DATAFLOW CODE GENERATION FOR FPGA

Alexandre Honorat, Mickaël Dardaillon, Jean-François Nezan
INSA Rennes / IETR, CNRS UMR 6164

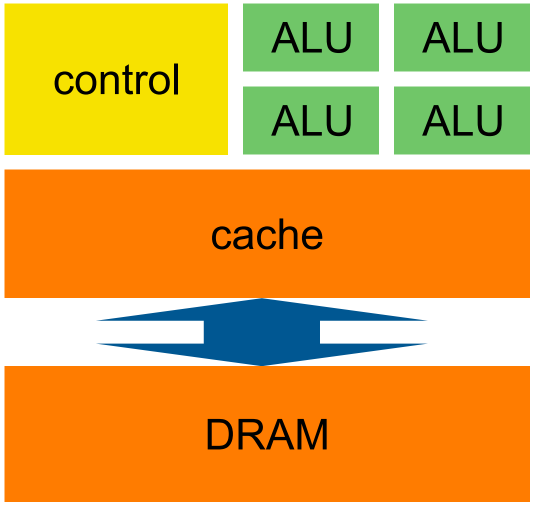
Work in progress

Hardware acceleration for HPC / embedded

- High throughput / low latency
- Low energy

Hardware acceleration for HPC / embedded

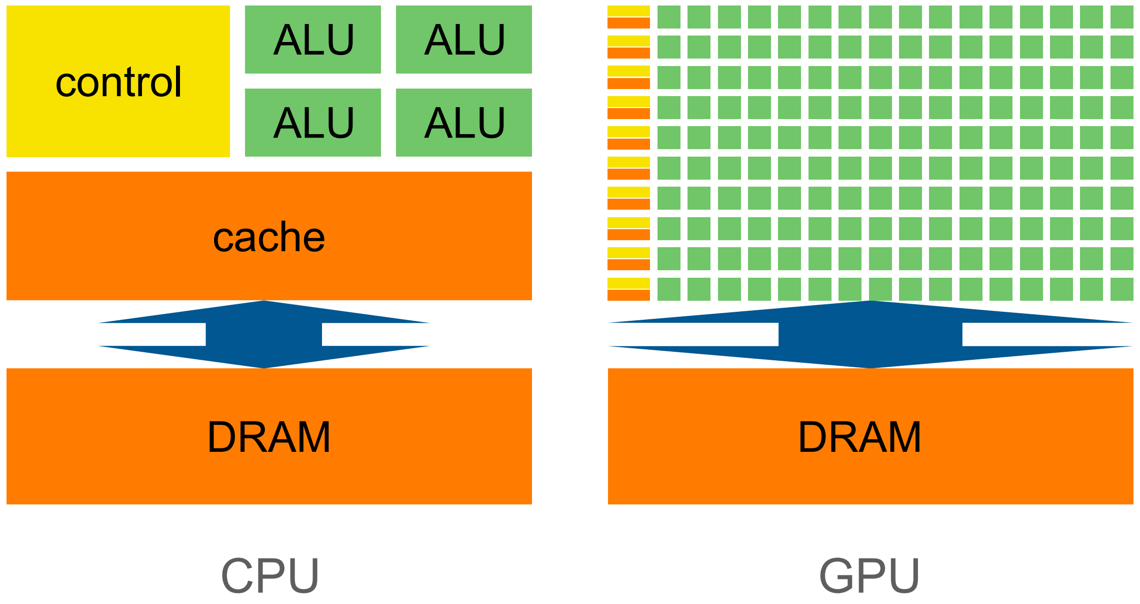
- High throughput / low latency
- Low energy



CPU

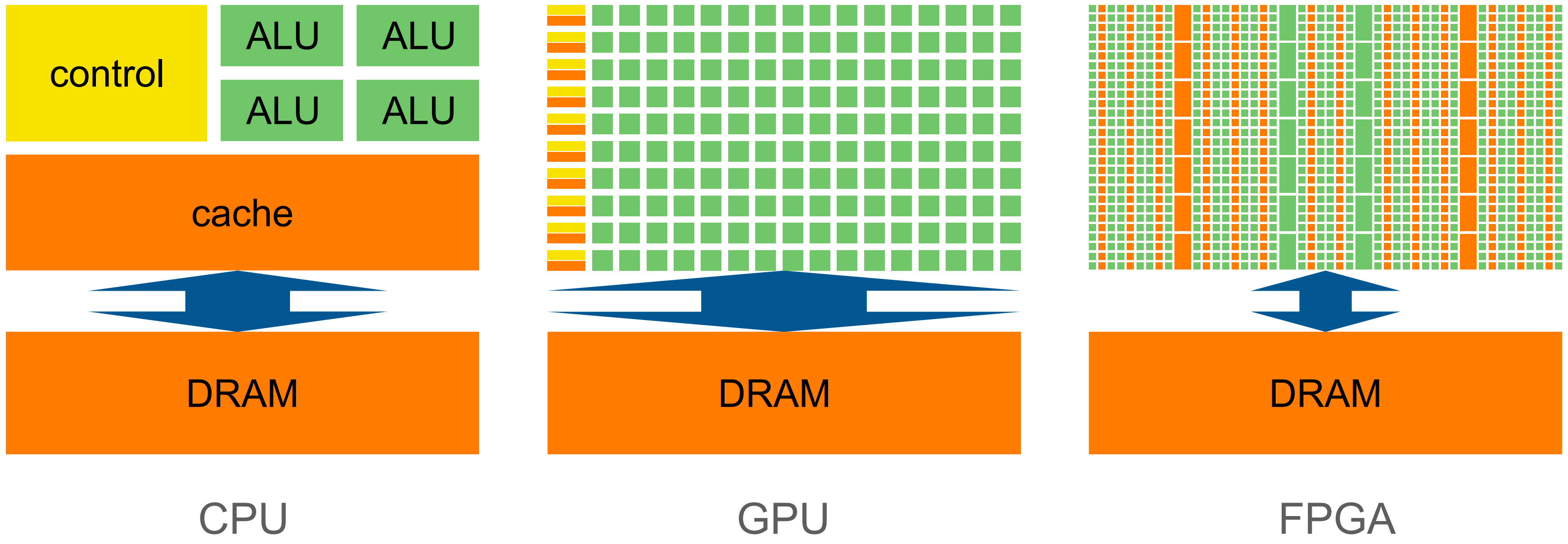
Hardware acceleration for HPC / embedded

- High throughput / low latency
- Low energy



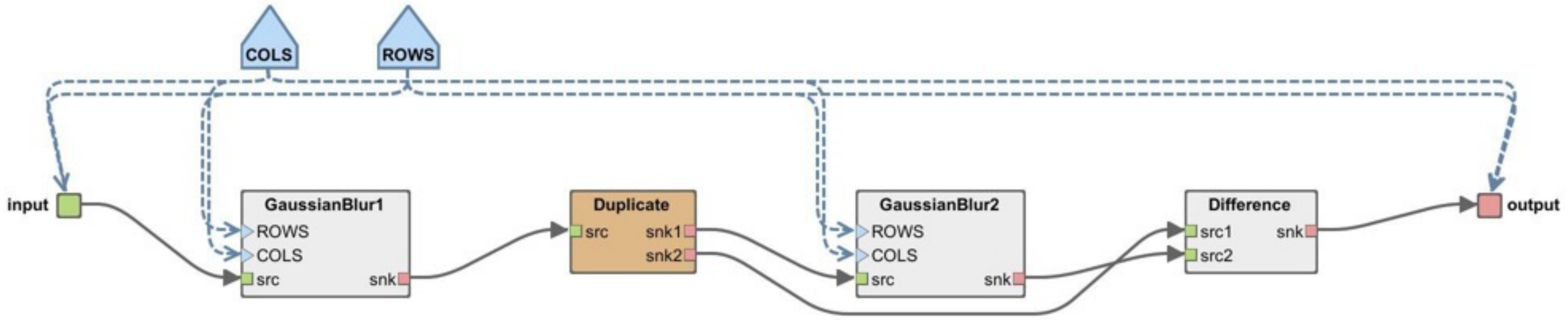
Hardware acceleration for HPC / embedded

- High throughput / low latency
- Low energy



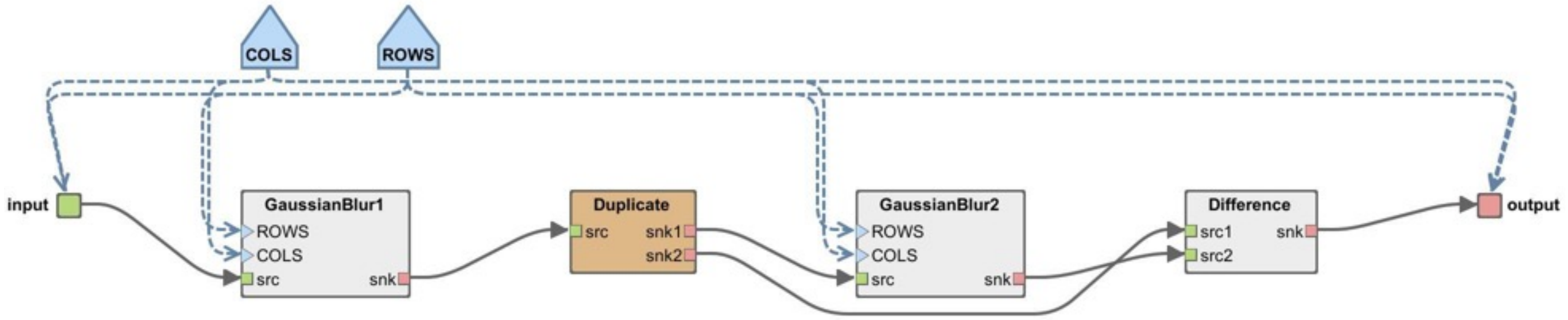
Dataflow programming

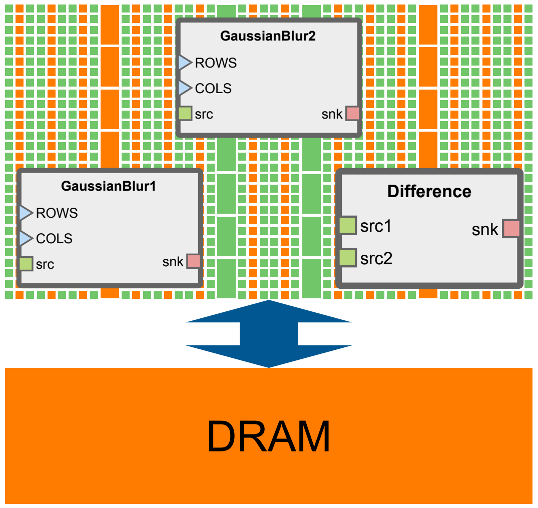
- Natural expression for signal and image processing



Dataflow programming

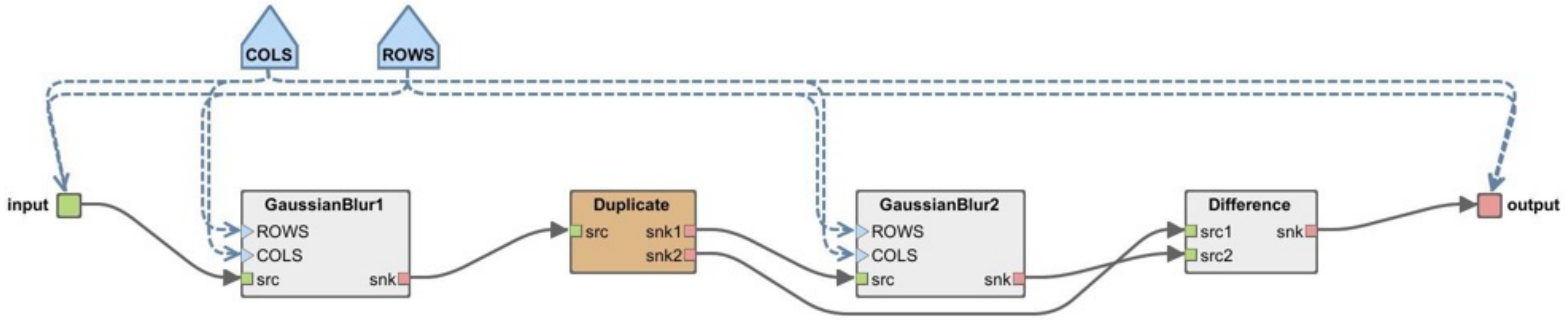
- Natural expression for signal and image processing

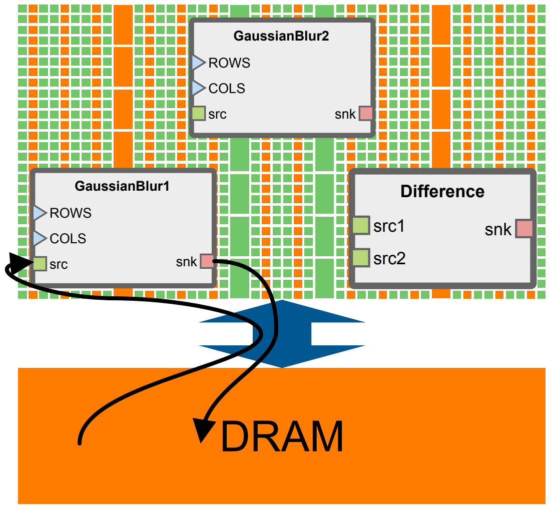




Dataflow programming

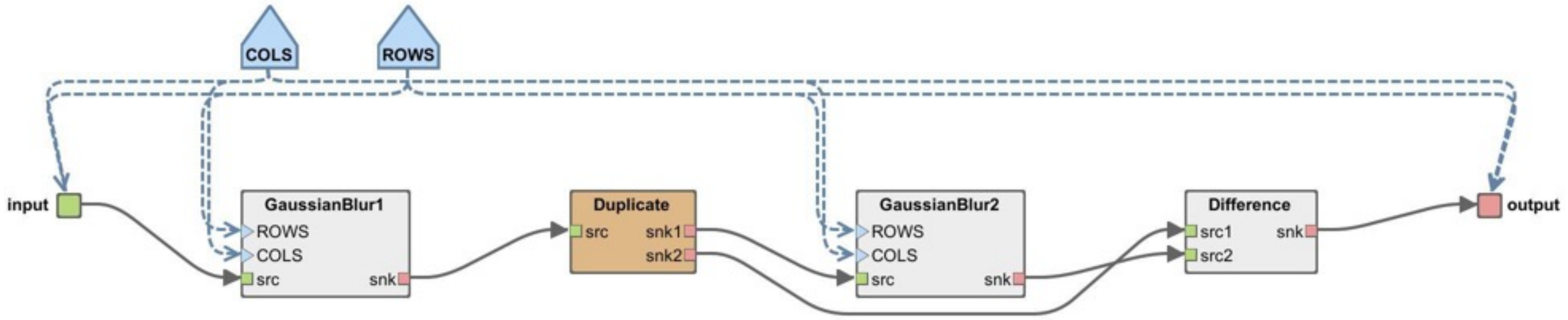
- Natural expression for signal and image processing

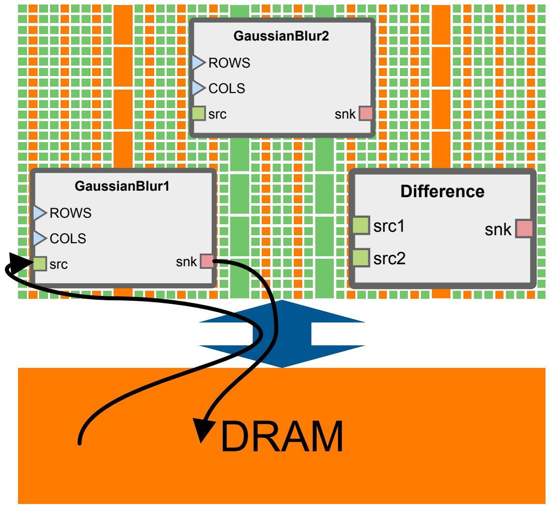




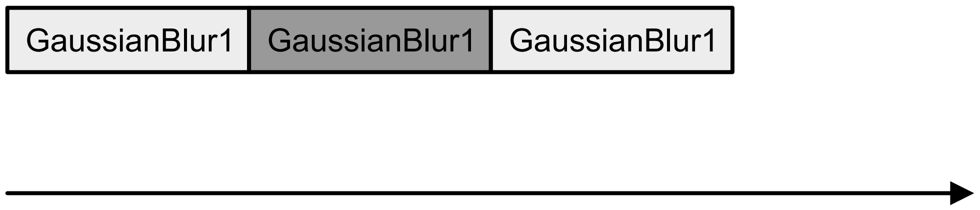
Dataflow programming

- Natural expression for signal and image processing



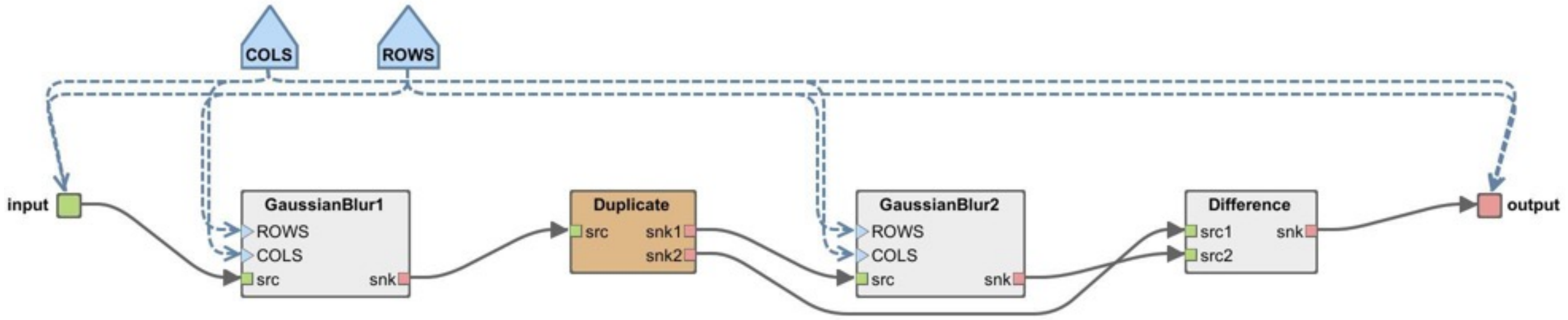


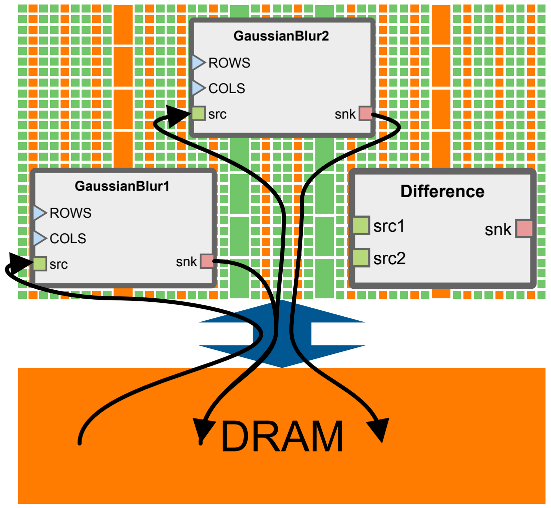
- High throughput



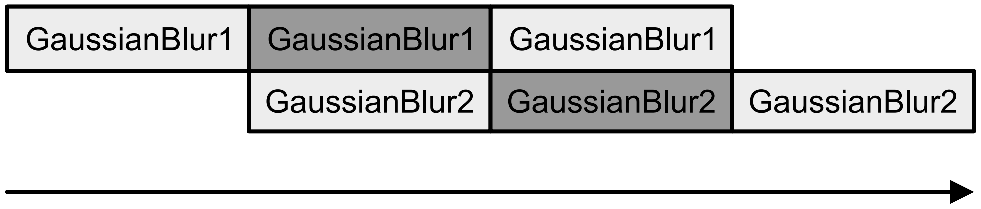
Dataflow programming

- Natural expression for signal and image processing





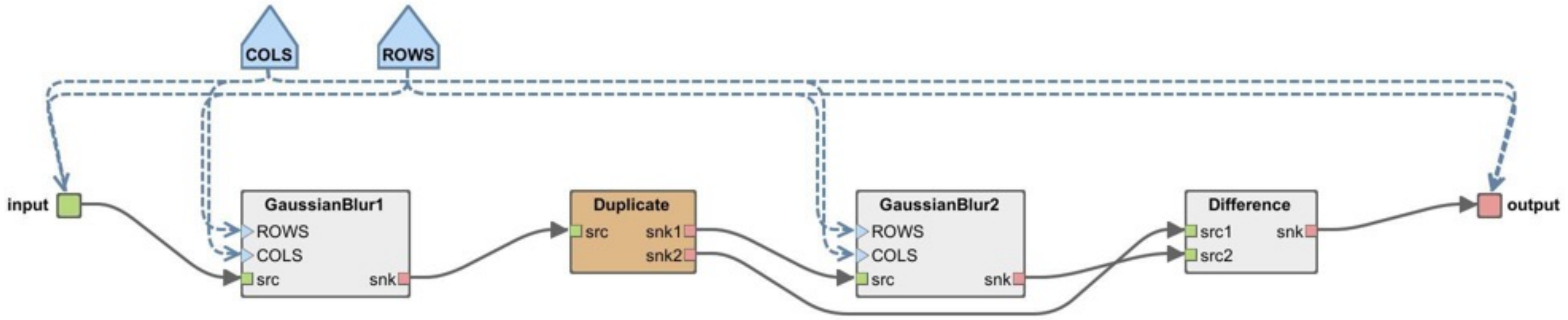
- High throughput
- High memory usage

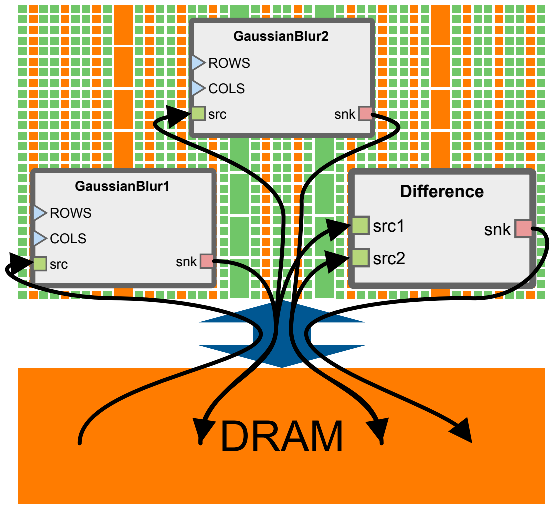


GaussianBlur1	GaussianBlur1	GaussianBlur1	
	GaussianBlur2	GaussianBlur2	GaussianBlur2

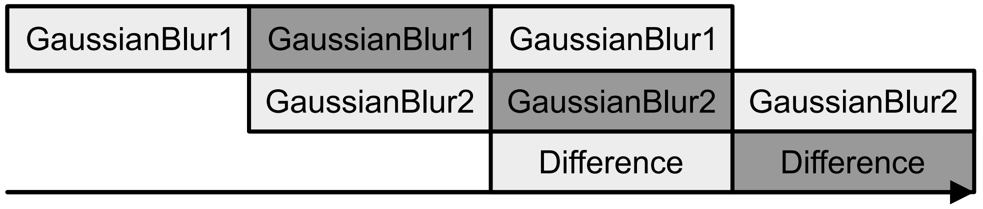
Dataflow programming

- Natural expression for signal and image processing



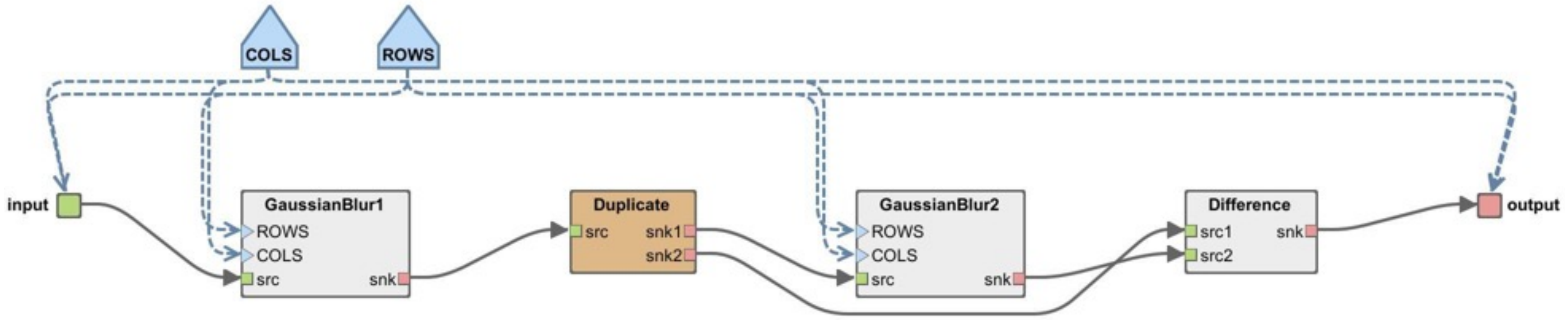


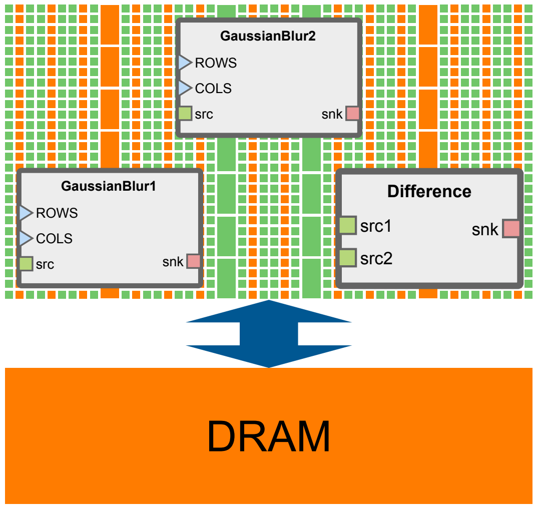
- High throughput
- High memory usage
- High bandwidth usage
- Long latency



Dataflow programming

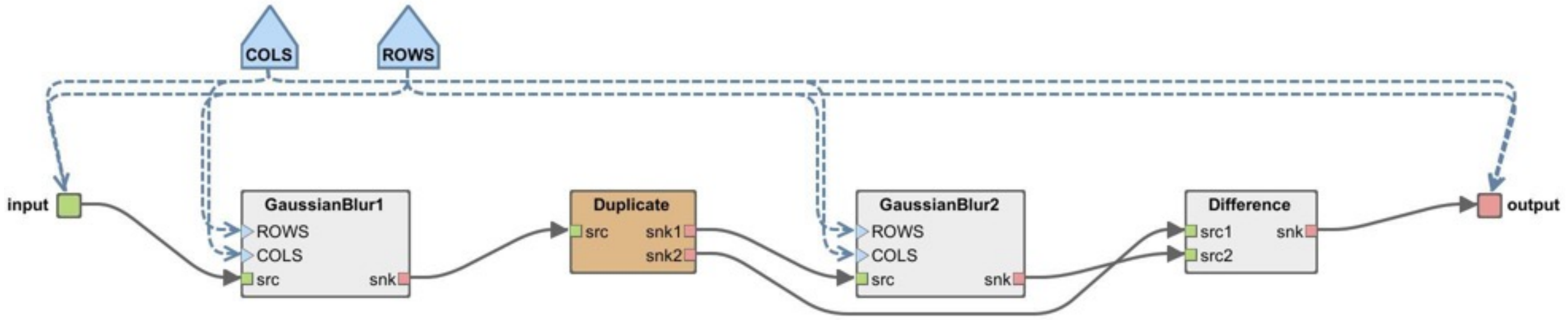
- Natural expression for signal and image processing

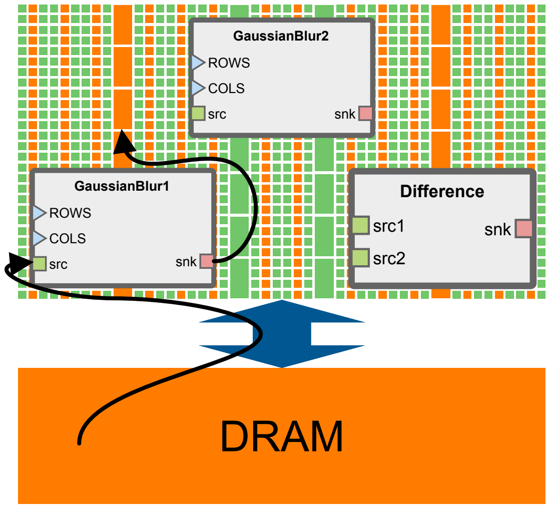




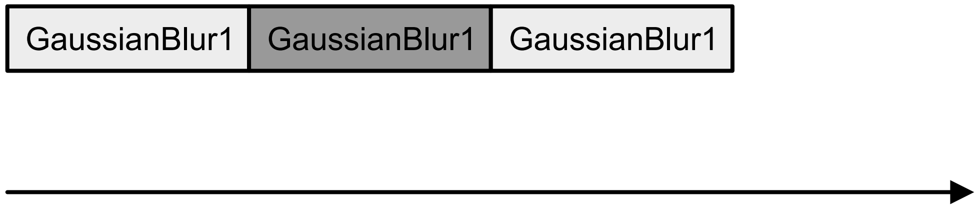
Dataflow programming

- Natural expression for signal and image processing



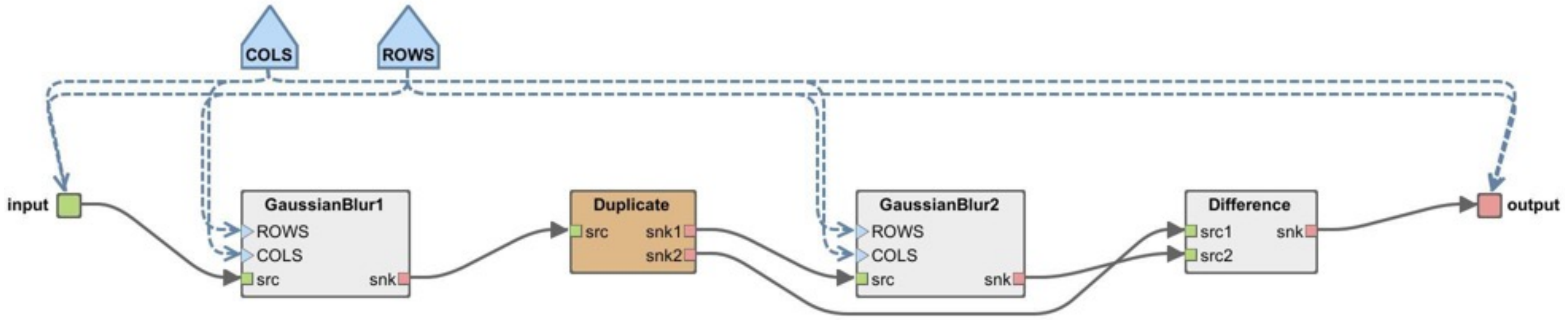


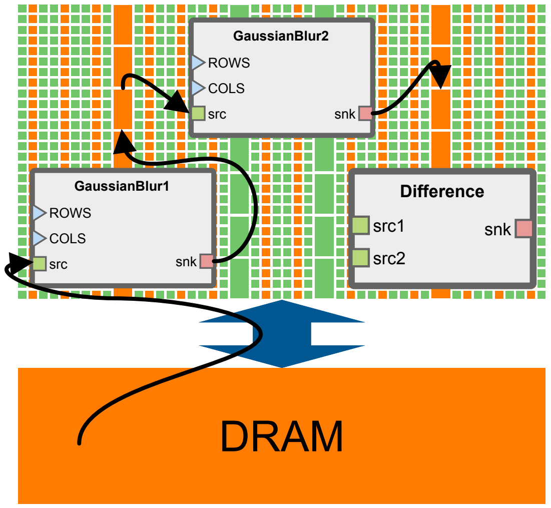
- High throughput



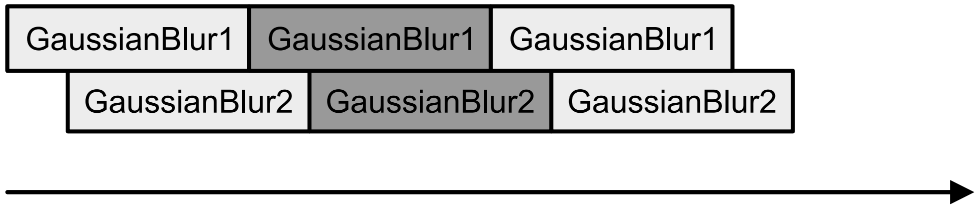
Dataflow programming

- Natural expression for signal and image processing



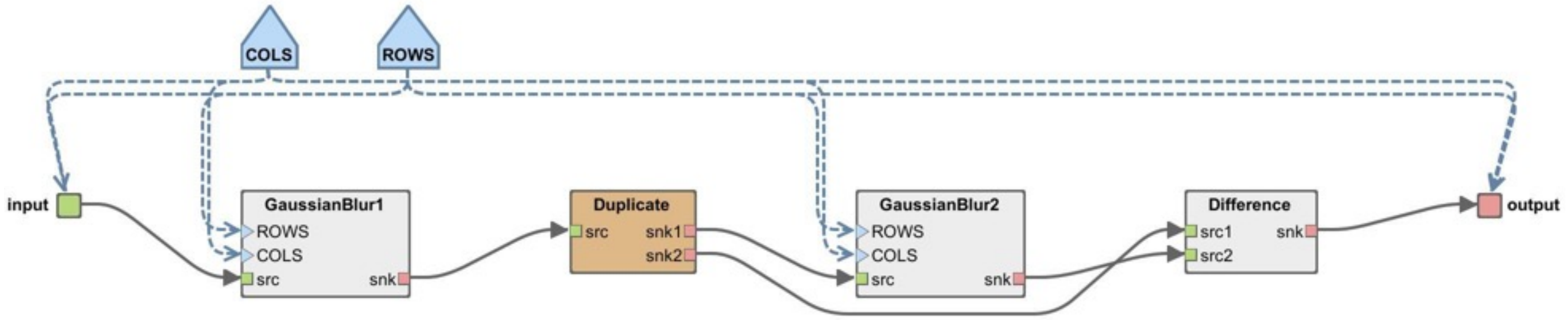


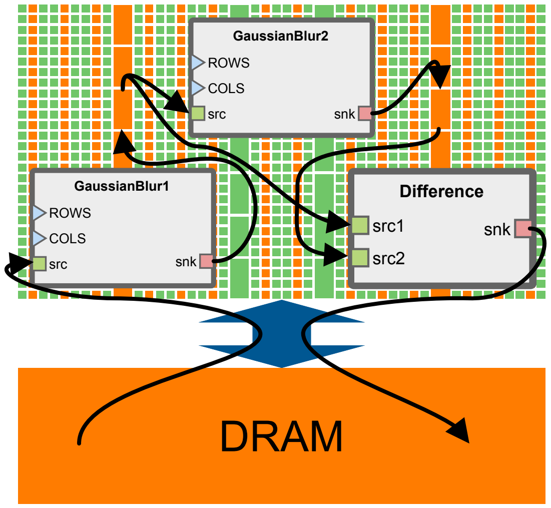
- High throughput
- Low memory usage



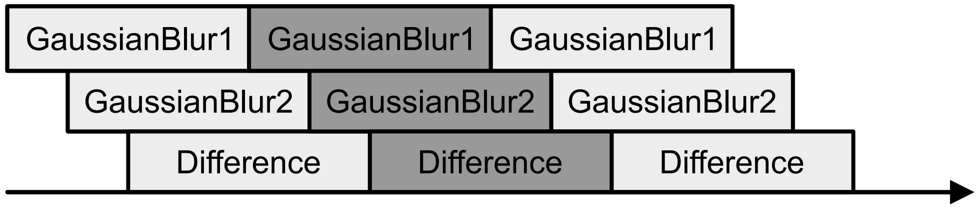
Dataflow programming

- Natural expression for signal and image processing





- High throughput
- Low memory usage
- Low bandwidth usage
- Low latency



Dataflow programming

- Natural expression for signal and image processing
- Pipeline implementation good match for FPGA

Dataflow programming

- Natural expression for signal and image processing
- Pipeline implementation good match for FPGA
- Native support in Vitis HLS

```
void top_graph(hls::stream<int> &input, hls::stream<int> &output) {
// FIFOs
  static hls::stream<int> GaussianBlur1ToDuplicate;
#pragma HLS stream variable=GaussianBlur1ToDuplicate depth=2
  static hls::stream<int> DuplicateToDifference;
#pragma HLS stream variable=DuplicateToDifference depth=N
  // …
  // Kernels
#pragma HLS DATAFLOW
  GaussianBlur1(input, GaussianBlur1ToDuplicate);
  Duplicate(GaussianBlur1ToDuplicate, DuplicateToGaussianBlur2,
DuplicateToDifference);
  GaussianBlur2(DuplicateToGaussianBlur2, GaussianBlur2ToDifference);
  Difference(GaussianBlur2ToDifference, DuplicateToDifference,
output);
}
```

Challenge for design productivity

- Express dataflow at high level
- Optimized hardware implementation

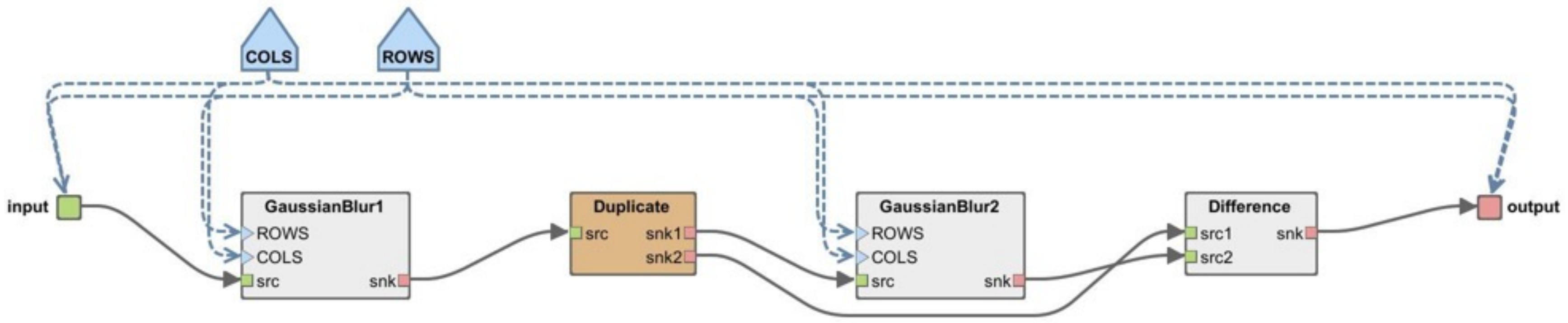
Problem statement

Challenge for design productivity

- Express dataflow at high level
- Optimized hardware implementation

Contributions

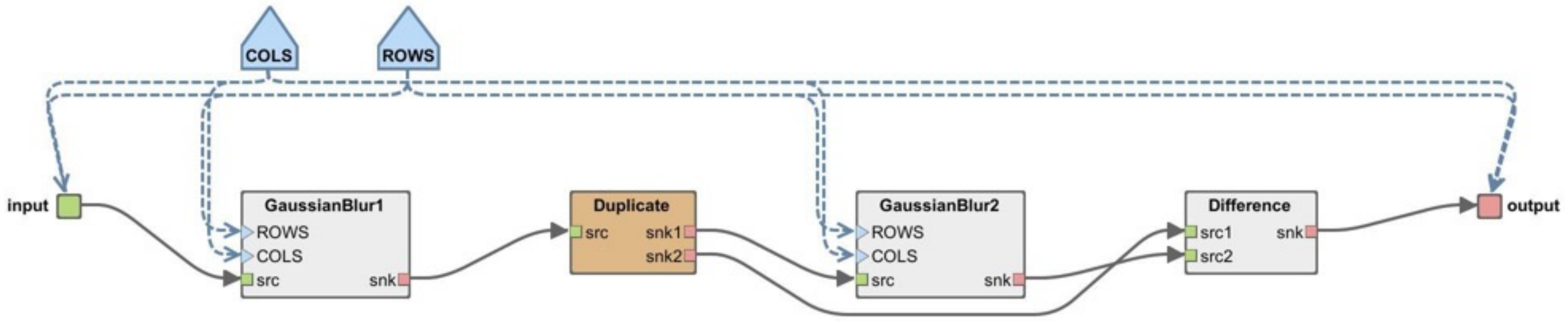
- Dataflow code generation for FPGA using HLS
- Automatic scheduling and buffer sizing
- Open source implementation in PREESM



https://preesm.github.io

DATAFLOW CODE GENERATION FOR FPGA USING HLS

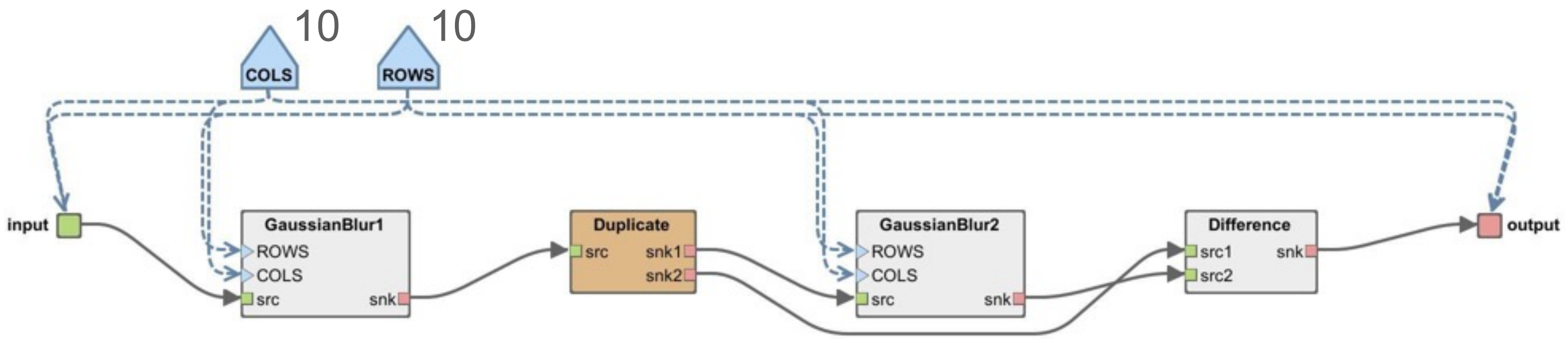
PiSDF graph based on PREESM



Dataflow definition

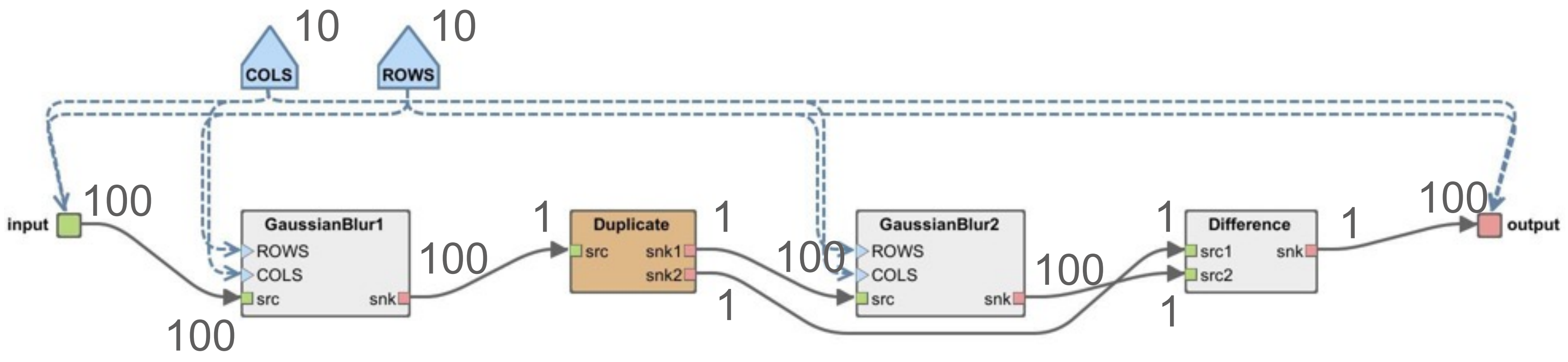
PiSDF graph based on PREESM

- **Parameterized**
- **Interfaced**
- **Synchronous Dataflow**



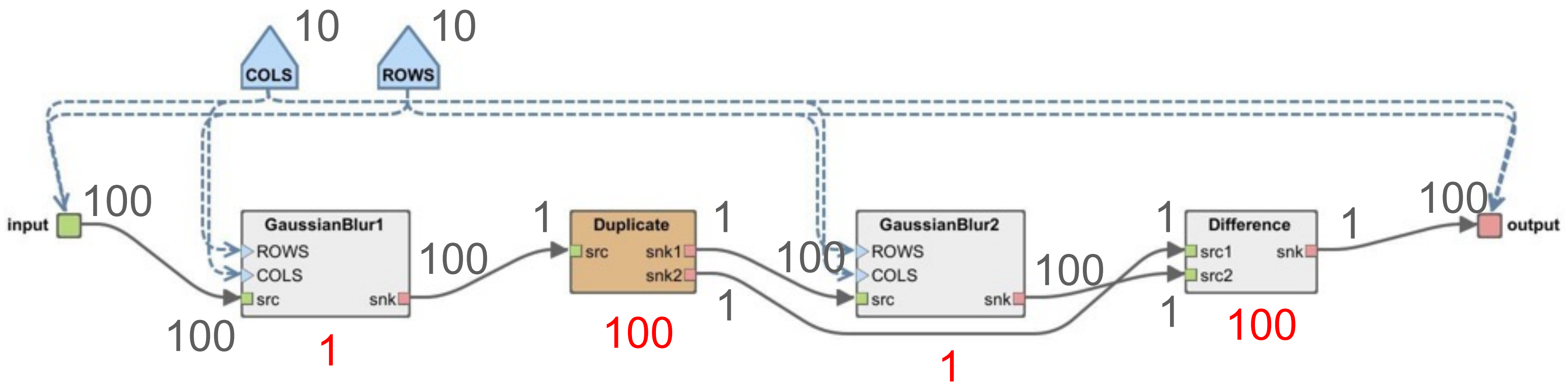
PiSDF graph based on PREESM

- Parameterized
- Interfaced
- Synchronous Dataflow
    - Multirate

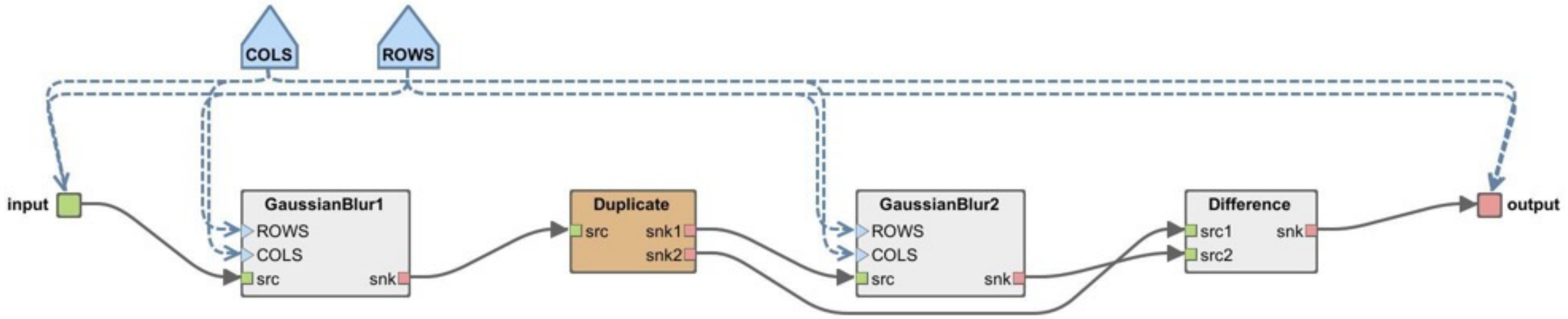


PiSDF graph based on PREESM

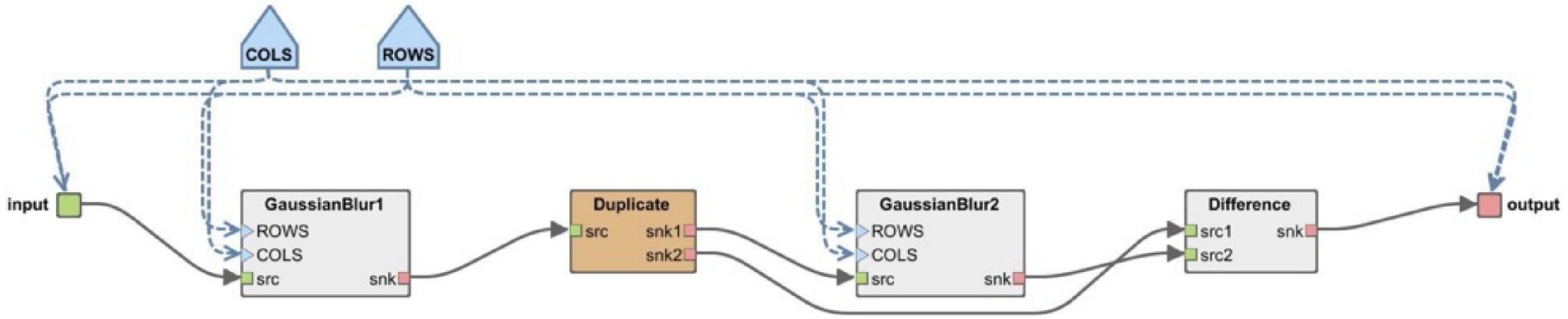
- **Parameterized**
- **Interfaced**
- **Synchronous Dataflow**
  - Multirate
  - Repetition factor
  - Deadlock free
  - Automatic scheduling and buffer sizing



Dataflow using HLS



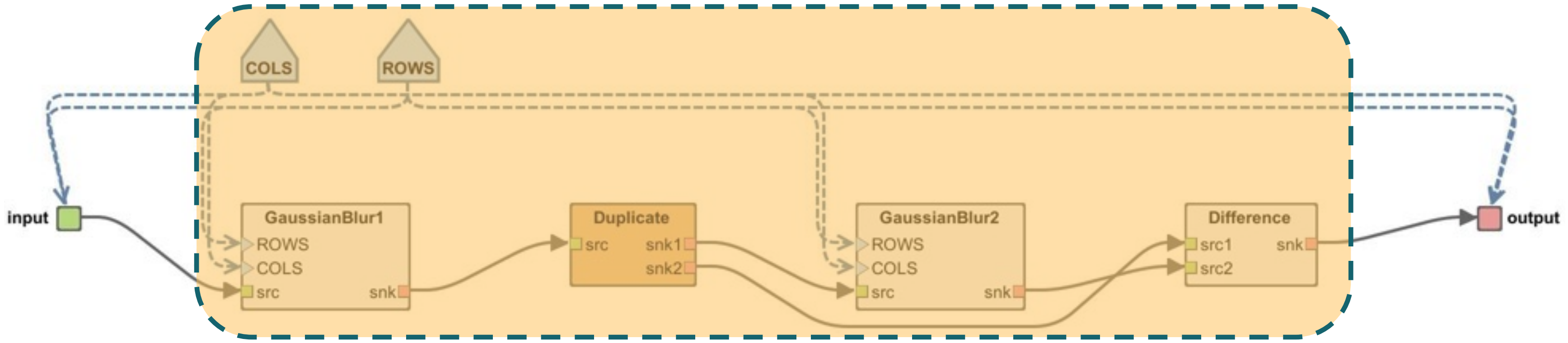
Dataflow using HLS



User provided

- **Kernels**
  - HLS
  - FIFO interface (stream<T>)

Dataflow using HLS



Kernels + Graph

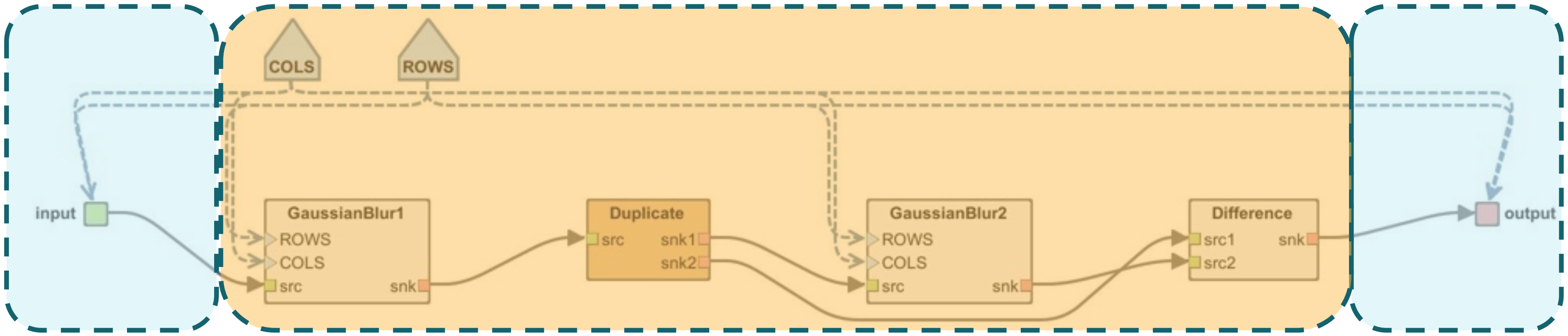
User provided

- **Kernels**
  - HLS
  - FIFO interface (stream<T>)

Code generation

- **Graph**
  - Multirate
  - Cyclic
  - Self-scheduled ASAP

Dataflow using HLS



Input

Kernels + Graph

Output

User provided

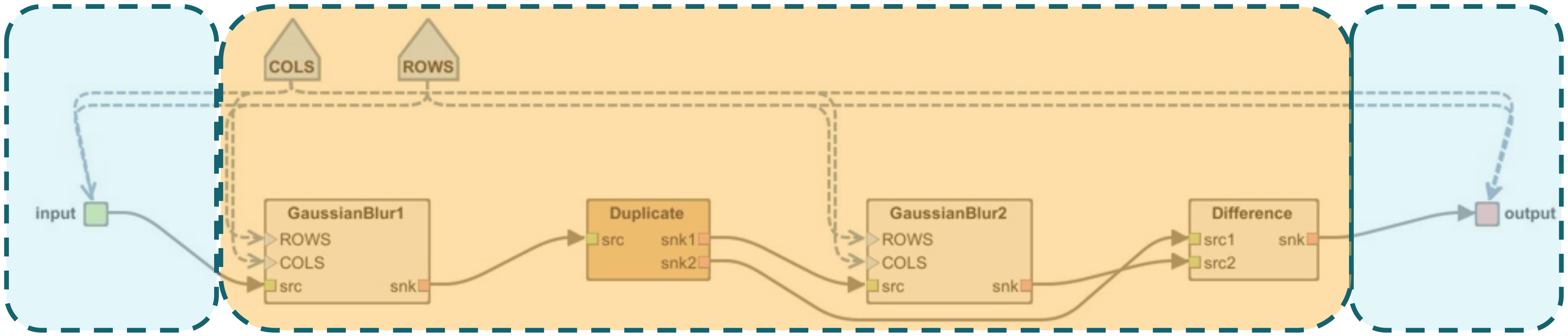
- **Kernels**
  - HLS
  - FIFO interface (stream<T>)

- **Input / output**
  - Array interface (T*)
  - Batch transfer from/to RAM
  - Scheduled by host

Code generation

- **Graph**
  - Multirate
  - Cyclic
  - Self-scheduled ASAP

Dataflow using HLS



Input   Kernels + Graph   Output

User provided

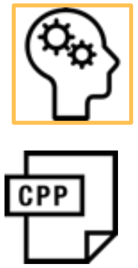
- **Kernels**
  - HLS
  - FIFO interface (stream<T>)

Code generation

- **Graph**
  - Multirate
  - Cyclic
  - Self-scheduled ASAP

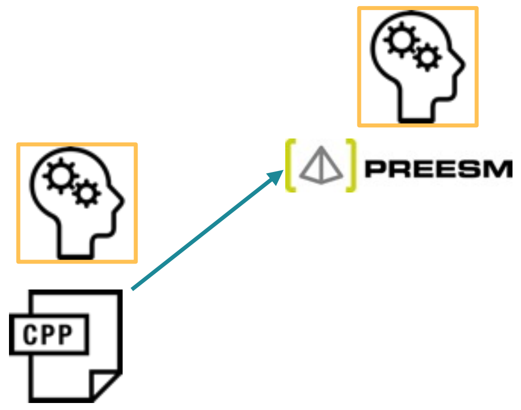
- **Input / output**
  - Array interface (T*)
  - Batch transfer from/to RAM
  - Scheduled by host
- **Host code**
  - OpenCL
  - PYNQ
  - Bare metal
  - Testbench

Compilation flow



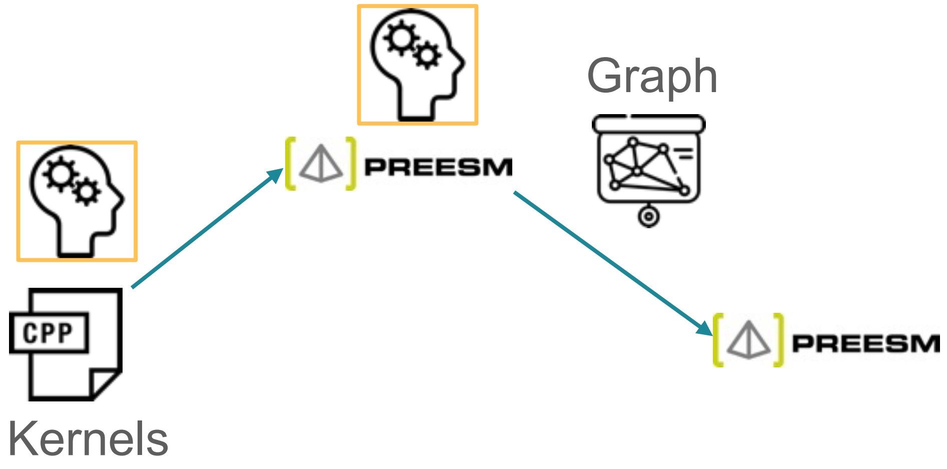
Kernels

Compilation flow

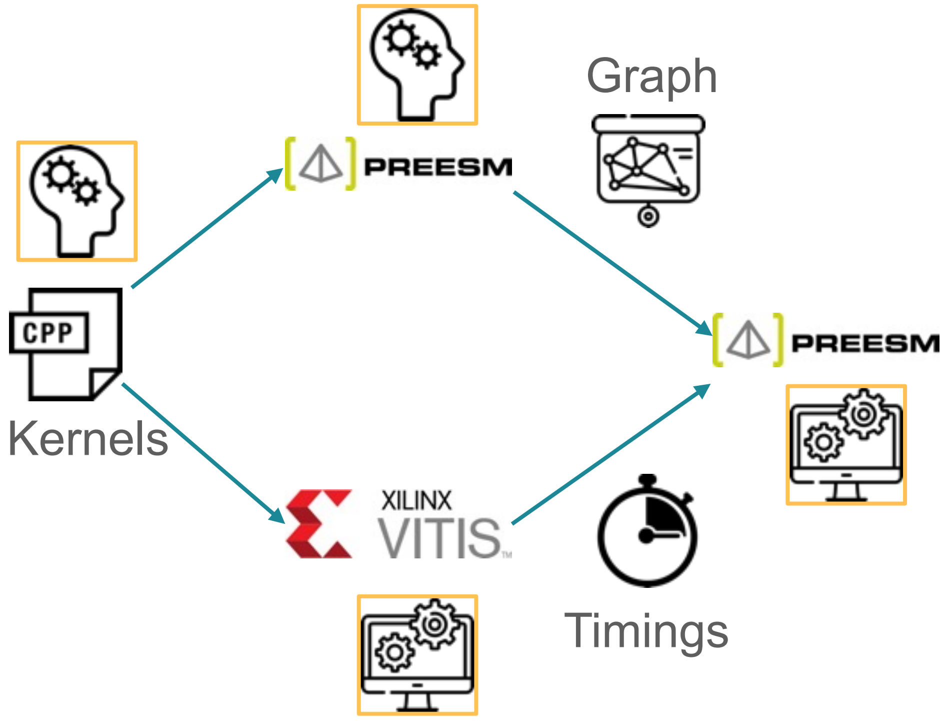


Kernels

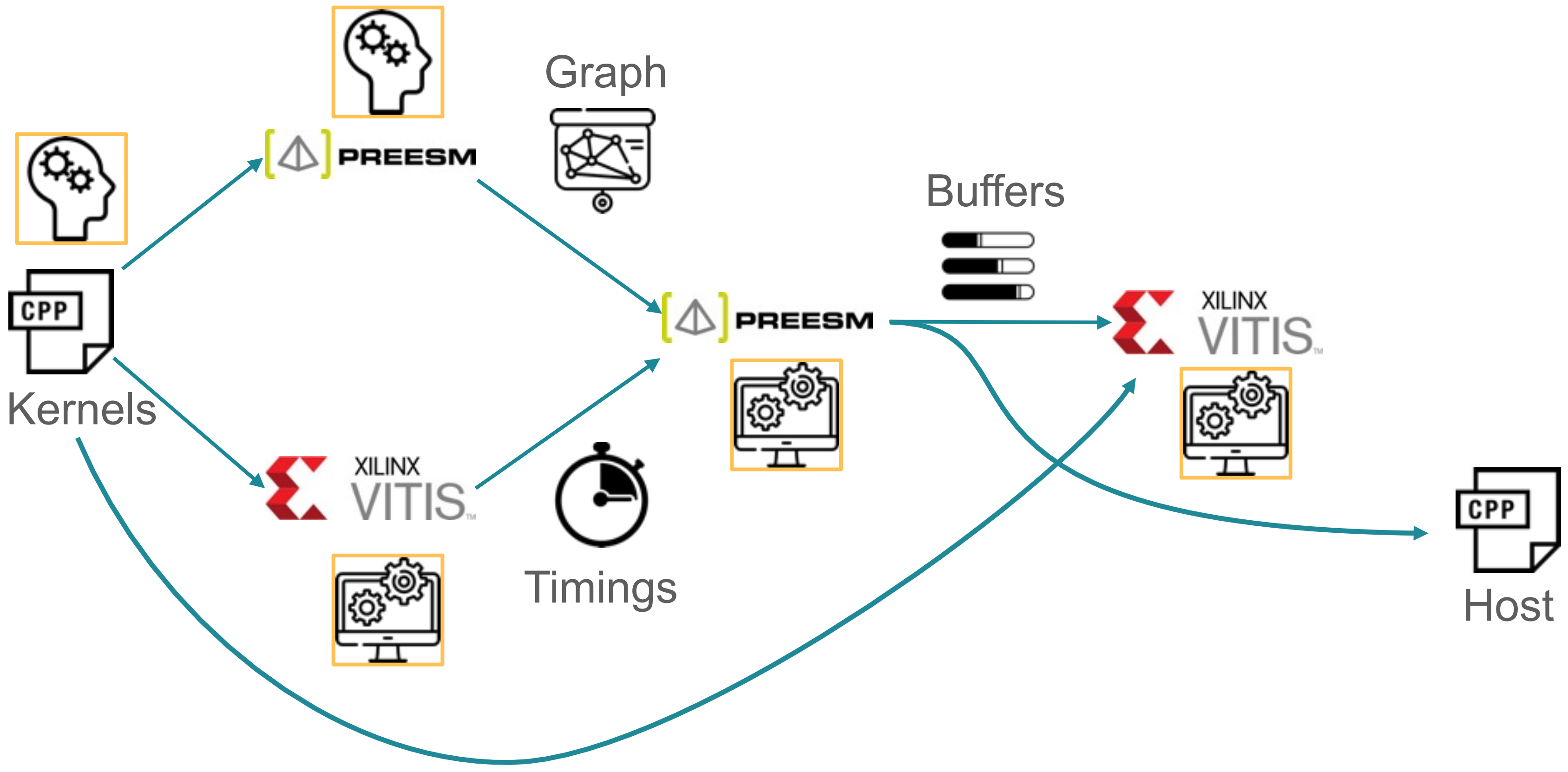
Compilation flow



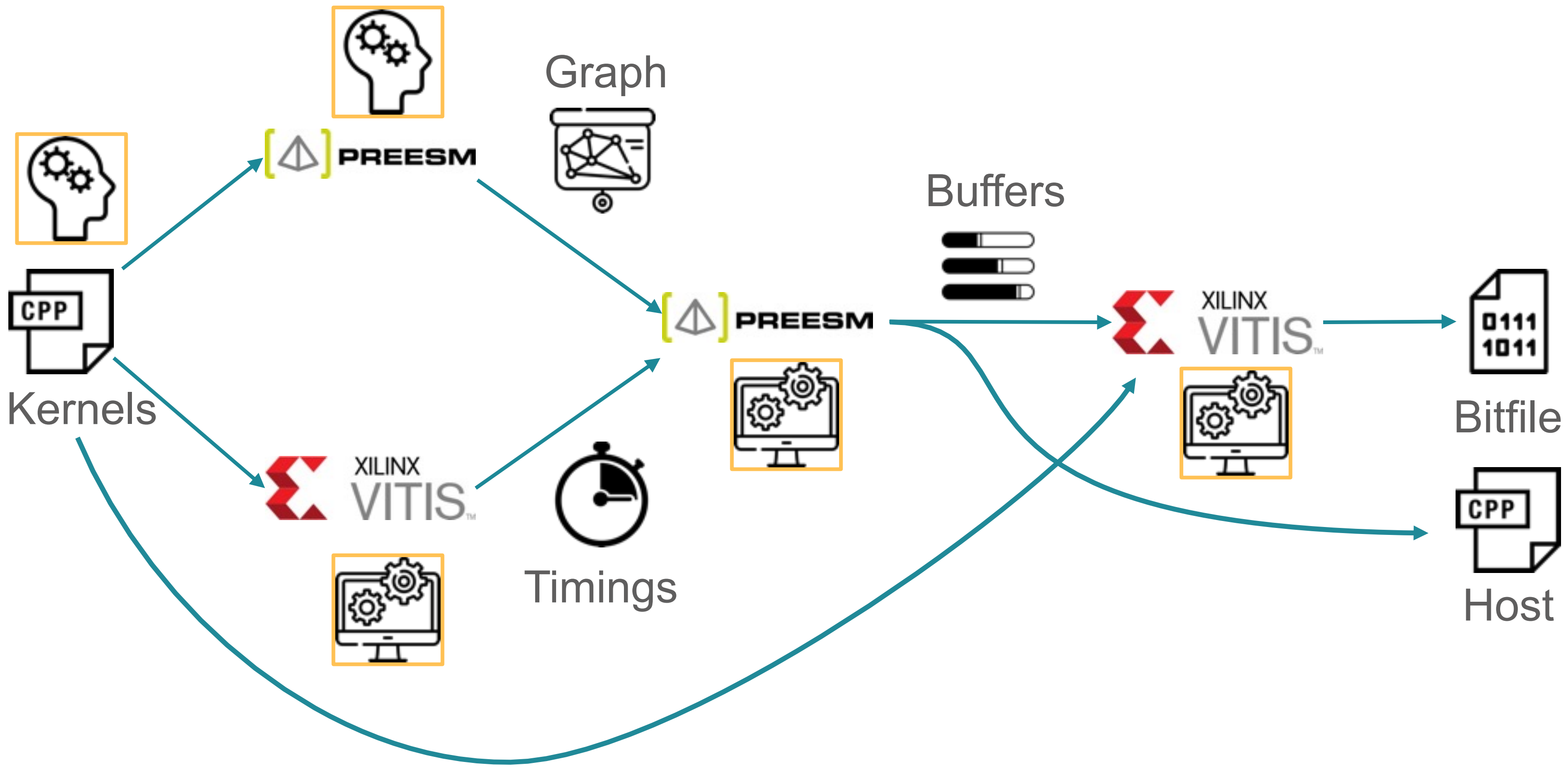
Compilation flow



Compilation flow

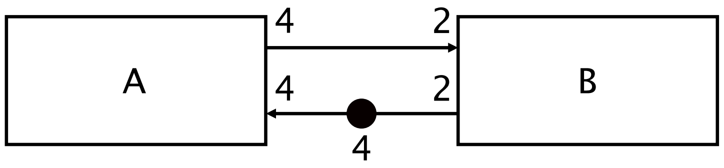


Compilation flow



AUTOMATIC SCHEDULING AND BUFFER SIZING

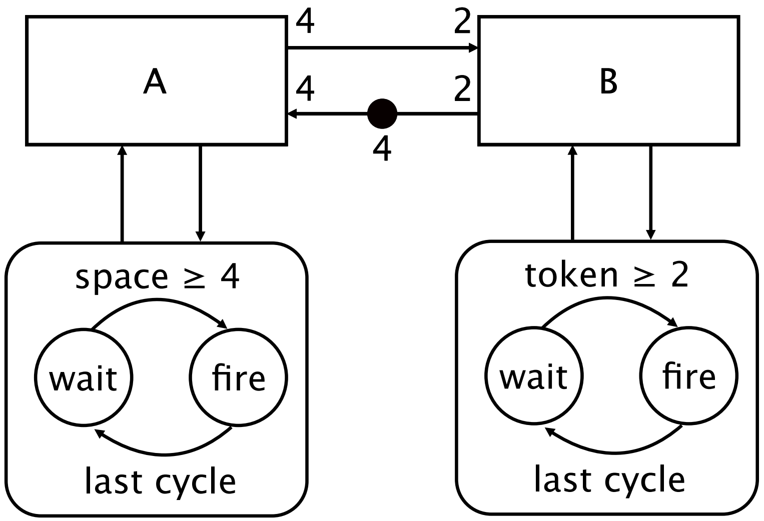
Problem: matching actor and hardware execution model



Problem: matching actor and hardware execution model

Acquire – Release

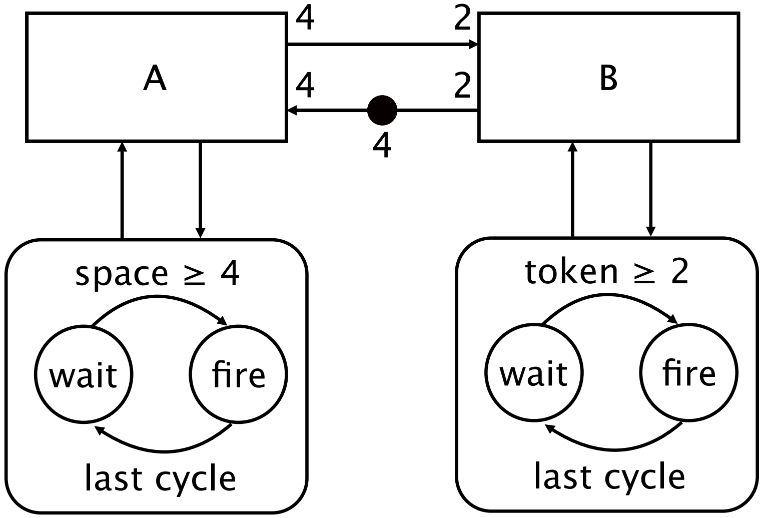
- Shared memory sync.

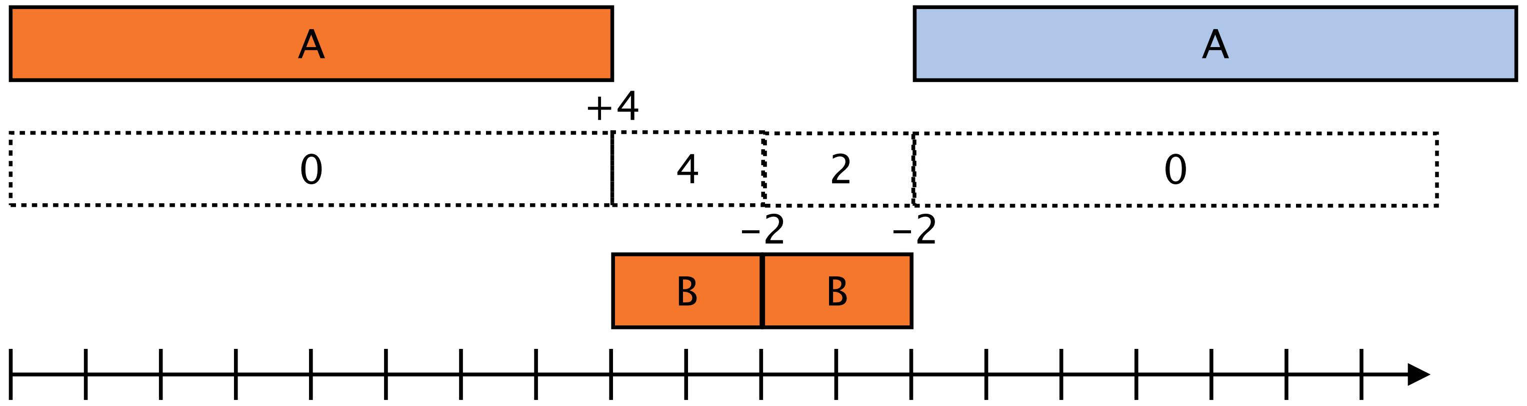


Problem: matching actor and hardware execution model

Acquire – Release

- Shared memory sync.

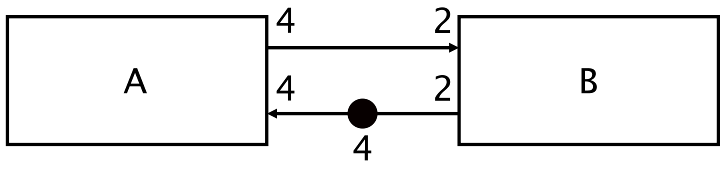




Problem: matching actor and hardware execution model

Acquire – Release

- Shared memory sync.



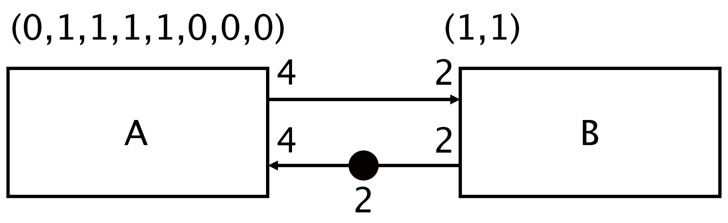
Problem: matching actor and hardware execution model

Acquire – Release

- Shared memory sync.

SDF Access Pattern [Tripakis et al. 11]

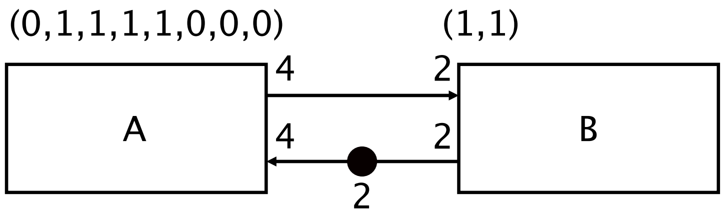
- Cycle accurate info
- Scalability issues



Problem: matching actor and hardware execution model

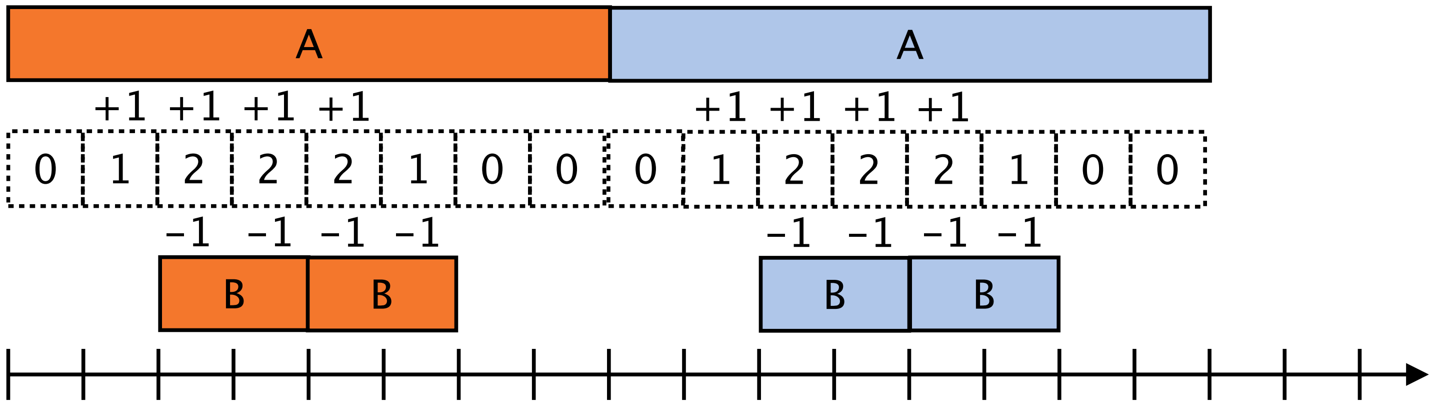
Acquire – Release

- Shared memory sync.



SDF Access Pattern [Tripakis et al. 11]

- Cycle accurate info
- Scalability issues



Affine approximation

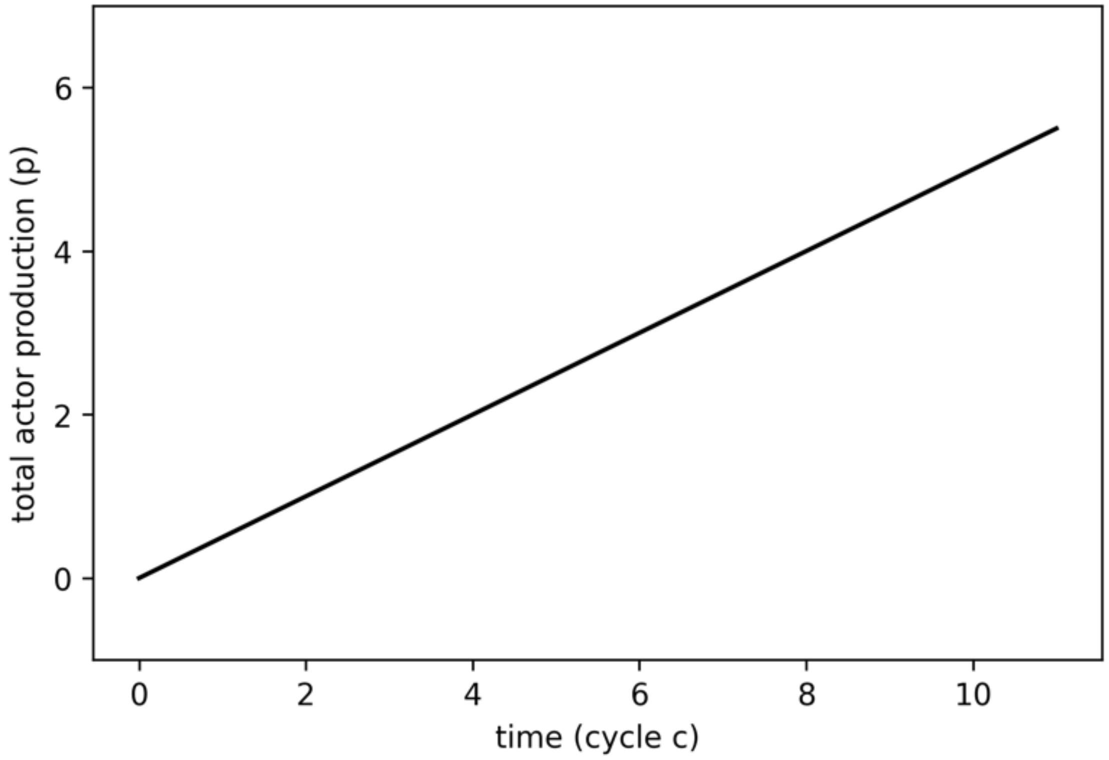
Metrics:

- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$

Affine approximation

Metrics:

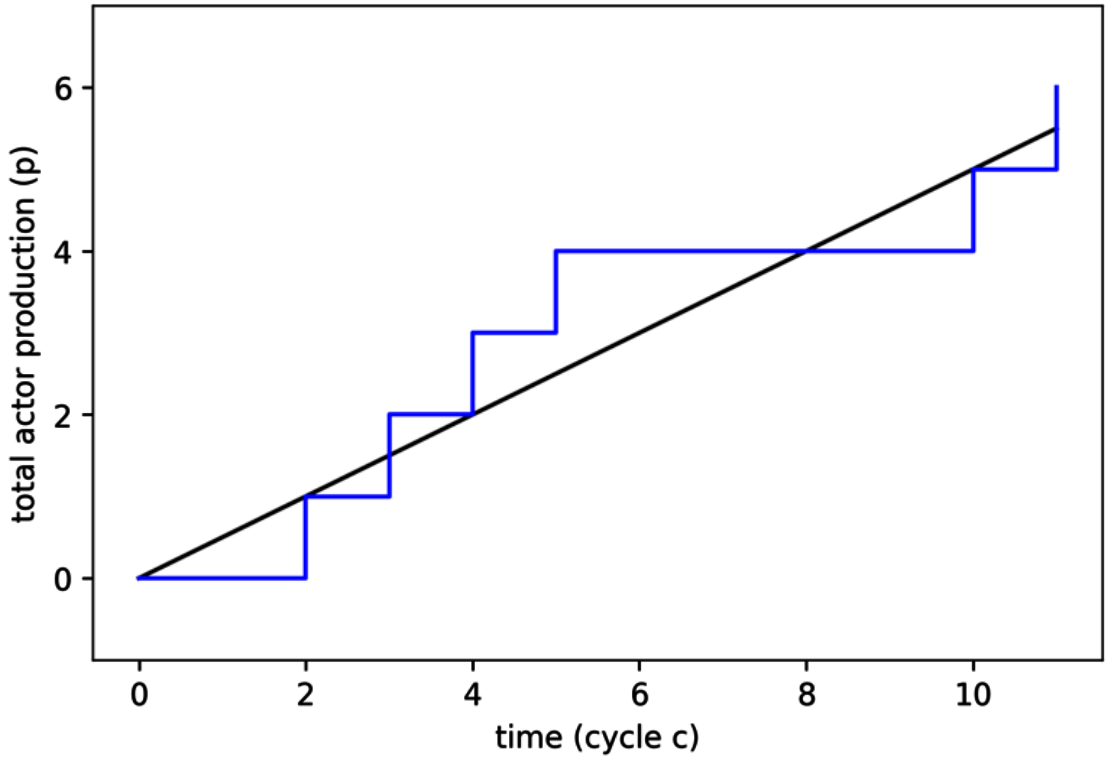
- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$



Affine approximation

Metrics:

- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$

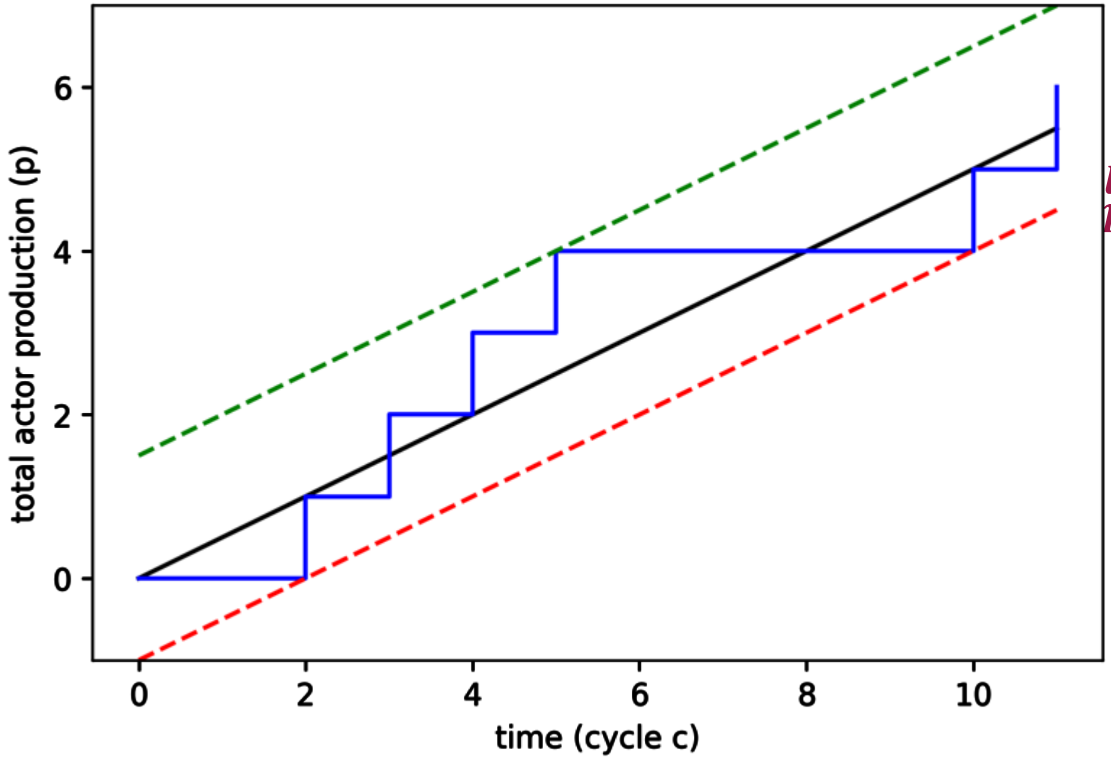


Affine approximation

Metrics:

- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$

$cc-\ \lambda\ p\ l\ \lambda\lambda\ \lambda\ p\ l\ pp\ \lambda\ p\ l\ ll\ \lambda\ p\ l$
$\leq pp\ c\ cc\ c \leq a\ p\ aa\ a\ p\ pp\ a\ p$
$\times cc+\ \lambda\ p\ u\ \lambda\lambda\ \lambda\ p\ u\ pp\ \lambda\ p\ u\ uu\ \lambda$
$p\ u$
$\mathbf{11}$



${}_p^l \leq p(c) \leq$

Affine approximation

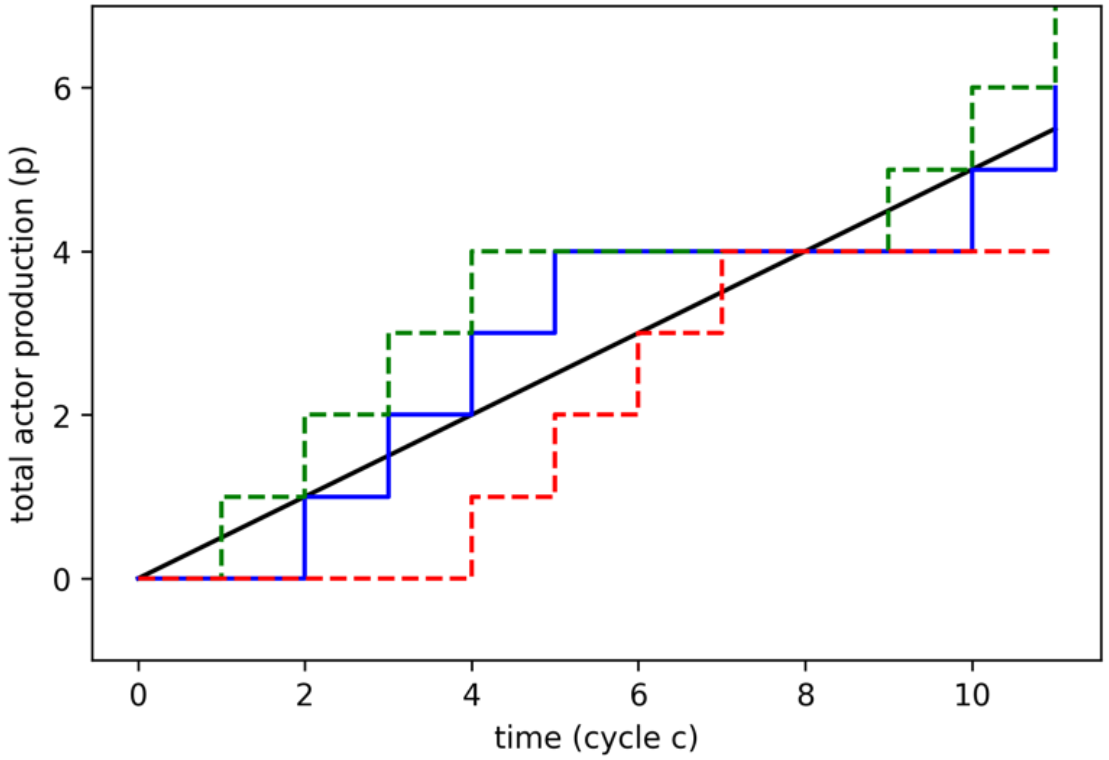
Metrics:

- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$

Affine approximation

Metrics:

- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$



Affine approximation

Metrics:

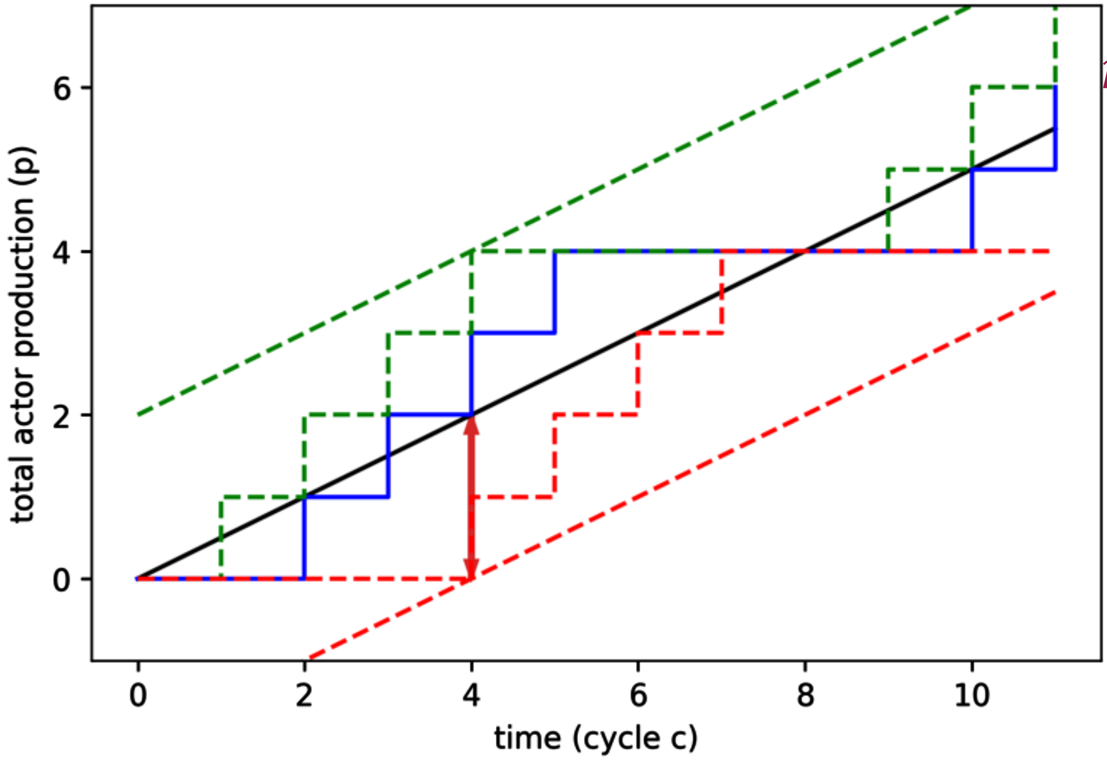
- $\tau_p = 4\ tokens\ per\ execution$
- $II = 8\ cycles$
- $a_p = 0.5\ token/cycle$

$\tau\ p\ c - \lambda p \leq p\ c \leq a\ p \times c + \lambda p$

$p\ p\ p\ p\ p\ \ p\ p\ \ p\ pp\ \tau\ p\ \ 1-\ \ \tau\ p$
$II\ \ 11-\ \ \tau\ p\ \ II\ \ \tau\ p\ \tau\tau\ \tau\ p\ pp\ \tau\ p$
$\tau\ p\ \ II\ IIII\ \ \tau\ p\ \ II\ \ 1-\ \ \tau\ p\ \ II$

Bounds:

$p \leq p(c) \leq$



Periodic scheduling

ILP constraints:

- **: FIFO size**
- **θ: delays**
- **φ: phase**

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

Affine scheduling

Periodic scheduling

$$(1)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{n} \geq \lambda_c^u + \lambda_p^l + a_p C_{under}$$

ILP constraints:

1. **Underflow**

- : FIFO size
- θ: delays
- φ: phase

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

Affine scheduling

Periodic scheduling

$$(1)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{n} \geq \lambda_c^u + \lambda_p^l + a_p C_{under}$$

$$(2)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{d} \leq \delta_{p\to c} - \lambda_c^u - \lambda_p^l - a_p C_{over}$$

ILP constraints:

1. **Underflow**
2. **Overflow**

- **: FIFO size**
- **θ: delays**
- **φ: phase**

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

Affine scheduling

Periodic scheduling

$$(1)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{n} \geq \lambda_c^u + \lambda_p^l + a_p C_{under}$$

ILP constraints:

1. Underflow
2. Overflow
3. Cycles

$$(2)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{d} \leq \delta_{p\to c} - \lambda_c^u - \lambda_p^l - a_p C_{over}$$

$$\frac{n}{d} = \frac{\tau_p}{II_p} \times \frac{II_c}{\tau_c}$$

$$(3)\quad \sum_{i=1}^{k} \left(\prod_{l=1}^{i-1} d_l \right) \left(\prod_{l=i+1}^{k} n_l \right) \varphi_i = 0$$

- : FIFO size
- θ: delays
- φ: phase

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

Affine scheduling

Periodic scheduling

$$(1)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{n} \geq \lambda_c^u + \lambda_p^l + a_p C_{under}$$

$$(2)\quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{d} \leq \delta_{p\to c} - \lambda_c^u - \lambda_p^l - a_p C_{over}$$

$$\frac{n}{d} = \frac{\tau_p}{II_p} \times \frac{II_c}{\tau_c}$$

$$(3)\quad \sum_{i=1}^{k} \left(\prod_{l=1}^{i-1} d_l \right) \left(\prod_{l=i+1}^{k} n_l \right) \varphi_i = 0$$

ILP constraints:

1. **Underflow**
2. **Overflow**
3. **Cycles**

Formally proven (A. Bouakaz thesis)

- δ: **FIFO size**
- θ: **delays**
- φ: **phase**

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

Affine scheduling

Periodic scheduling

$$(1) \quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{n} \geq \lambda_c^u + \lambda_p^l + a_p C_{under}$$

ILP constraints:

1. **Underflow**
2. **Overflow**
3. **Cycles**

$$(2) \quad \theta_{e_{p\to c}} + a_p \frac{\varphi_{p\to c}}{d} \leq \delta_{p\to c} - \lambda_c^u - \lambda_p^l - a_p C_{over}$$

$$\frac{n}{d} = \frac{\tau_p}{II_p} \times \frac{II_c}{\tau_c}$$

$$(3) \quad \sum_{i=1}^{k} \left(\prod_{l=1}^{i-1} d_l \right) \left(\prod_{l=i+1}^{k} n_l \right) \varphi_i = 0$$

Formally proven (A. Bouakaz thesis)

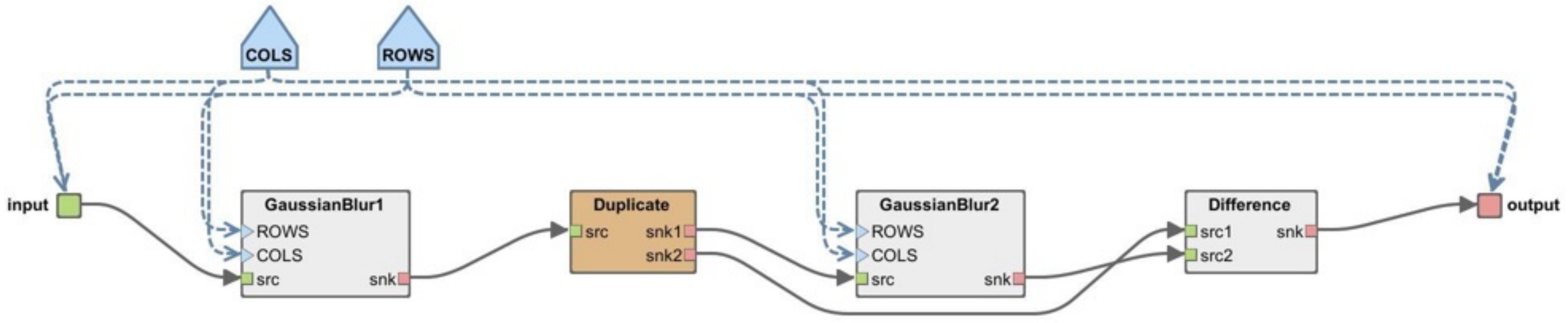
- **δ: FIFO size**
- **θ: delays**
- **φ: phase**

Fast for ILP formulation (3 variables per edge)

Guarantee optimal throughput (no push back by overflow)

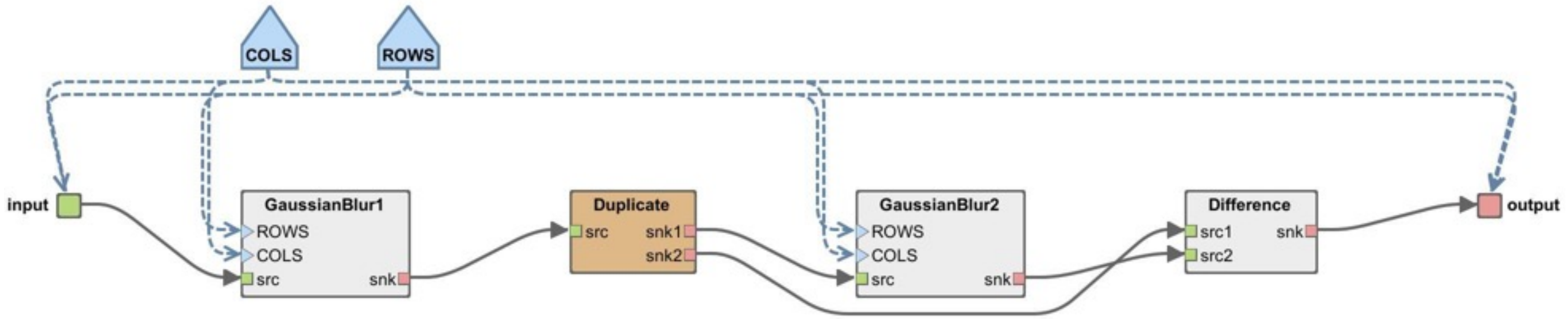
EXPERIMENTAL RESULTS

Gaussian Difference



[Undirected cycle]

Gaussian Difference

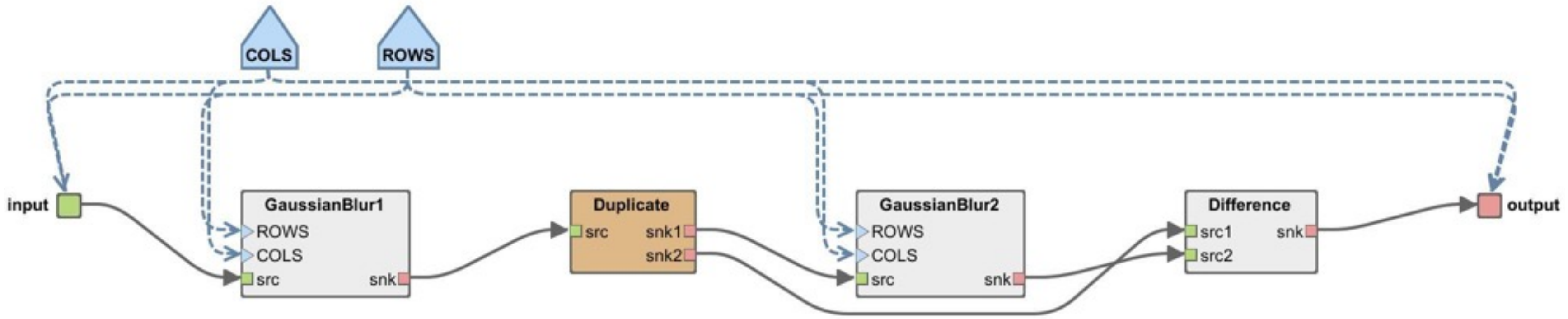


[Undirected cycle]

Setup:

- **Vitis Library kernels + graph**
- **PREESM computed FIFO sizes**

Gaussian Difference



[Undirected cycle]

Setup:

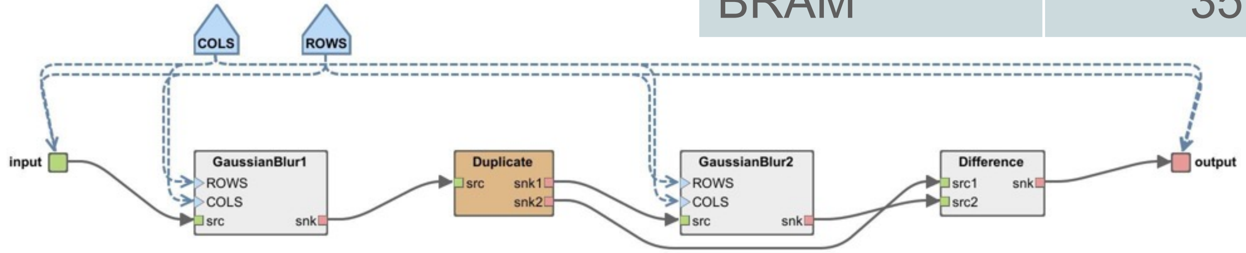
- **Vitis Library kernels + graph**
- **PREESM computed FIFO sizes**

	Vitis Library	PREESM
[Undirected cycle]	15360	8580
[FIFO]	2	1276 - 7022
BRAM	18	35 (+49%)
Latency (Synthesis)	19914	19914
Latency (Cosimulation)	20870	19827 (-5%)

Gaussian Difference

- Reduce kernel latency

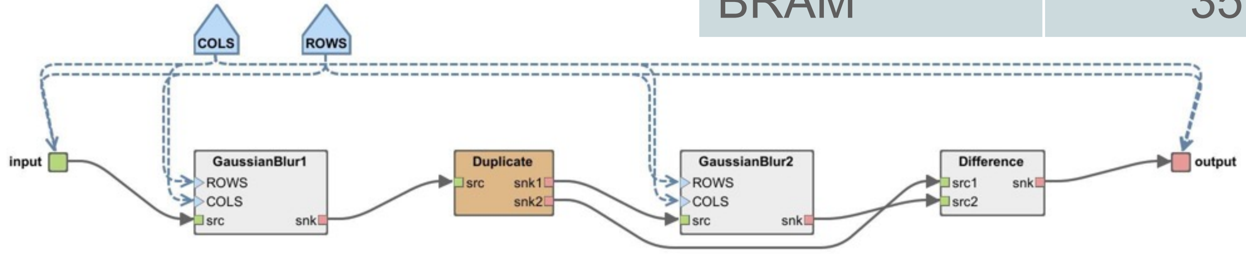
	Vitis + FIFO	PREESM
[Undir. cycle]	8580	2541
[FIFO]	1276 - 7022	2540
BRAM	35	12



Gaussian Difference

- **Reduce kernel latency**

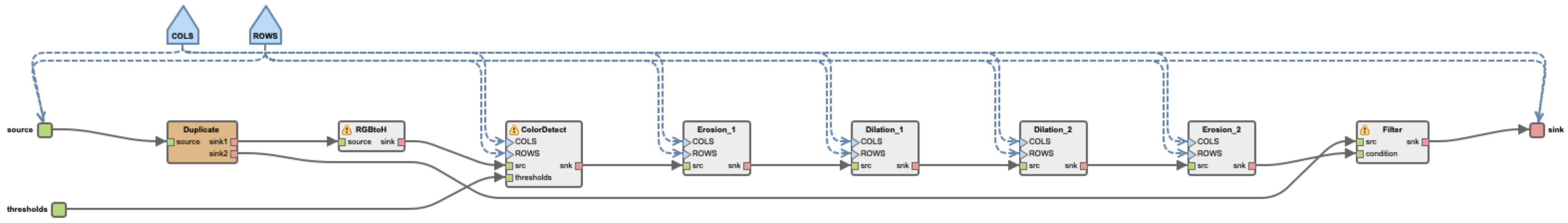
	Vitis + FIFO	PREESM
[Undir. cycle]	8580	2541
[FIFO]	1276 - 7022	2540
BRAM	35	12



Color Filter

- **Move downsampling to small kernel**

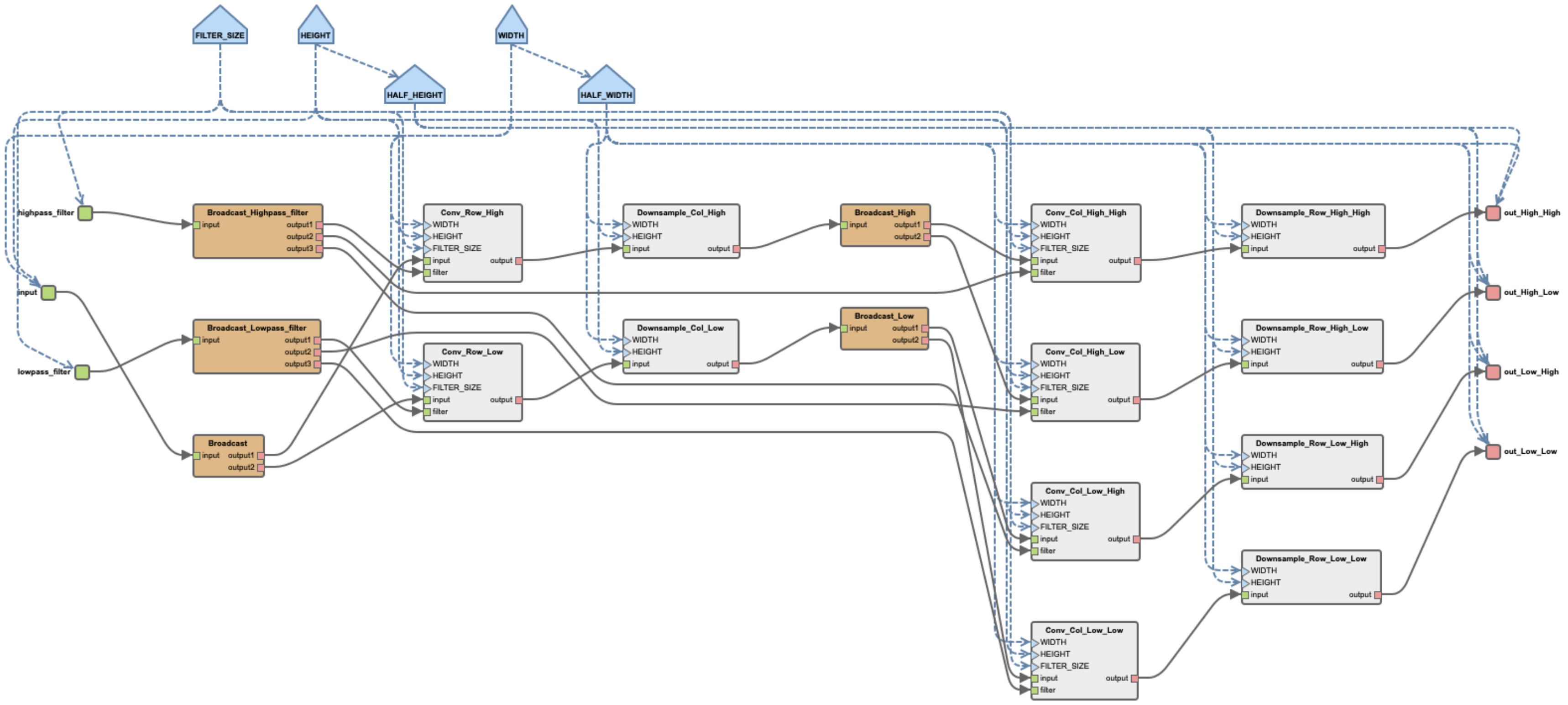
	PREESM	Optimized
[Undir. cycle]	808011	38002
[FIFO]	2 - 520936	2 - 5082
BRAM	790	61



2D Wavelet Transform

- **Reduce actor granularity**

	PREESM	Optimized
[FIFO]	12 - 50897	12 - 1510
BRAM	368	108



CONCLUSION

Contributions

- Dataflow code generation for FPGA using HLS
- Automatic scheduling and buffer sizing
- Open source implementation in PREESM

https://preesm.github.io

Contributions

- Dataflow code generation for FPGA using HLS
- Automatic scheduling and buffer sizing
- Open source implementation in PREESM

Future work

- Improve buffer sizing
    - Data dependency
    - Actor internal scheduling

https://preesm.github.io

Contributions

- Dataflow code generation for FPGA using HLS
- Automatic scheduling and buffer sizing
- Open source implementation in PREESM

Future work

- Improve buffer sizing
    - Data dependency
    - Actor internal scheduling
- Target heterogeneous platform
    - FPGA + CPU
    - Mapping + Scheduling + Code generation

https://preesm.github.io

Contributions

- Dataflow code generation for FPGA using HLS
- Automatic scheduling and buffer sizing
- Open source implementation in PREESM

Future work

- Improve buffer sizing
  - Data dependency
  - Actor internal scheduling
- Target heterogeneous platform
  - FPGA + CPU
  - Mapping + Scheduling + Code generation
- Design Space Exploration
  - Constraints based optimization
  - Memory optimization

https://preesm.github.io