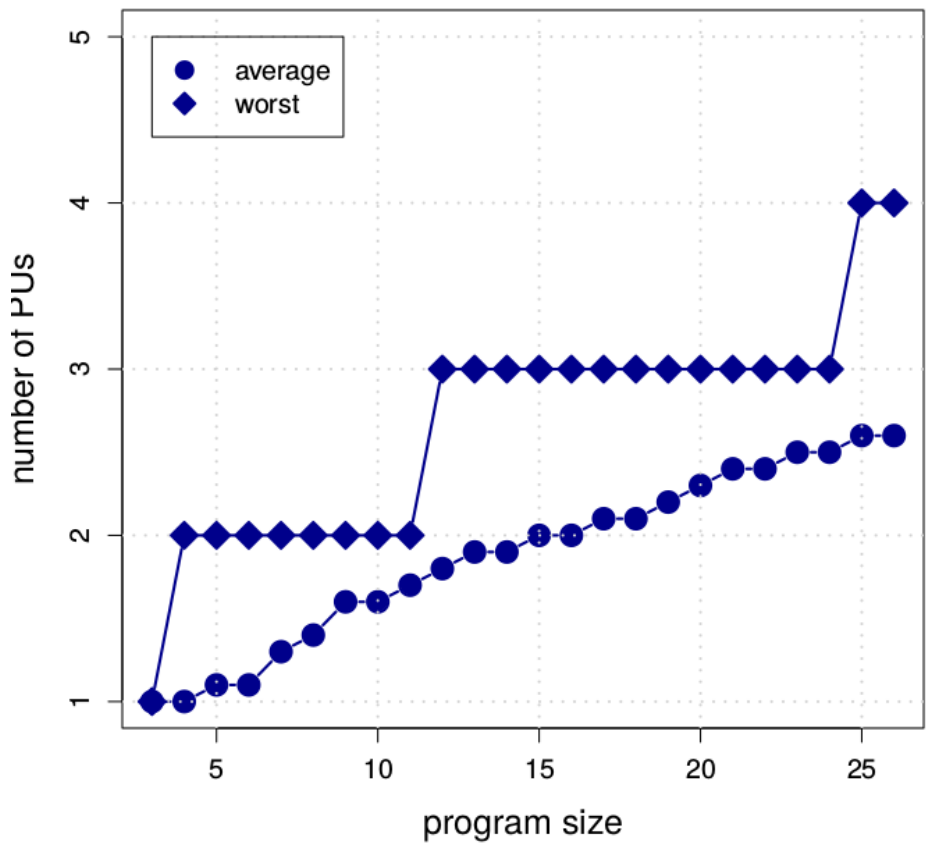
Optimal Code Generation: An Example

program	constraints	solution	move program
$x4 = x0 + x1$ $x5 = x2 * x3$ $x6 = x4 / x5$ $x7 = x5 - x4$ $x8 = x6 + x7$	$\left(\bigwedge_{x_i \in \mathcal{V}} \bigvee_{k=0}^{2} \alpha_{i,k}\right) \wedge$ $\left(\bigwedge_{x_i \in \mathcal{V}} \bigwedge_{k=0}^{2} \alpha_{i,k} \Rightarrow \bigwedge_{j=0, j \neq k}^{2} \neg \alpha_{i,j}\right)$ $\beta_{i,j} :\Leftrightarrow \bigvee_{k=0}^{2} \alpha_{i,k} \wedge \alpha_{j,k}$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} x_i \prec x_j \Rightarrow \neg x_j \prec_d x_i \\ \wedge \\ x_j \prec x_i \Rightarrow \neg x_i \prec_d x_j \end{pmatrix}$ $\bigwedge_{x_i, x_j, x_k \in \mathcal{V}} x_i \prec x_j \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i \in \mathcal{V}} \neg x_i \prec x_i$ $\gamma_{i,j,k} :\Leftrightarrow \beta_{j,k} \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i, x_j \in \mathcal{V}} \bigwedge_{x_k \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} \begin{pmatrix} x_i \prec x_j \\ \wedge \\ \gamma_{L(i),L(j),k} \wedge \gamma_{R(i),R(j),k} \end{pmatrix} \\ \vee \\ \begin{pmatrix} x_j \prec x_i \\ \wedge \\ \gamma_{L(j),L(i),k} \wedge \gamma_{R(j),R(i),k} \end{pmatrix} \end{pmatrix}$	0 1 2 x3 x5 x4 x2 x6 x8 x0 x7 x1	$3 \rightarrow$ lsu@in0 $0 \rightarrow$ lsu@in1 (ld,1) $\rightarrow$ lsu@opc $2 \rightarrow$ lsu@in0 $0 \rightarrow$ lsu@in1 (ld,1) $\rightarrow$ lsu@opc $0 \rightarrow$ lsu@in0 $0 \rightarrow$ lsu@in1 (ld,1) $\rightarrow$ lsu@opc $1 \rightarrow$ lsu@in0 $0 \rightarrow$ lsu@in1 (ld,1) $\rightarrow$ lsu@opc lsu@out $\rightarrow$ pu0@in1 lsu@out $\rightarrow$ pu0@in0 (*,2) $\rightarrow$ pu0@opc pu0@out $\rightarrow$ pu0@in1 lsu@out $\rightarrow$ pu1@in0 lsu@out $\rightarrow$ pu1@in1 (+,2) $\rightarrow$ pu1@opc

Optimal Code Generation: An Example

program	constraints	solution	move program
x4 = x0 + x1 x5 = x2 ∗ x3 x6 = x4 / x5 x7 = x5 − x4 x8 = x6 + x7	$\left(\bigwedge_{x_i \in \mathcal{V}} \bigvee_{k=0}^{2} \alpha_{i,k}\right) \wedge$ $\left(\bigwedge_{x_i \in \mathcal{V}} \bigwedge_{k=0}^{2} \alpha_{i,k} \Rightarrow \bigwedge_{j=0, j\neq k}^{2} \neg\alpha_{i,j}\right)$ $\beta_{i,j} :\Leftrightarrow \bigvee_{k=0}^{2} \alpha_{i,k} \wedge \alpha_{j,k}$ $\bigwedge_{x_i,x_j \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} x_i \prec x_j \Rightarrow \neg x_j \prec_d x_i \\ \wedge \\ x_j \prec x_i \Rightarrow \neg x_i \prec_d x_j \end{pmatrix}$ $\bigwedge_{x_i,x_j,x_k \in \mathcal{V}} x_i \prec x_j \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i \in \mathcal{V}} \neg x_i \prec x_i$ $\gamma_{i,j,k} :\Leftrightarrow \beta_{j,k} \wedge x_j \prec x_k \Rightarrow x_i \prec x_k$ $\bigwedge_{x_i,x_j \in \mathcal{V}} \bigwedge_{x_k \in \mathcal{V}} \beta_{i,j} \Rightarrow \begin{pmatrix} \begin{pmatrix} x_i \prec x_j \\ \wedge \\ \gamma_{\mathsf{L}(i),\mathsf{L}(j),k} \wedge \gamma_{\mathsf{R}(i),\mathsf{R}(j),k} \end{pmatrix} \\ \vee \\ \begin{pmatrix} x_j \prec x_i \\ \wedge \\ \gamma_{\mathsf{L}(j),\mathsf{L}(i),k} \wedge \gamma_{\mathsf{R}(j),\mathsf{R}(i),k} \end{pmatrix} \end{pmatrix}$	0 1 2 x3 x5 x4 x2 x6 x8 x0 x7 x1	3 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 2 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 0 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc 1 → lsu@in0 0 → lsu@in1 (ld,1) → lsu@opc lsu@out → pu0@in1 lsu@out → pu0@in0 (*,2) → pu0@opc pu0@out → pu0@in1 lsu@out → pu1@in0 lsu@out → pu1@in1 (+,2) → pu1@opc

Proof of Concept: Minimal PUs



Z3 SAT solver

- ⇝ up to 26 instr programs
- ⇝ timeout of 60 seconds

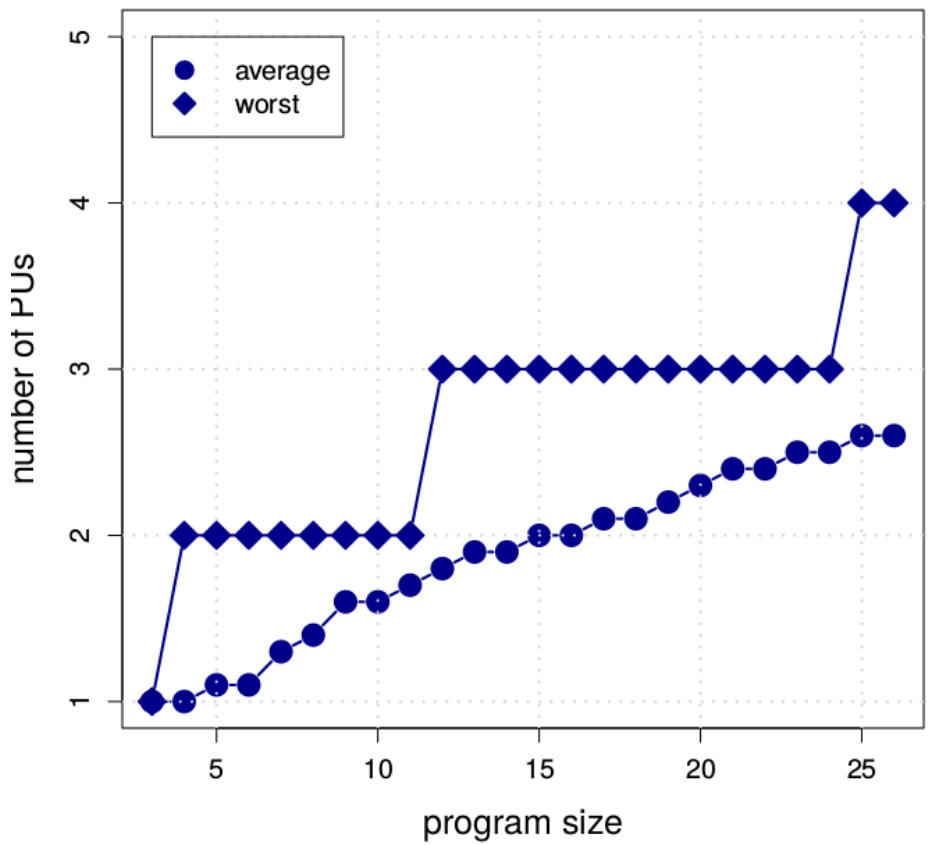
direct instr communication

- ⇝ overhead-free
- ⇝ **only up to** 4 **PUs**

heuristic

- ⇝ compile real benchmarks

Proof of Concept: Minimal PUs



Z3 SAT solver

- ⇝ up to 26 instr programs
- ⇝ timeout of 60 seconds

direct instr communication

- ⇝ overhead-free
- ⇝ **only up to 4 PUs**

heuristic

- ⇝ **compile real benchmarks**

Outline

Heuristic

variable ordering

- $\rightsquigarrow$ fixed to program order

PU assignment

- $\rightsquigarrow$ buffer interference graph
- $\rightsquigarrow$ graph coloring heuristic
- $\rightsquigarrow$ data flow analysis
  - cover control flow

$$
\begin{array}{l}
x \longleftarrow \ldots \\
y \longleftarrow \ldots \\
z \longleftarrow \ldots \\
\vdots \\
\ldots \longleftarrow x \\
\ldots \longleftarrow x \\
\ldots \longleftarrow y \\
\vdots \\
\ldots \longleftarrow z \\
\ldots \longleftarrow y
\end{array}
$$

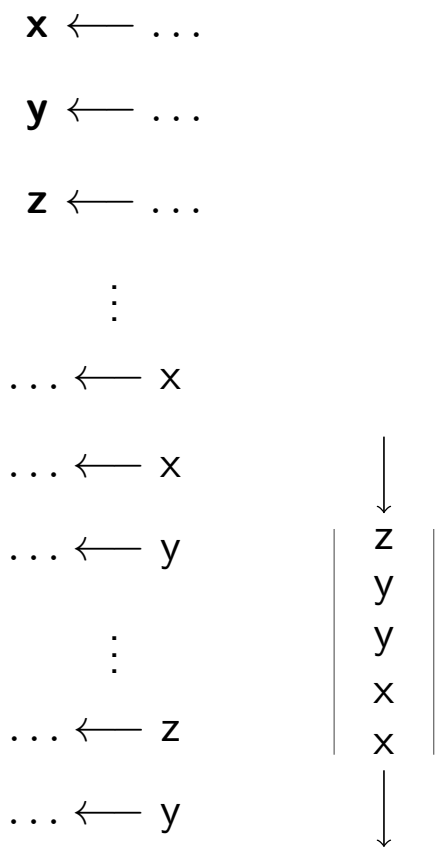
Heuristic

variable ordering

- $\rightsquigarrow$ fixed to program order

PU assignment

- $\rightsquigarrow$ buffer interference graph
- $\rightsquigarrow$ graph coloring heuristic
- $\rightsquigarrow$ data flow analysis
  - cover control flow



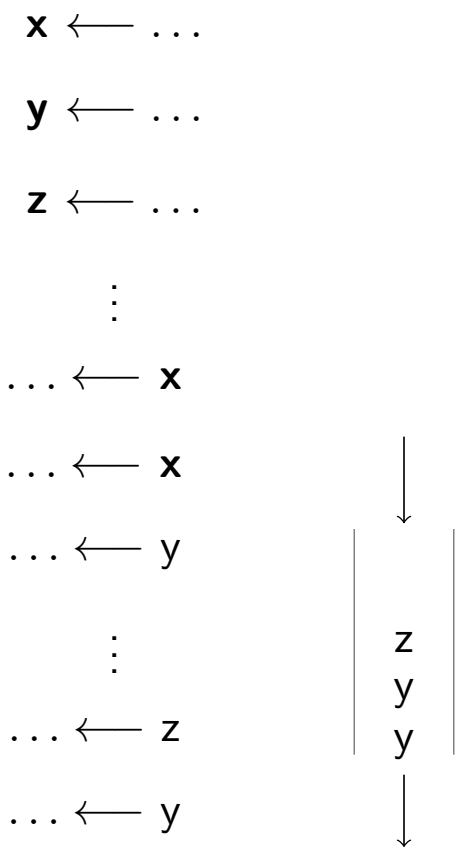
Heuristic

variable ordering

- $\rightsquigarrow$ fixed to program order

PU assignment

- $\rightsquigarrow$ buffer interference graph
- $\rightsquigarrow$ graph coloring heuristic
- $\rightsquigarrow$ data flow analysis
  - cover control flow



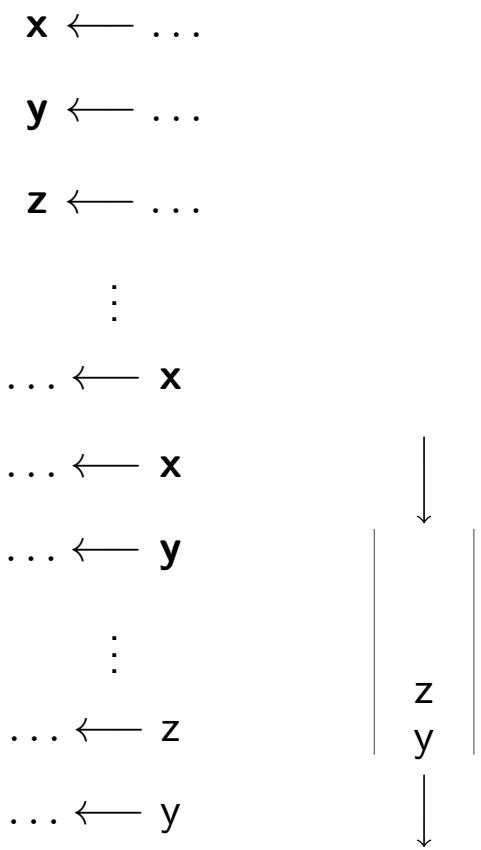
Heuristic

variable ordering

- $\rightsquigarrow$ fixed to program order

PU assignment

- $\rightsquigarrow$ buffer interference graph
- $\rightsquigarrow$ graph coloring heuristic
- $\rightsquigarrow$ data flow analysis
  - cover control flow



Heuristic

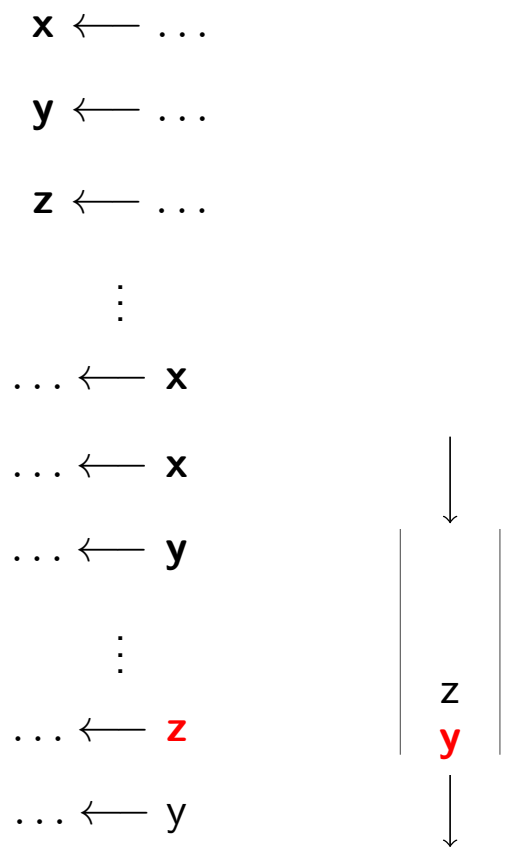
variable ordering

- ⇝ fixed to program order

PU assignment

- ⇝ buffer interference graph
- ⇝ graph coloring heuristic
- ⇝ data flow analysis

  cover control flow



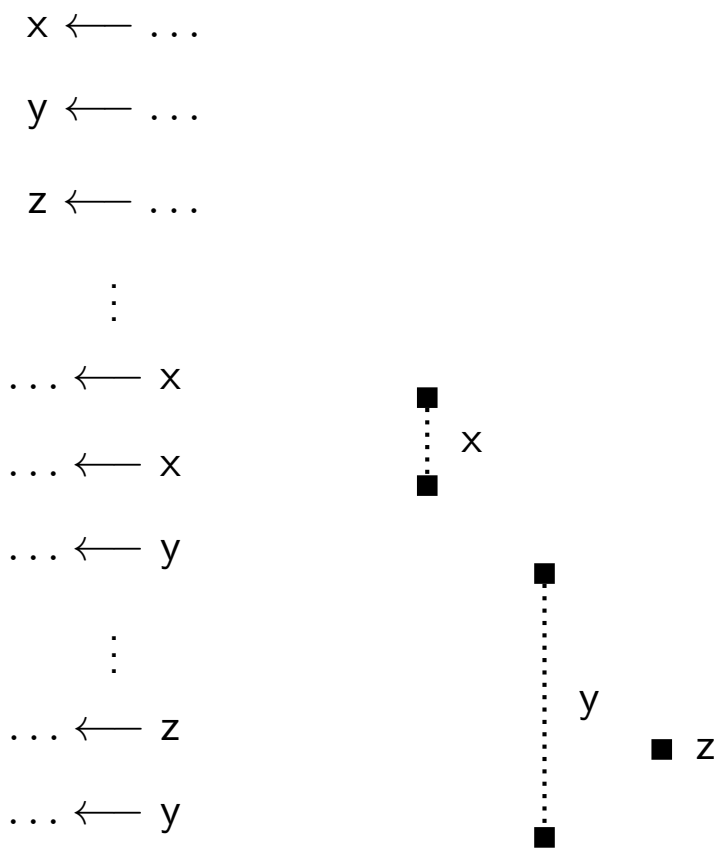
Heuristic

variable ordering

- $\rightsquigarrow$ fixed to program order

PU assignment

- $\rightsquigarrow$ buffer interference graph
- $\rightsquigarrow$ graph coloring heuristic
- $\rightsquigarrow$ data flow analysis
  - cover control flow



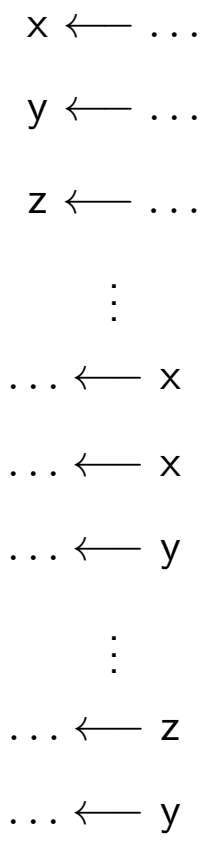
Heuristic

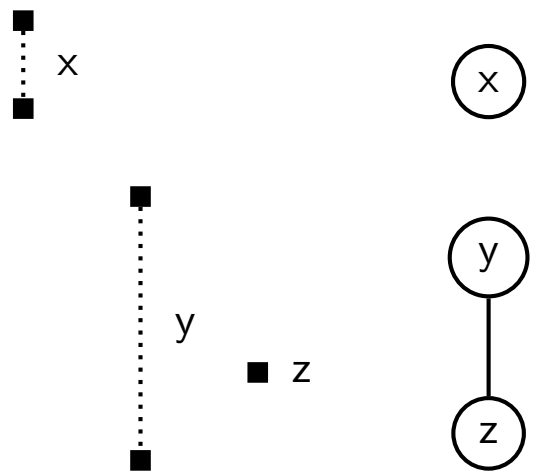
variable ordering

- ⇝ fixed to program order

PU assignment

- ⇝ buffer interference graph
- ⇝ graph coloring heuristic
- ⇝ data flow analysis
  - cover control flow





Heuristic

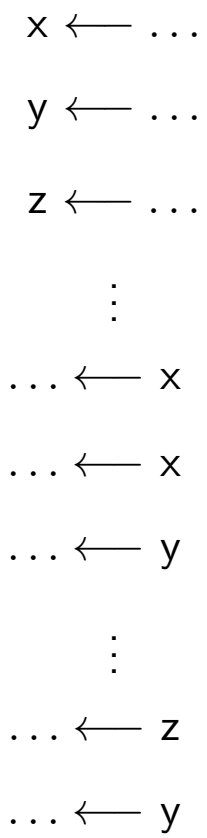
variable ordering

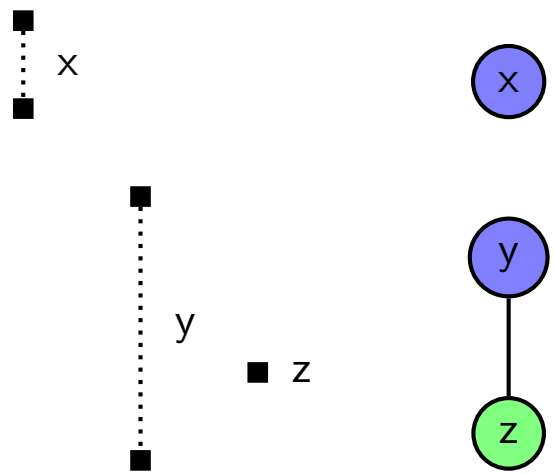
- ⇝ fixed to program order

PU assignment

- ⇝ buffer interference graph
- ⇝ graph coloring heuristic
- ⇝ data flow analysis

cover control flow





Heuristic

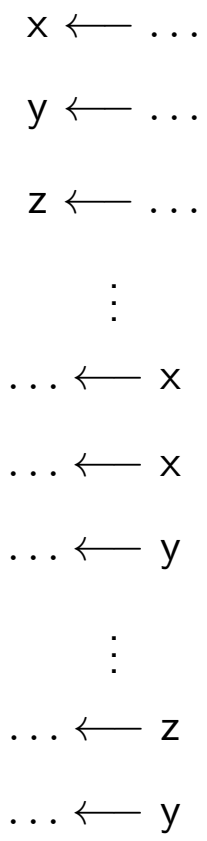
variable ordering

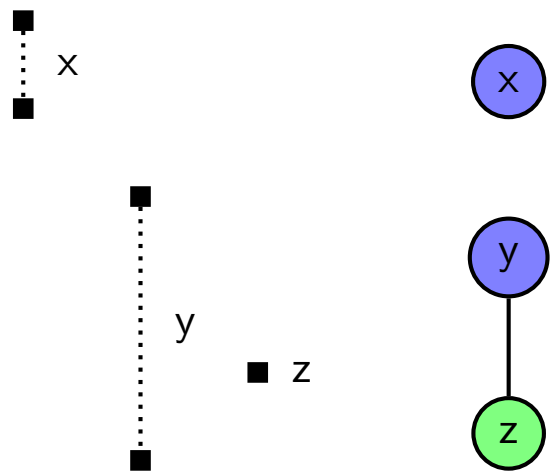
- ⇝ fixed to program order

PU assignment

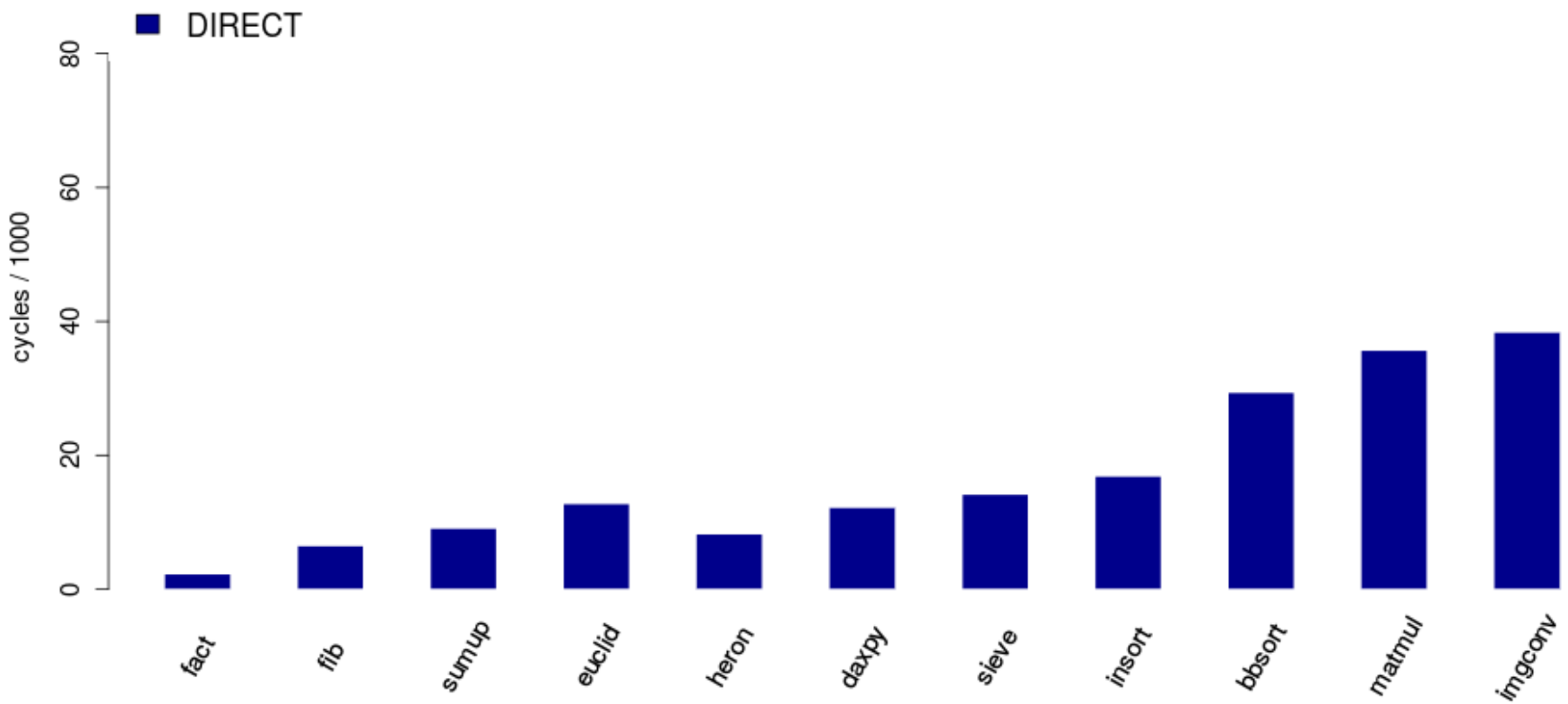
- ⇝ buffer interference graph
- ⇝ graph coloring heuristic
- ⇝ data flow analysis

  cover control flow

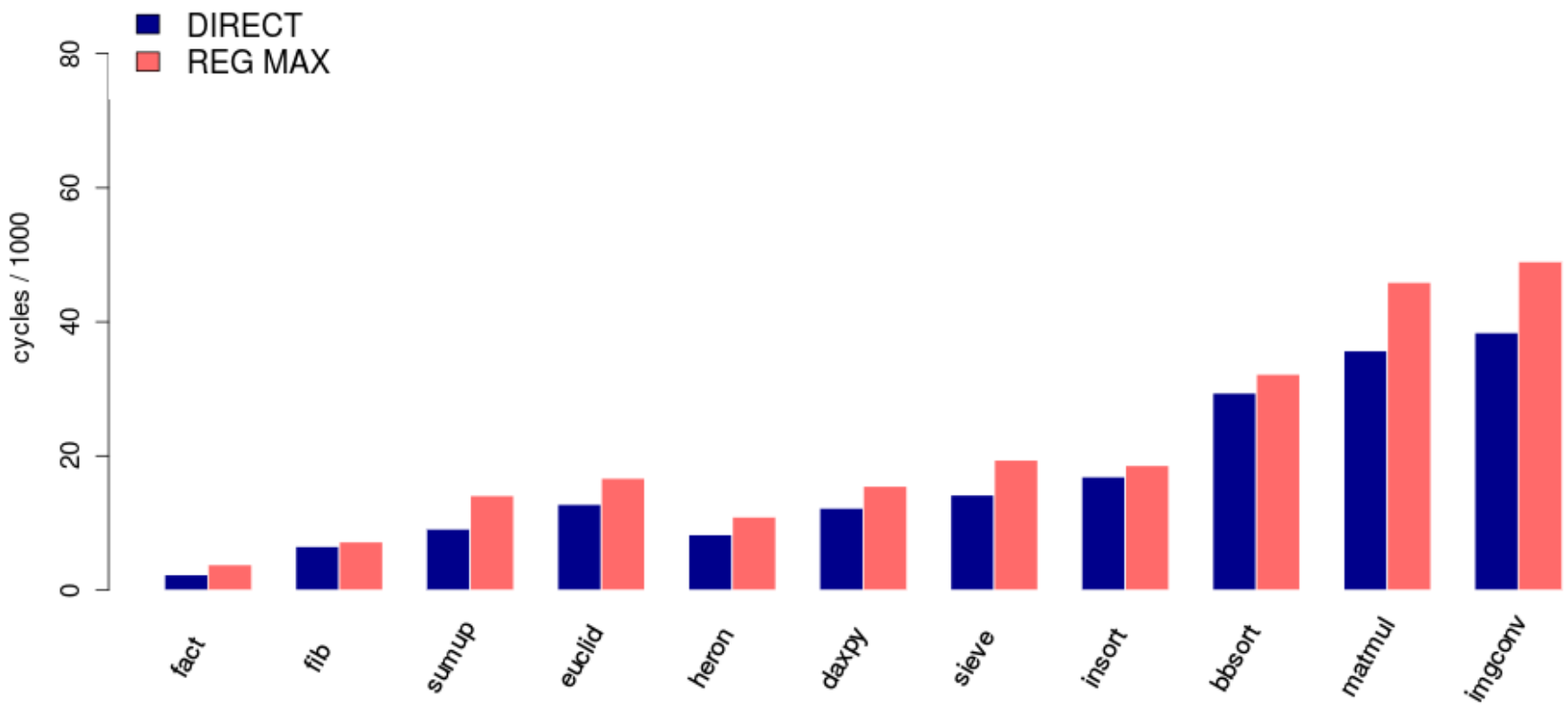




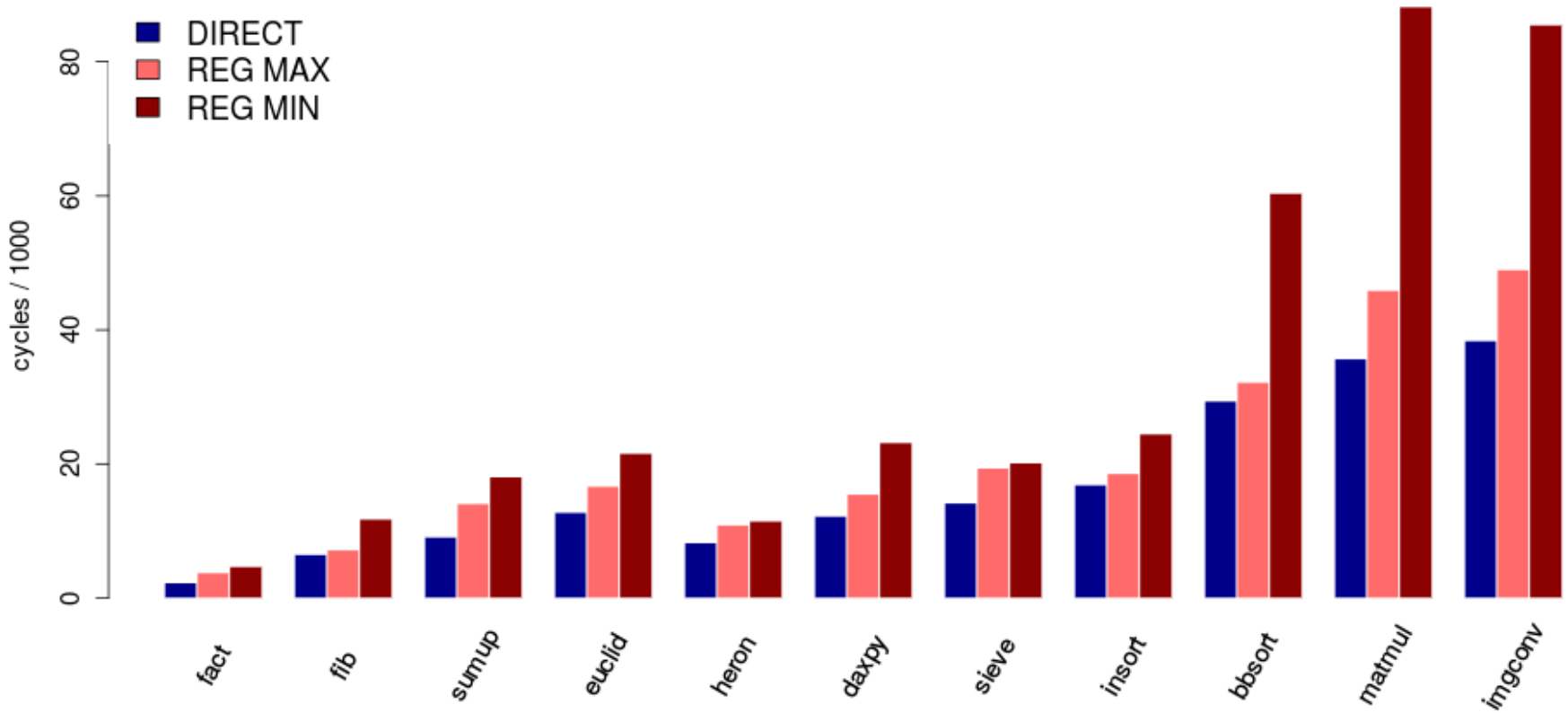
Experiments: Performance



Experiments: Performance



Experiments: Performance



Outline

Summary

commercial architectures

- ⇝ register file bottleneck ⇝ restricted use of ILP

state-of-the-art exposed datapath architectures

- ⇝ bypass registers by direct instruction communication
- ⇝ still require registers to execute programs

SCAD architecture

- ⇝ uses FIFO buffers ⇝ scalable
- ⇝ register-less code generation
  - ⇝ optimal code by SAT solver
  - ⇝ heuristic for real benchmarks

Summary

commercial architectures

- ⇝ register file bottleneck ⇝ restricted use of ILP

state-of-the-art exposed datapath architectures

- ⇝ bypass registers by direct instruction communication
- ⇝ still require registers to execute programs

SCAD architecture

- ⇝ uses FIFO buffers ⇝ scalable
- ⇝ register-less code generation
  - ⇝ optimal code by SAT solver
  - ⇝ heuristic for real benchmarks

Summary

commercial architectures

- $\rightsquigarrow$ register file bottleneck $\rightsquigarrow$ restricted use of ILP

state-of-the-art exposed datapath architectures

- $\rightsquigarrow$ bypass registers by direct instruction communication
- $\rightsquigarrow$ still require registers to execute programs

SCAD architecture

- $\rightsquigarrow$ uses FIFO buffers $\rightsquigarrow$ scalable
- $\rightsquigarrow$ register-less code generation
  - $\rightsquigarrow$ optimal code by SAT solver
  - $\rightsquigarrow$ heuristic for real benchmarks

Summary

commercial architectures

- $\rightsquigarrow$ register file bottleneck $\rightsquigarrow$ restricted use of ILP

state-of-the-art exposed datapath architectures

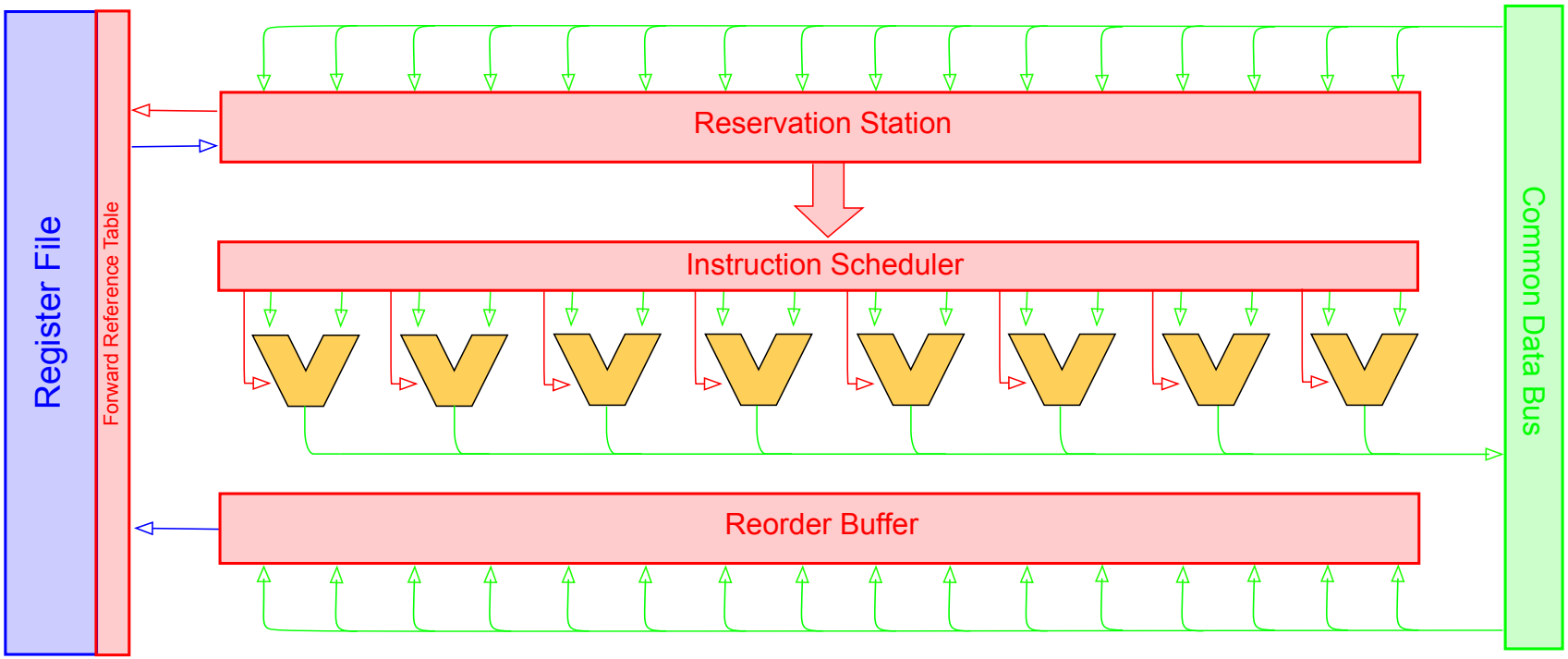
- $\rightsquigarrow$ bypass registers by direct instruction communication
- $\rightsquigarrow$ still require registers to execute programs

SCAD architecture

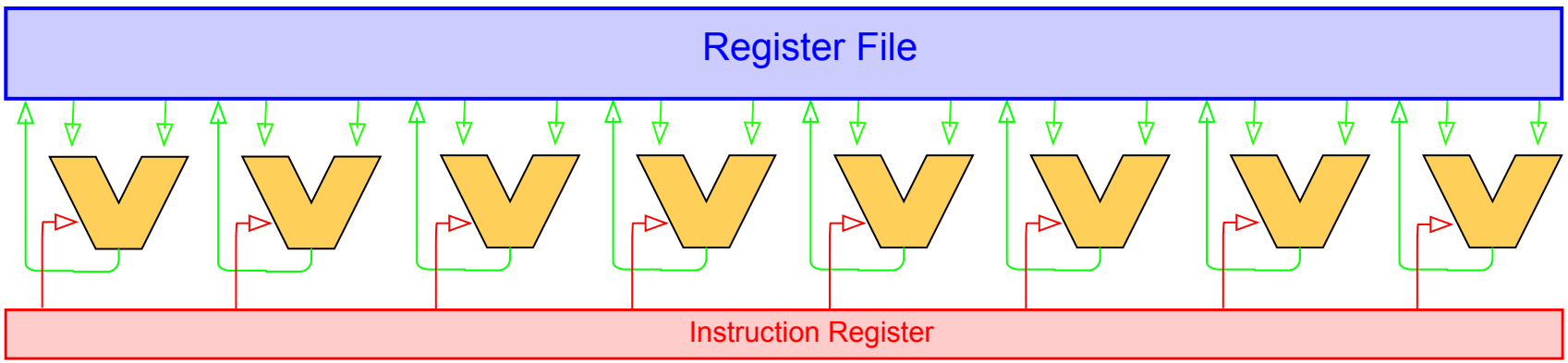
- $\rightsquigarrow$ uses FIFO buffers $\rightsquigarrow$ scalable
- $\rightsquigarrow$ register-less code generation
  - $\rightsquigarrow$ optimal code by SAT solver
  - $\rightsquigarrow$ heuristic for real benchmarks

Thank You!

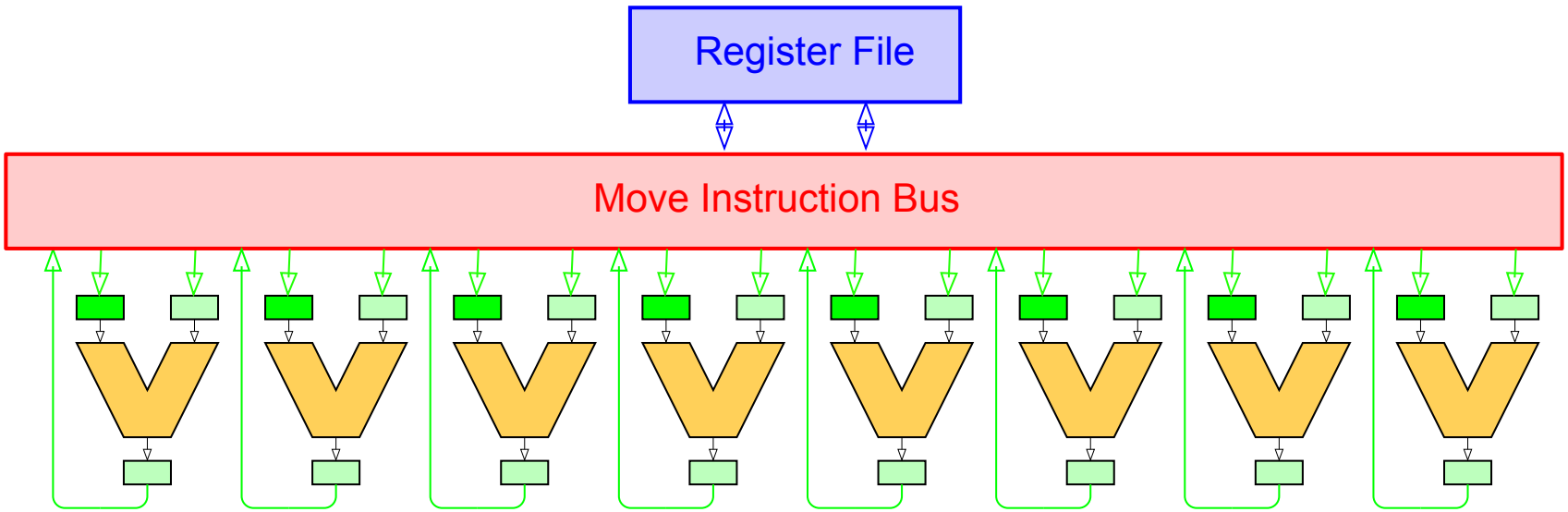
Superscalar design



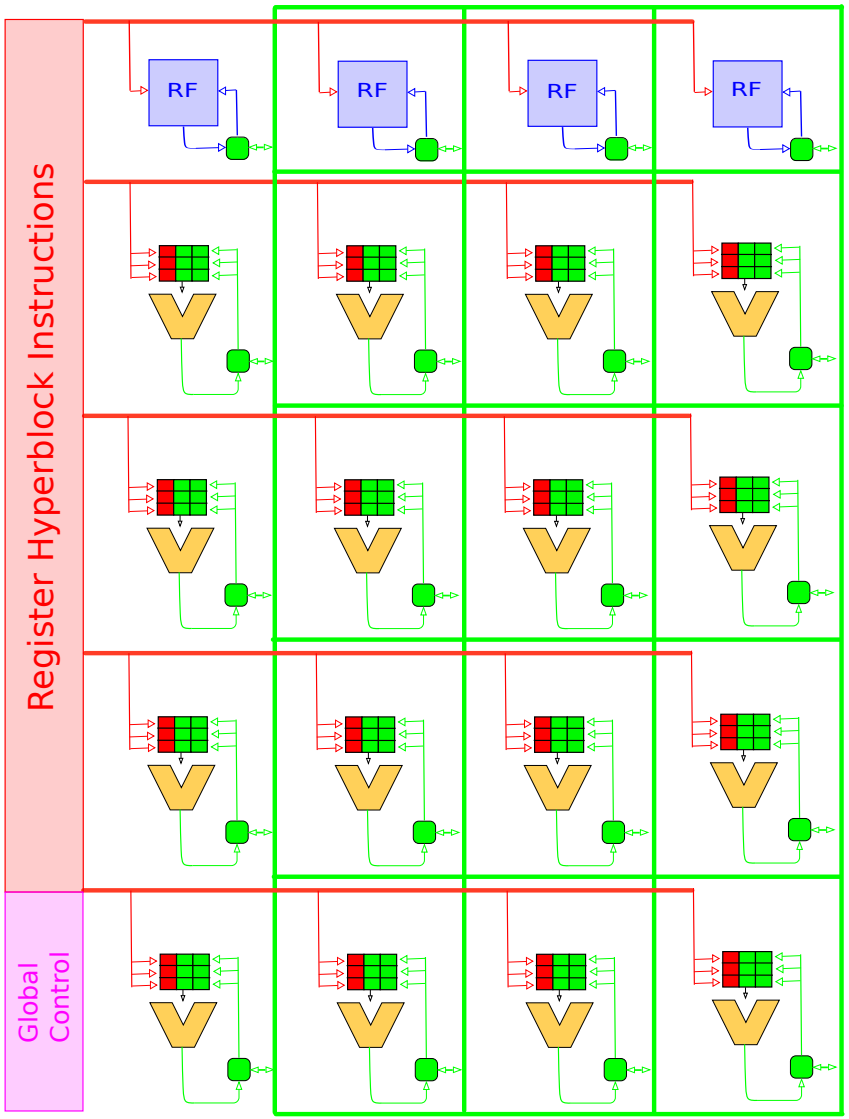
VLIW design



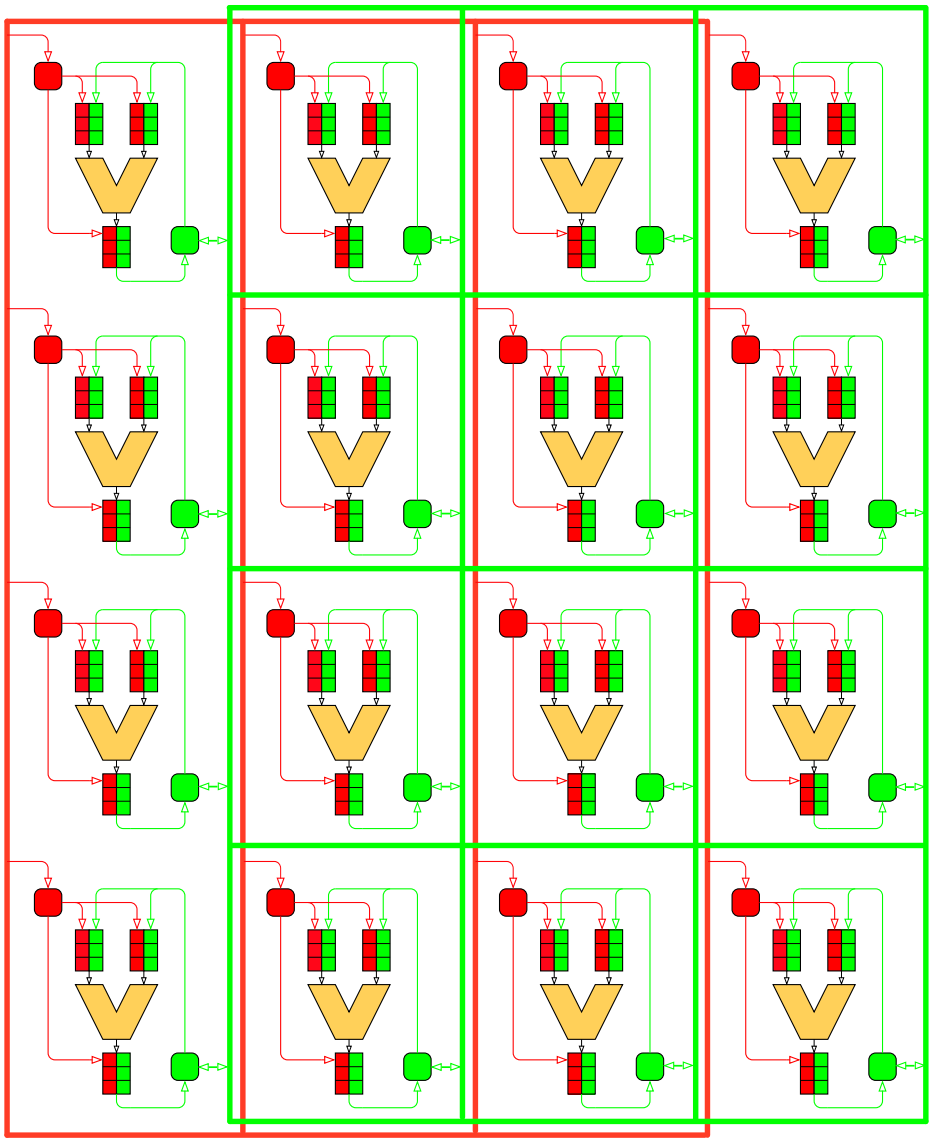
TTA design



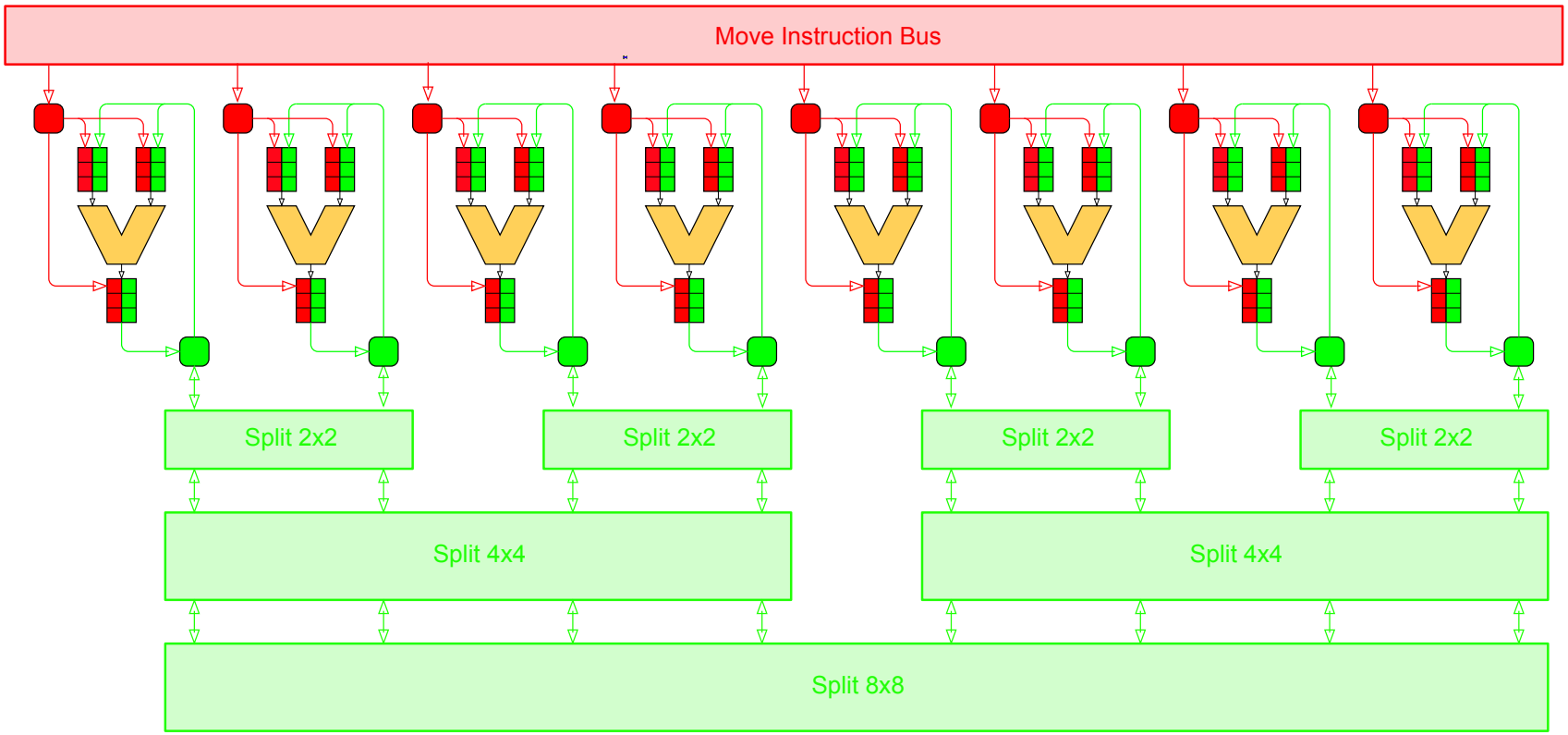
TRIPS design



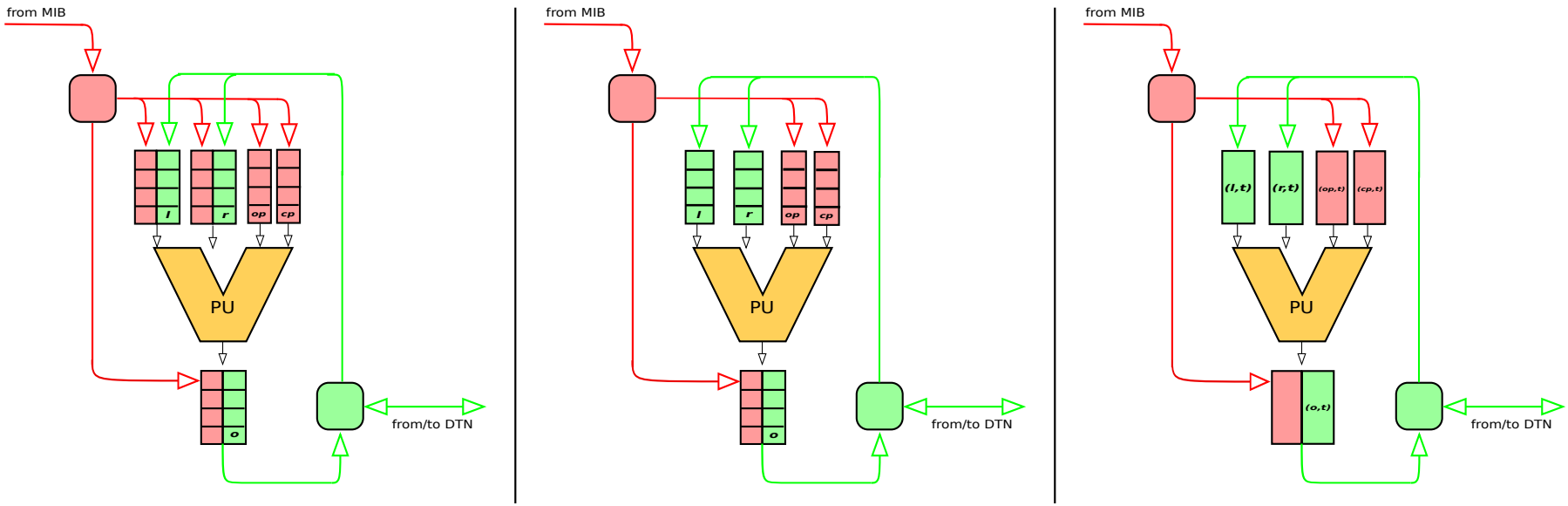
SCAD with mesh network



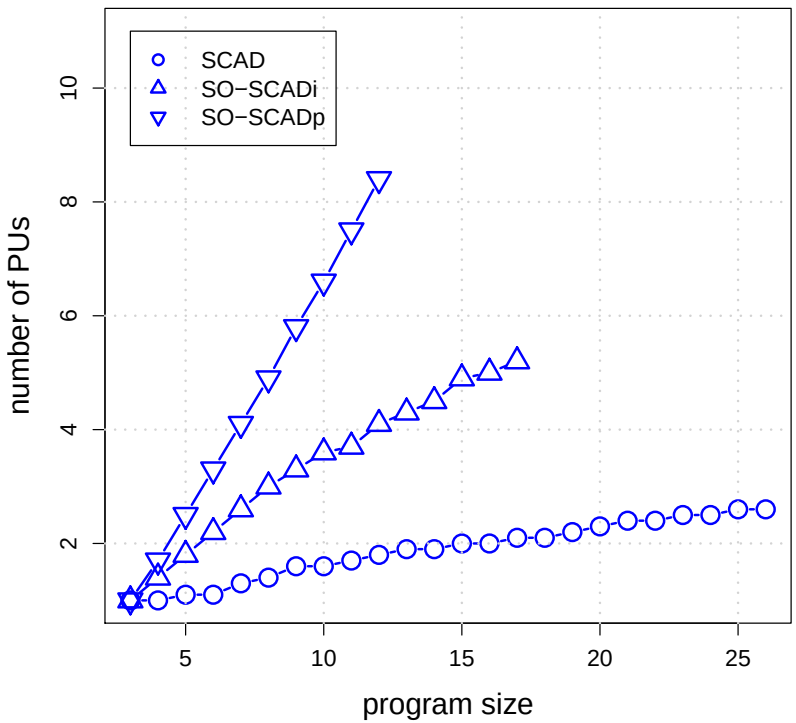
SCAD with fat tree network



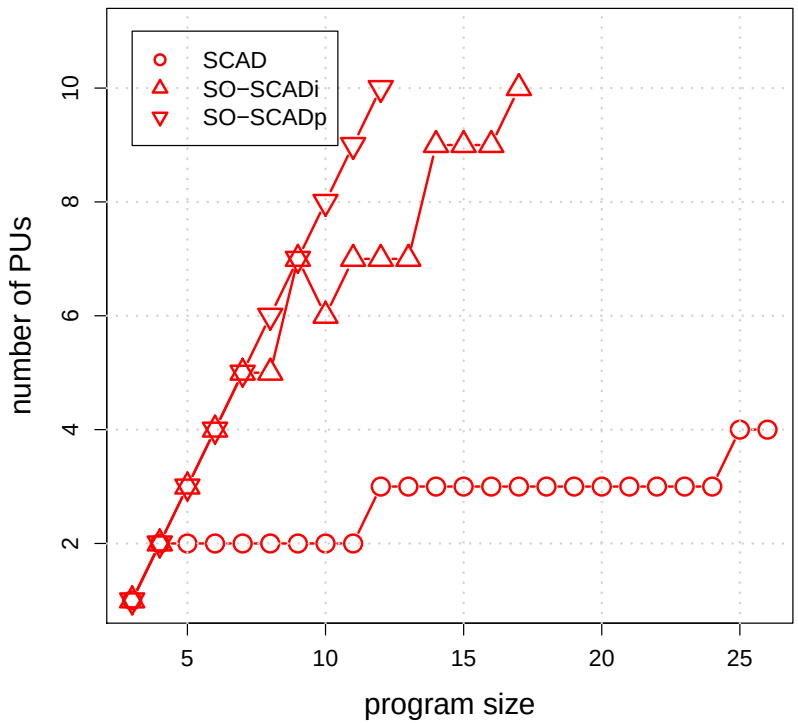
SCAD, SO-SCAD, DO-SCAD PUs



Minimal PU: program size

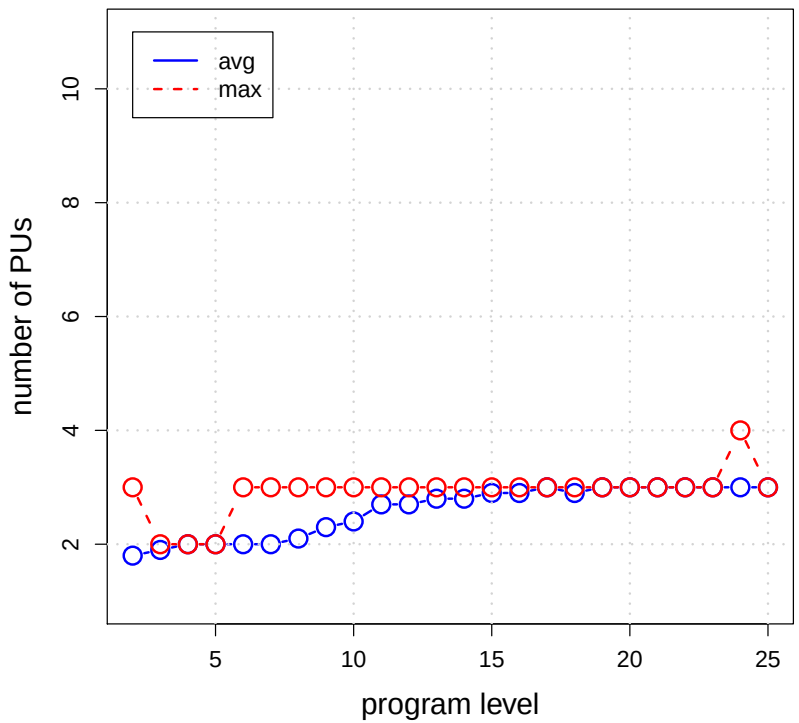


average

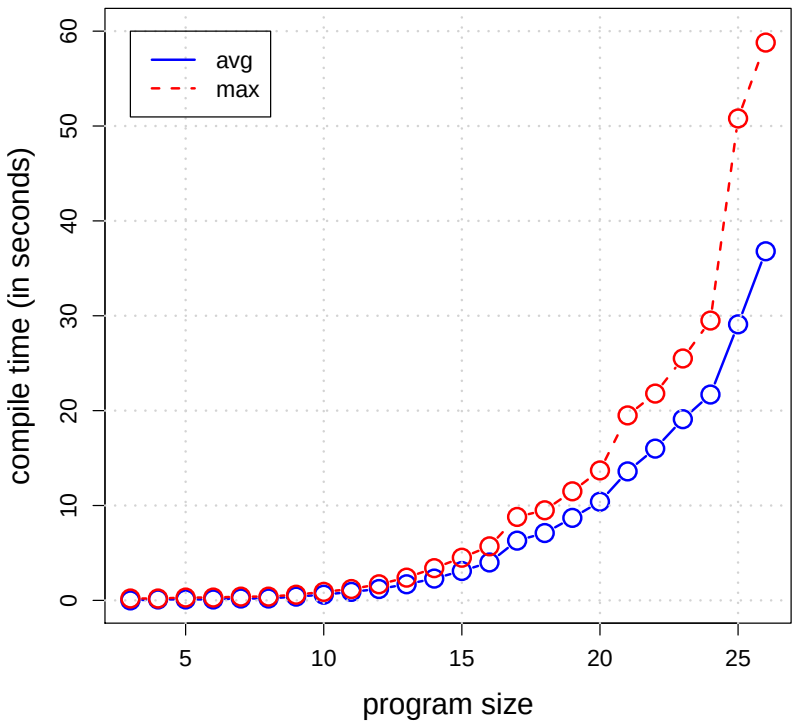


worst

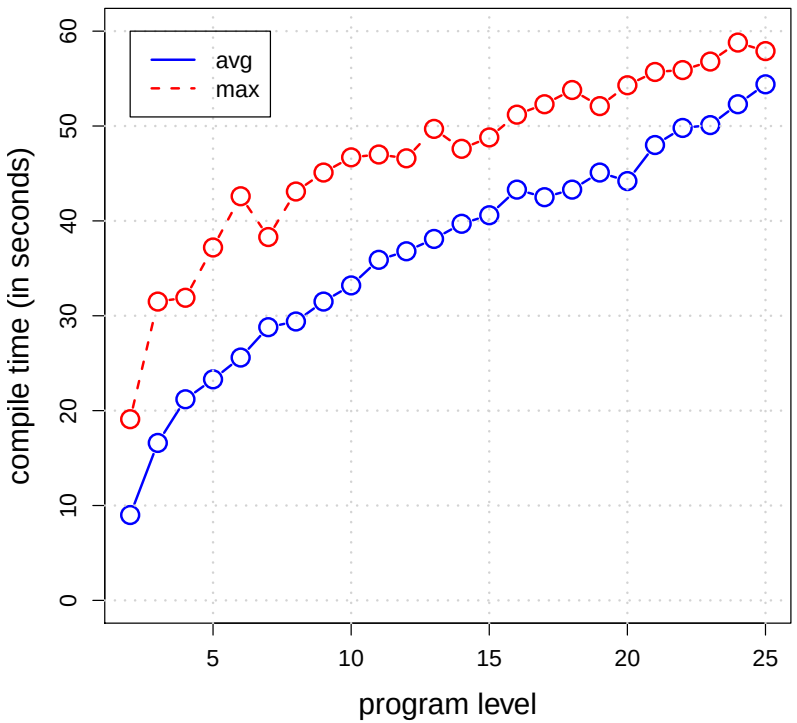
Minimal PU: program level



Optimal code generation feasibility

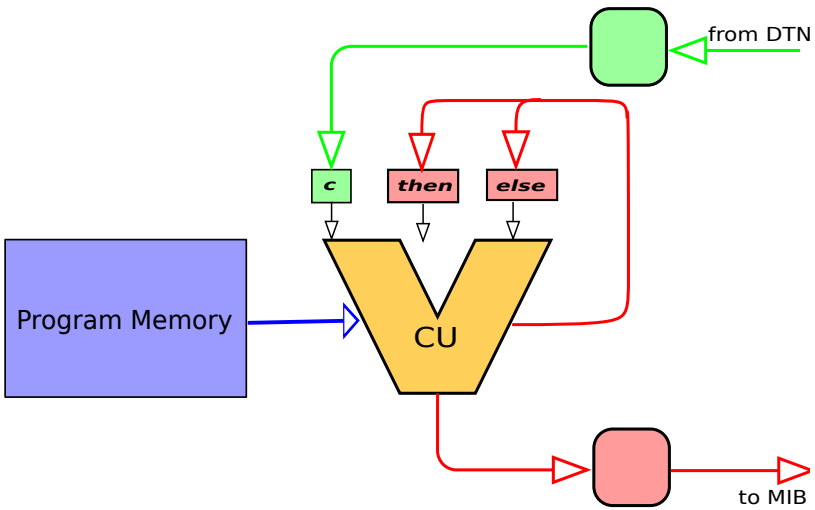


program size

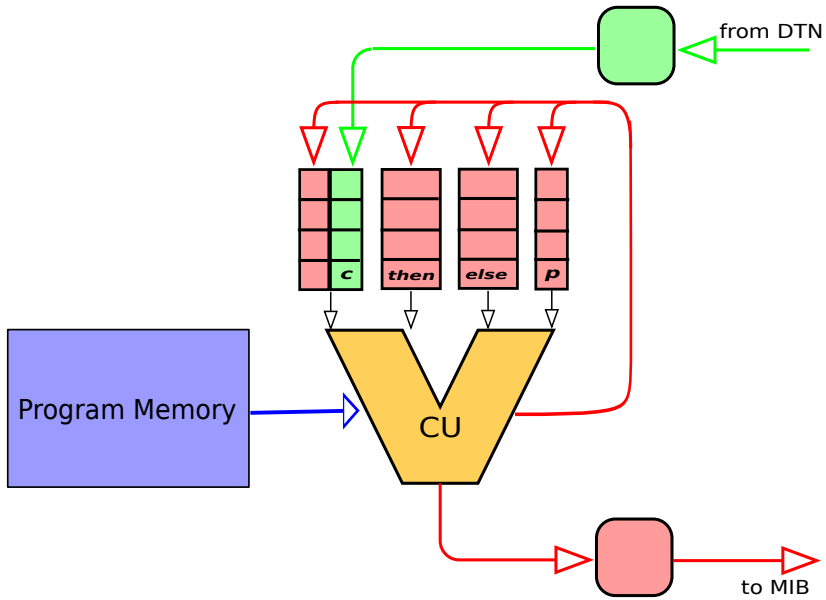


program level

Control unit in SCAD

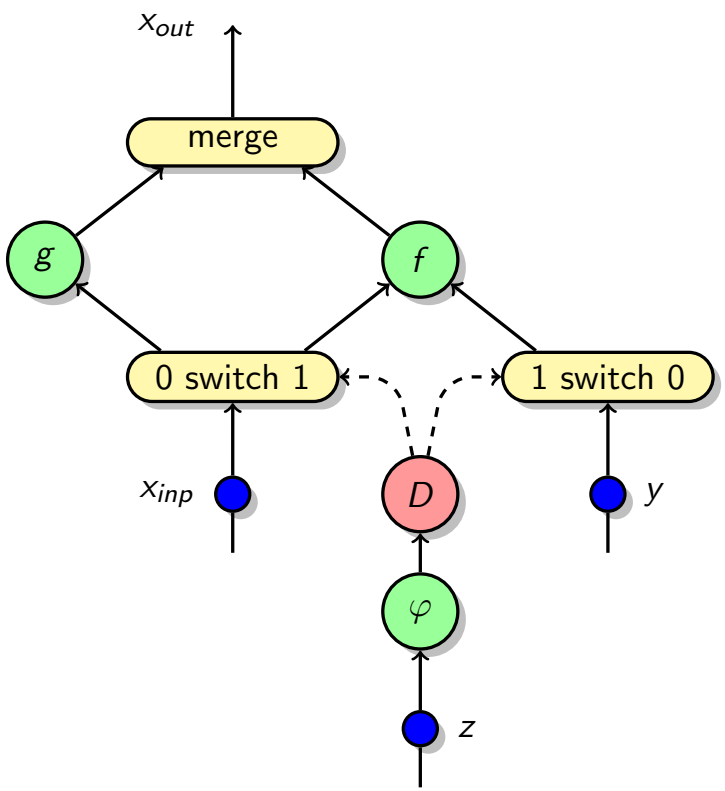


stall on branch

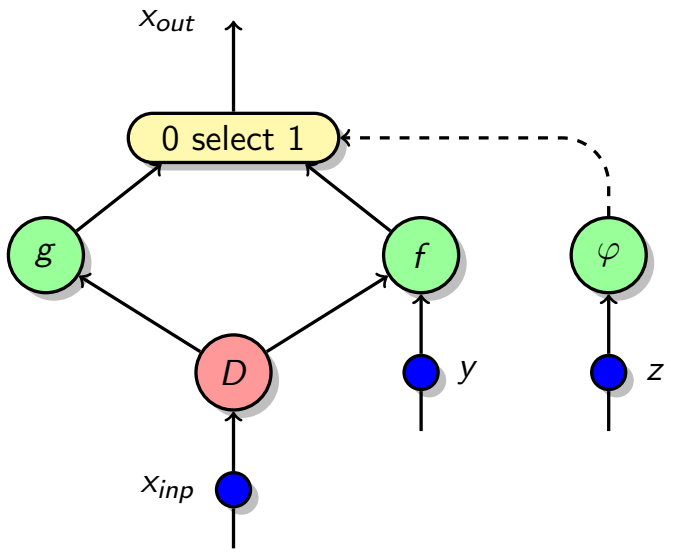


predict branch

Predicated execution in SCAD



sequential ite

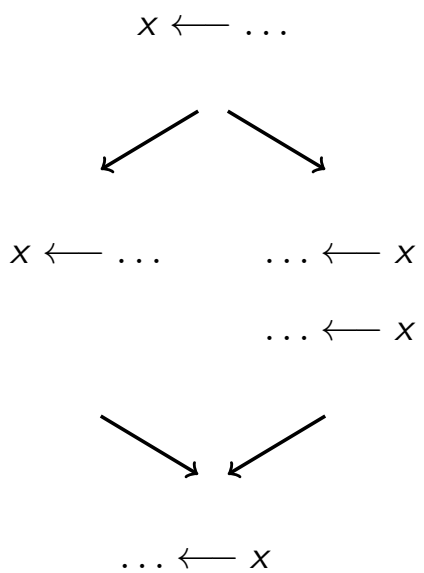


parallel ite

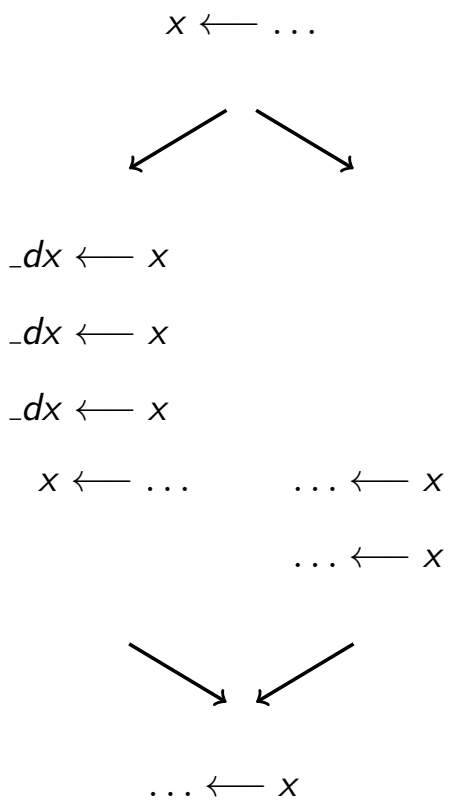
$$\textit{if } \varphi(z) \textit{ then } \{x = f(x, y);\} \textit{ else } \{x = g(x);\}$$

Balancing Copies

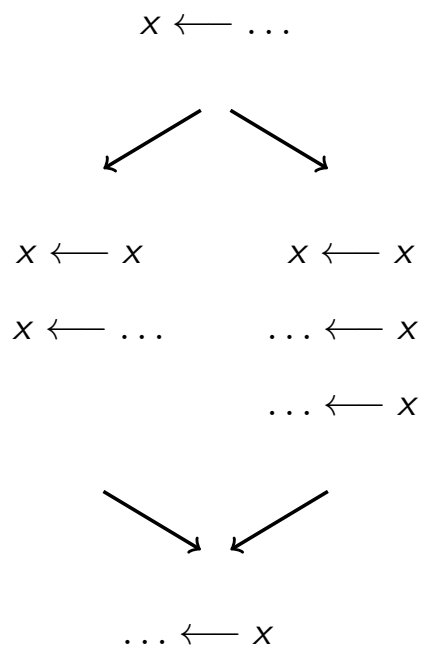
unbalanced program



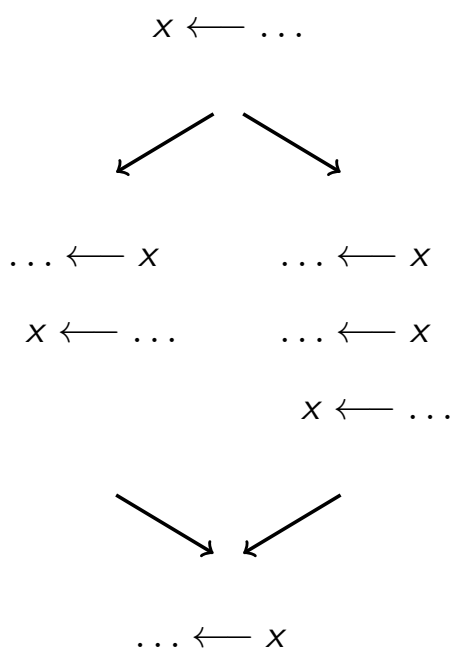
max copies



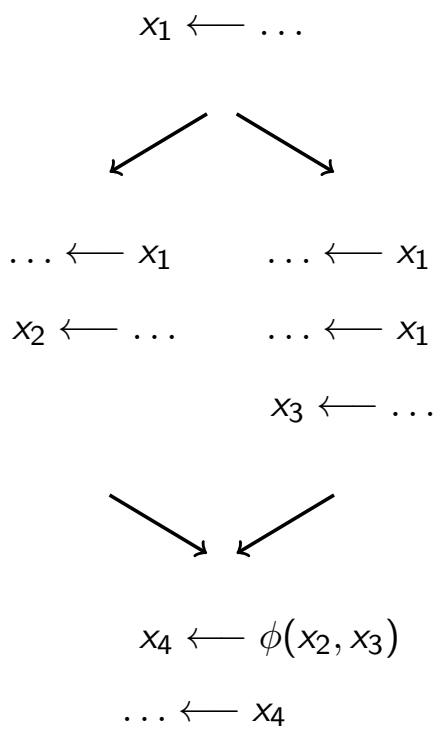
min copies



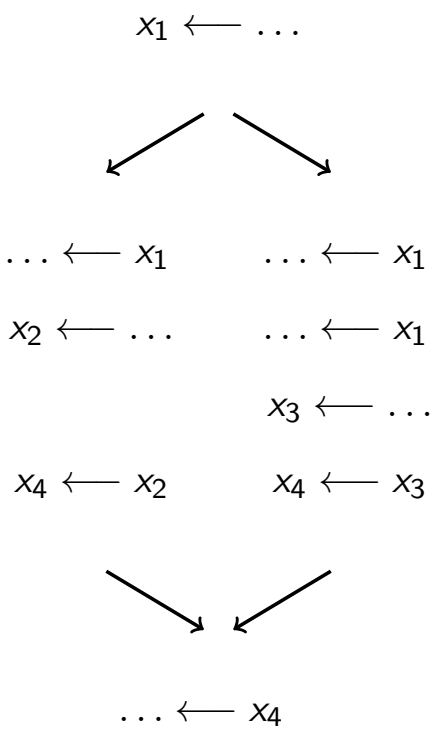
SSA



non-SSA

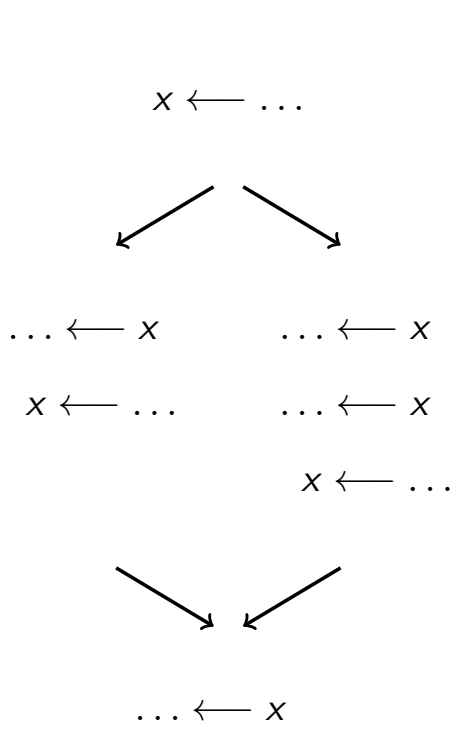


SSA

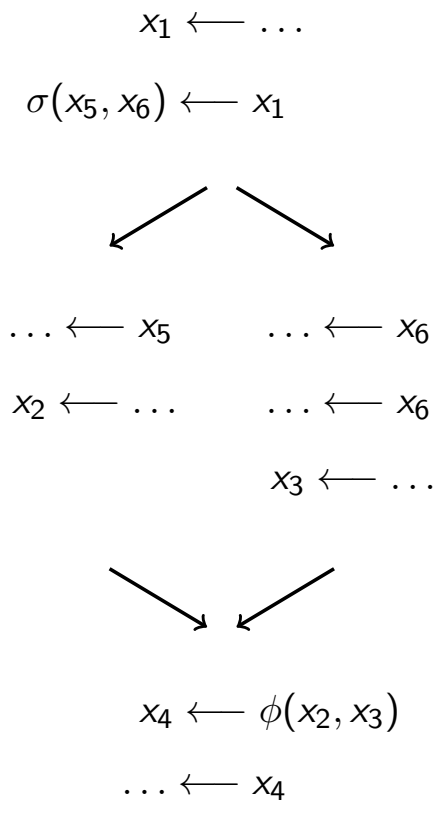


after elimination

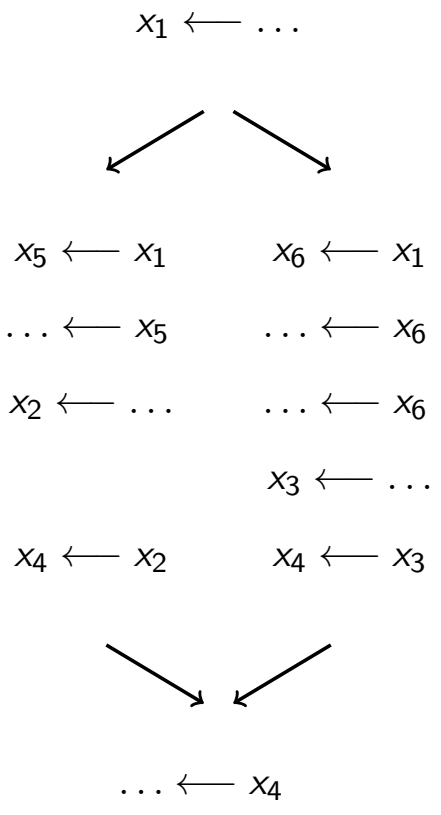
SSI



non-SSI



SSI

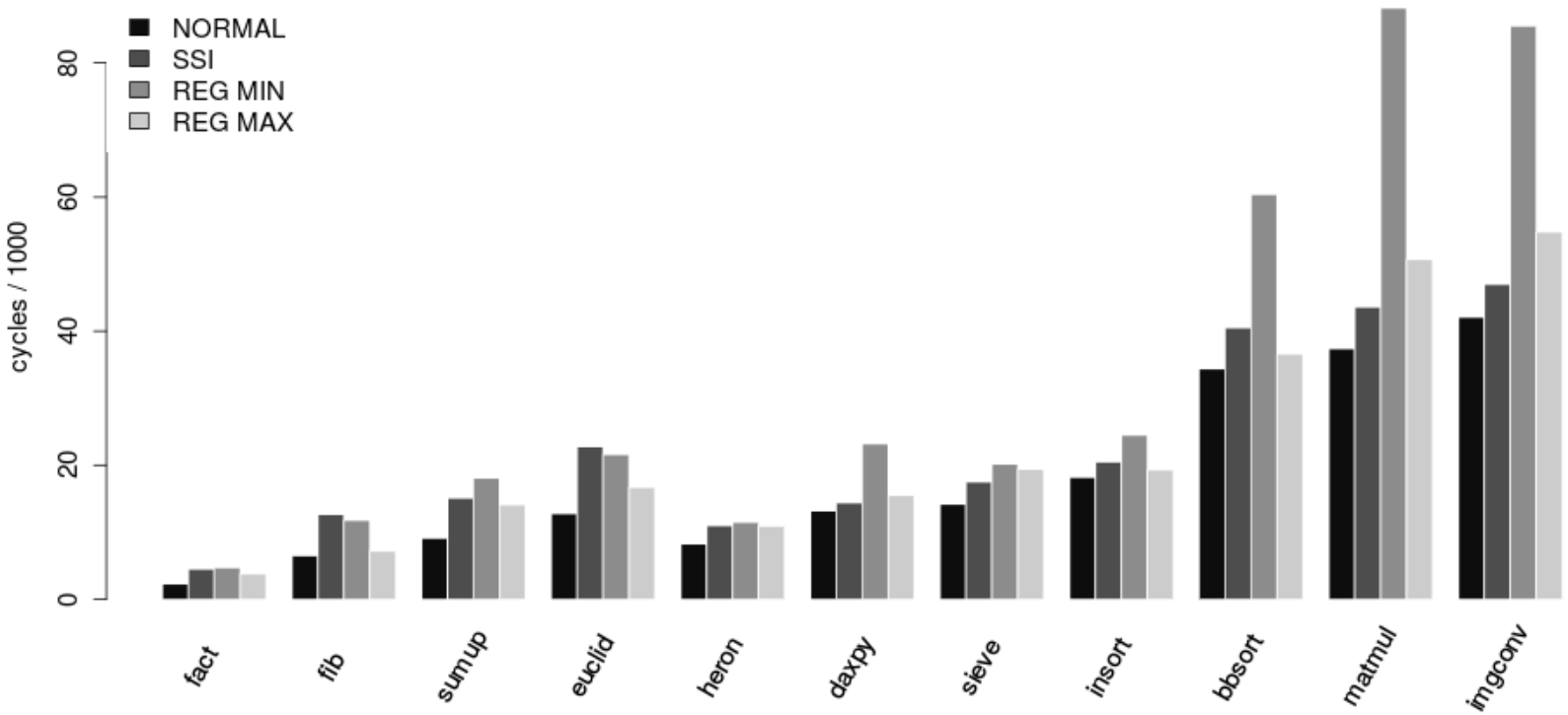


after elimination

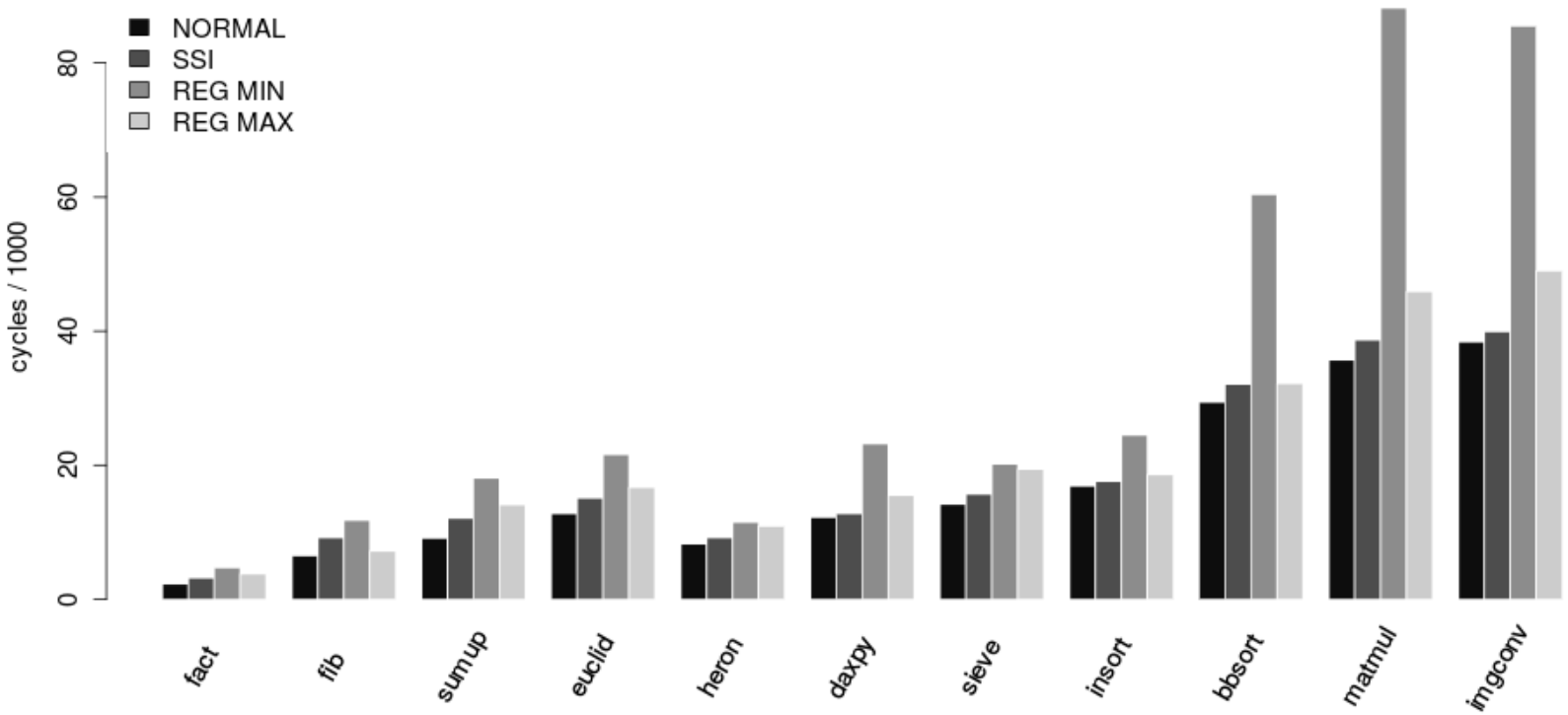
Benchmarks

Program	Label	Input
factorial	fact	$\{1, \ldots, 12\}$
fibonacci	fib	$\{1, \ldots, 20\}$
sumup	sumup	add numbers $\{1, \ldots, 500\}$ in a loop
euclid	euclid	compute gcd of pairs $\{101, 1001\}, \ldots, \{150, 1050\}$
heron	heron	compute square root of first 20 perfect squares
daxpy	daxpy	vector length 100
eratosthenes sieve	sieve	determine prime numbers in $\{1, \ldots, 100\}$
insertion sort	insort	reverse sorted array $[15, \ldots, 1]$
bubble sort	bbsort	reverse sorted array $[15, \ldots, 1]$
matrix multiply	matmul	4×6 and 6×8 matrices
image convolution	imgconv	6×6 image and 3×3 kernel

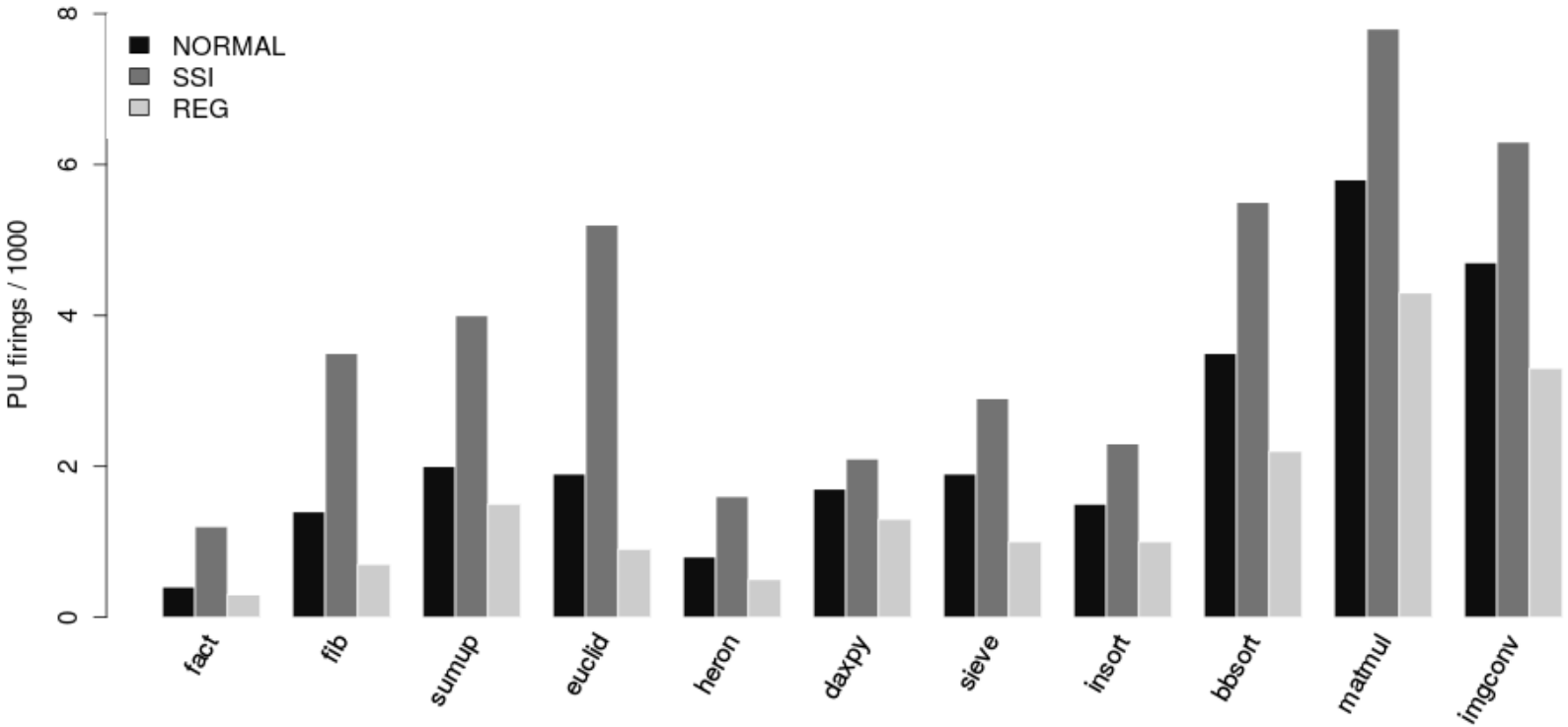
Execution time using minimal buffer size



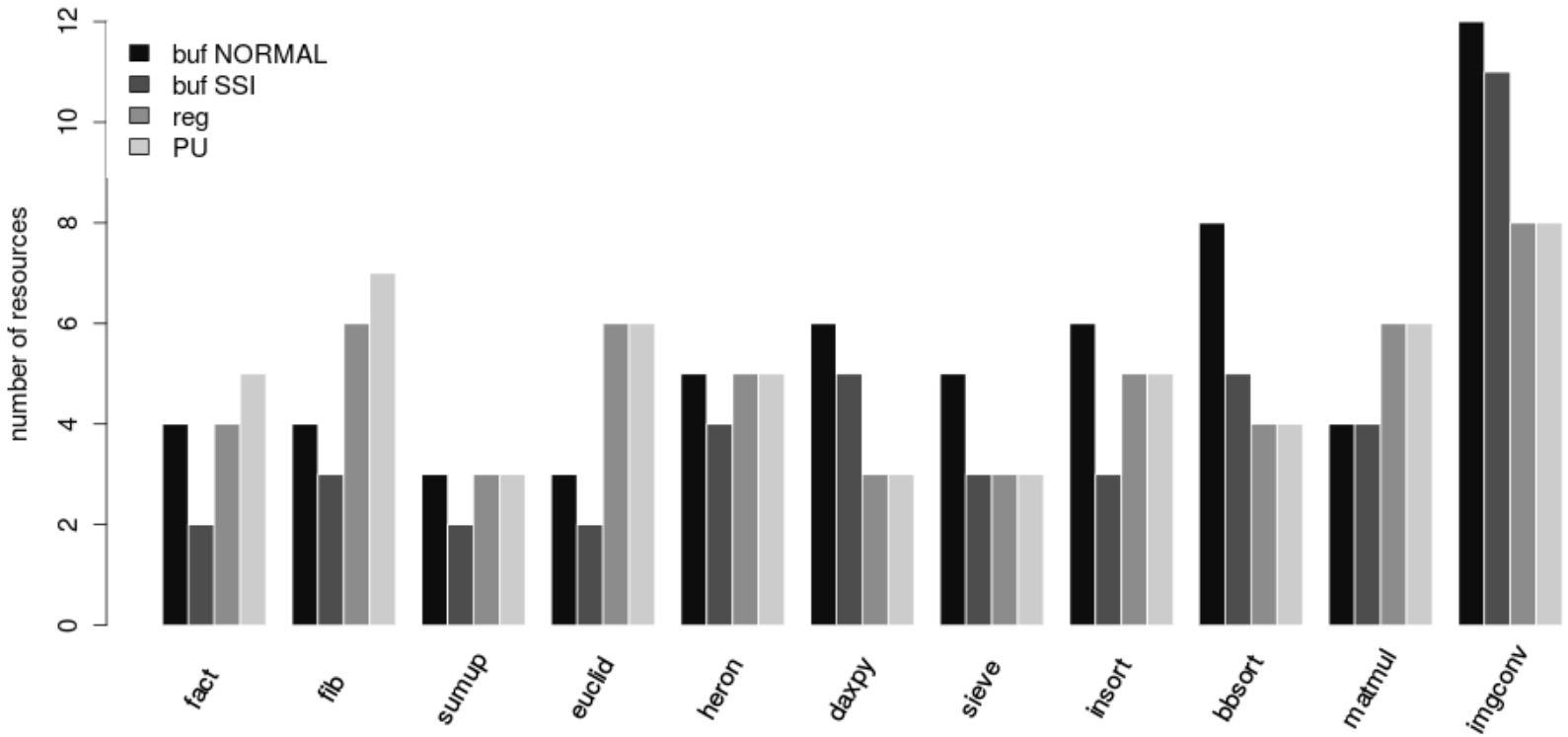
Execution time using maximal buffer size



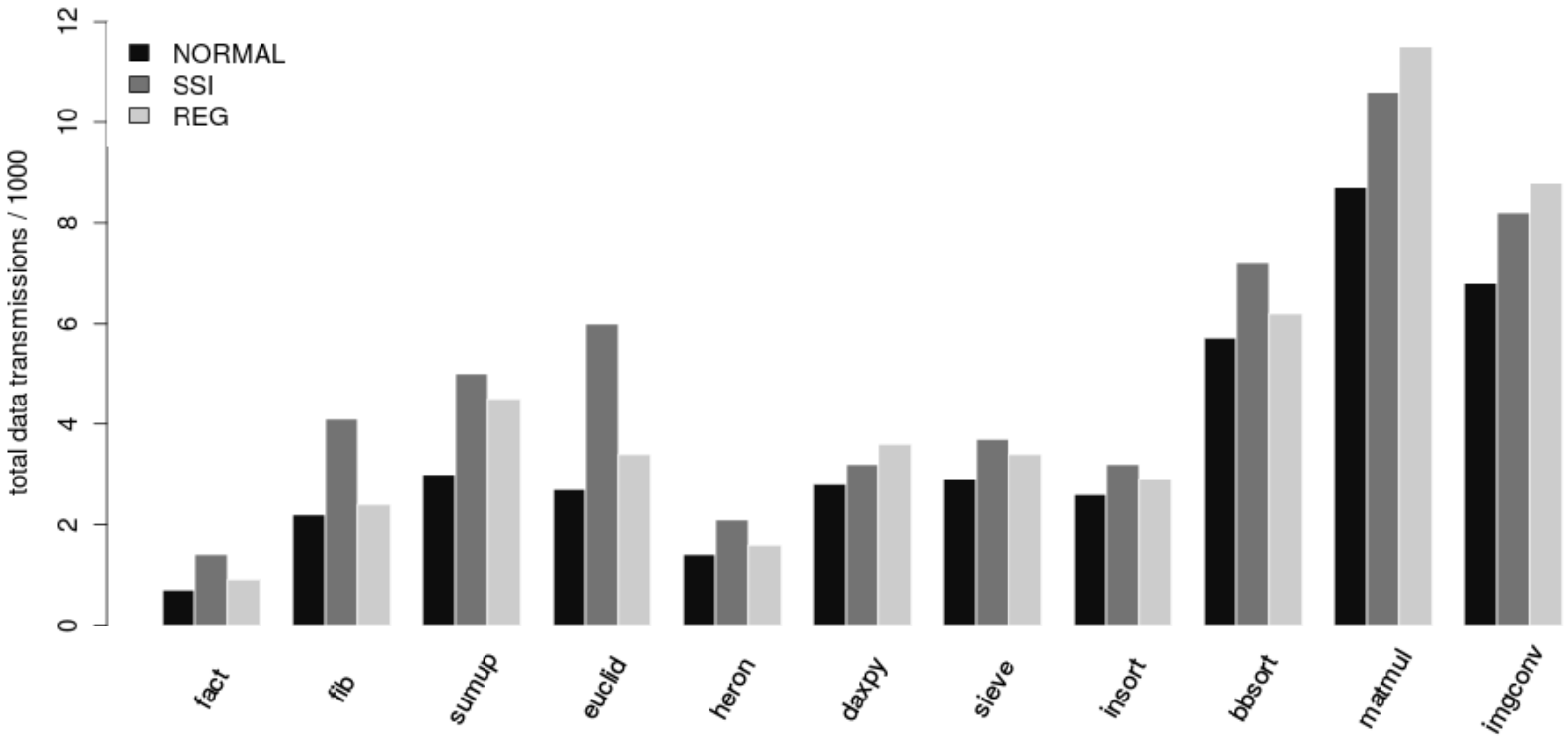
Number of PU firings



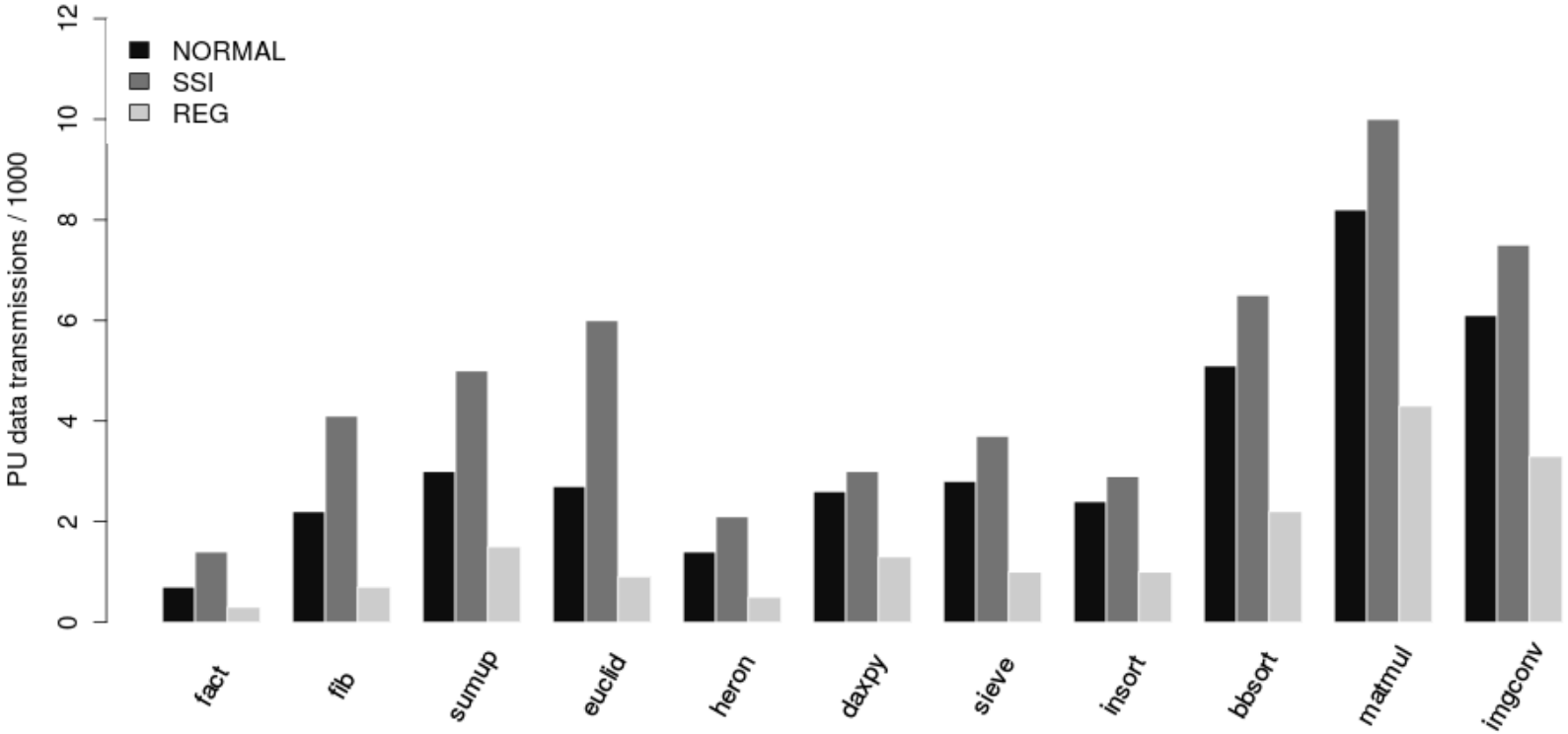
Resource usage



Number of data transmissions



Number of data transmissions by PUs



1-level programs on 1 PU, 1 LSU SCAD

Loop dataflow graph