Imperial College London

Automatic Kernel Code Generation for Focal-plane Sensor-Processor Devices

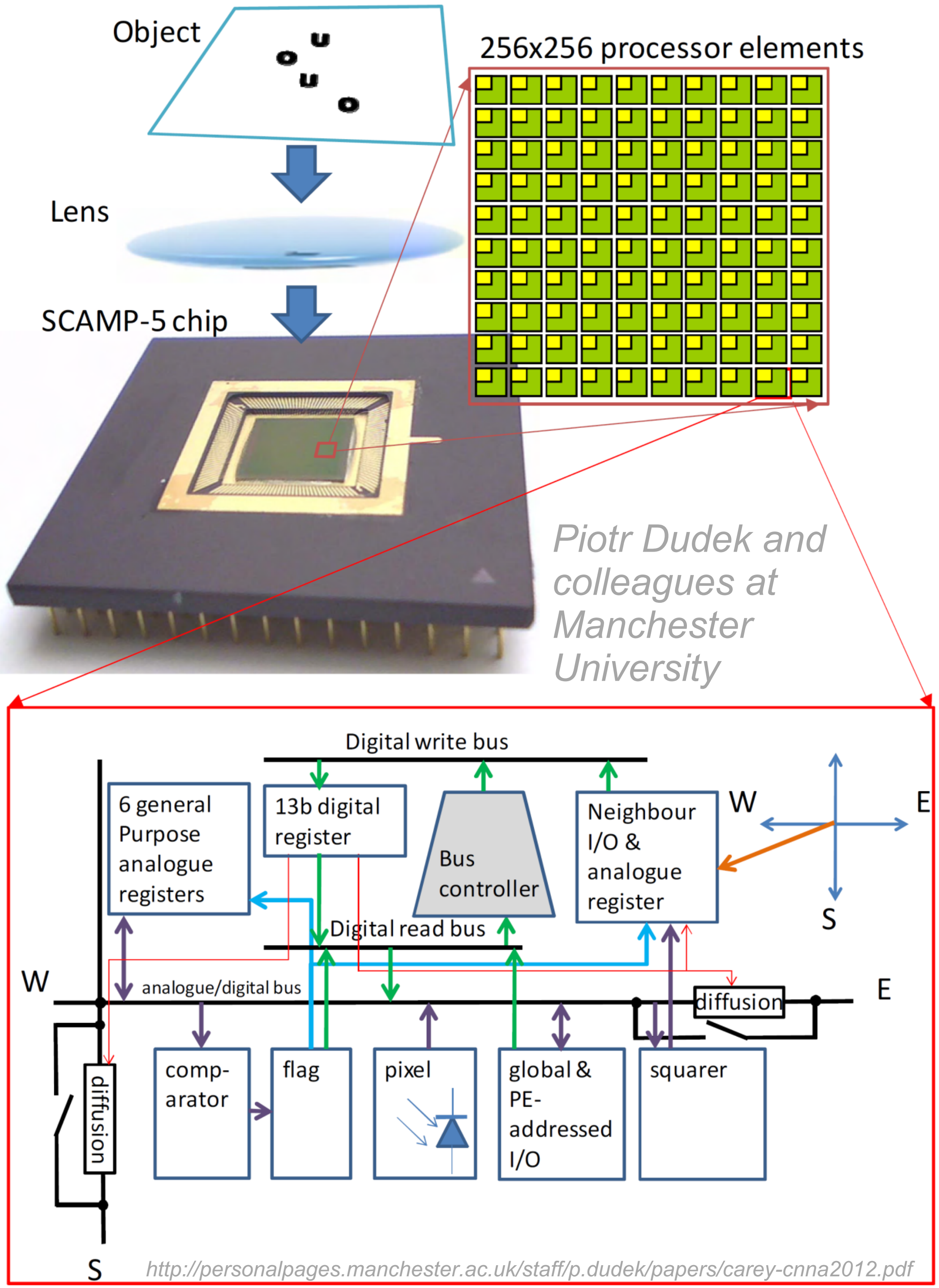
Thomas Debrunner - *MSc Student Imperial College London*
Paul Kelly - *Software Performance Optimisation Group Lead, Imperial College London*
Sajad Saeedi – *Research Fellow, Imperial College London*

With kind support from Piotr Dudek and his team at Manchester University

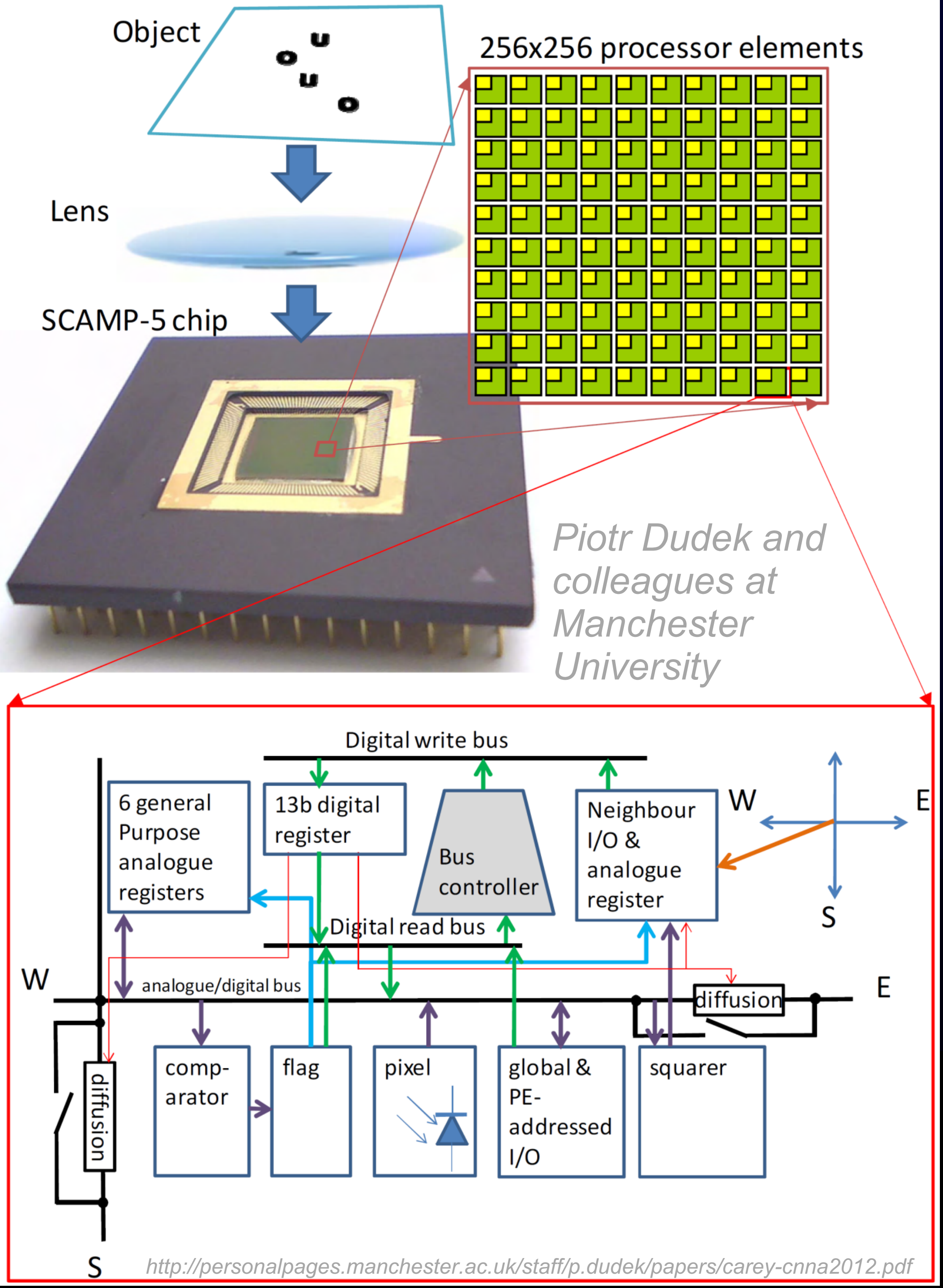
This work is part of the EPSRC "PAMELA" Project

Cameras produce images for humans, not machines

SCAMP 5 focal-plane sensor processor



- **256x256 SIMD** processor array
- **Light sensor** on every processor
- Ca.**170 transistors** per processor



SCAMP 5 focal-plane sensor processor

- **Seven registers** holding analogue values
- Computation by **moving charge**
- **Addition** is easy
- **No multiply**
- North-east-west-south **data movement**

Basic instruction set *(of interest)*

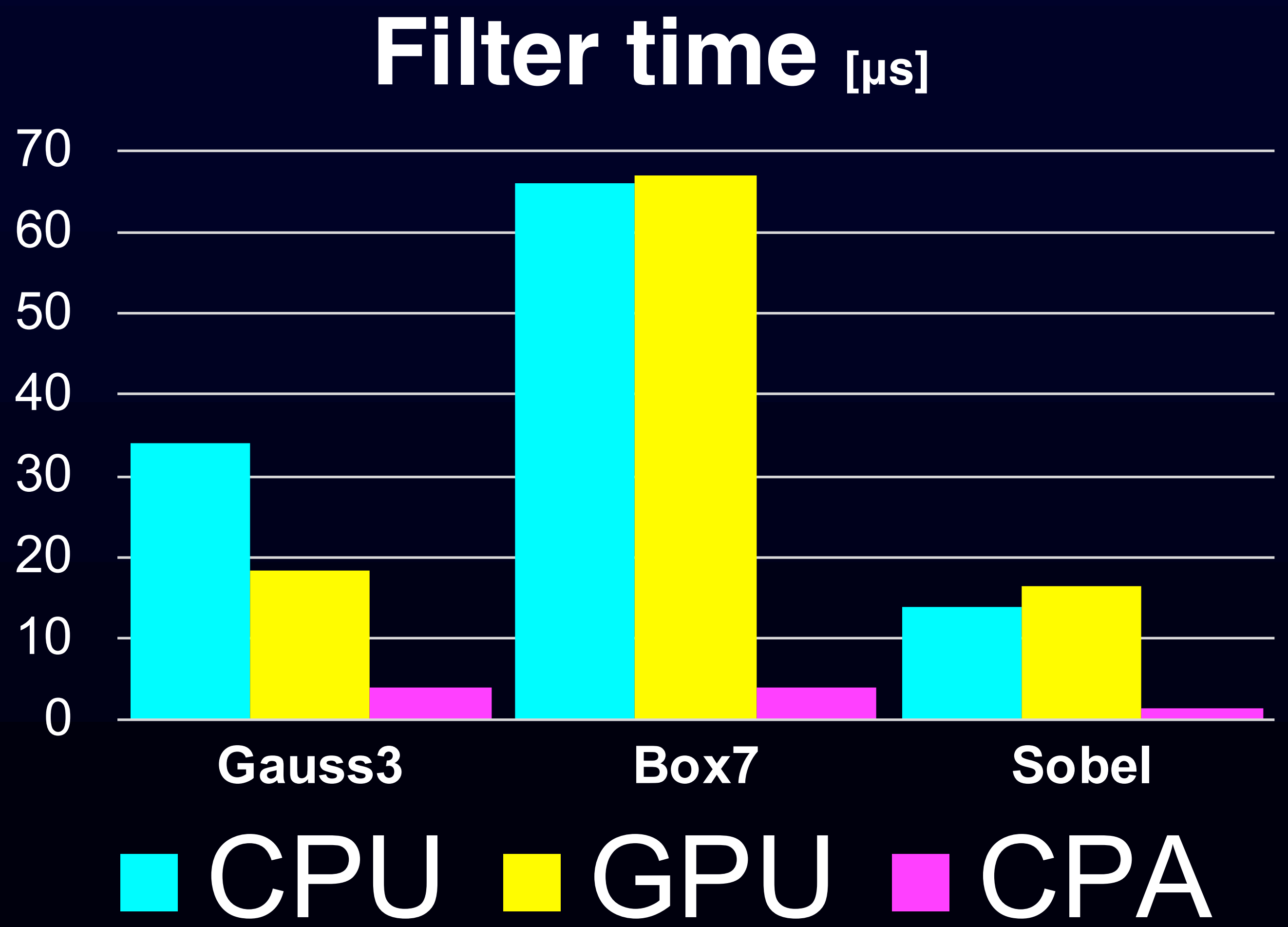
- **Shift** image x
- **Shift** image y
- **Add** two images
- **Subtract** two images
- **Scale** image by 1/2
- Take **absolute value** of image

This talk

- **How to do convolution filters on SCAMP 5?**
- For image filtering
- As a component in image processing algorithms
  - Notably CNNs

- Potential
  - low power
  - Extreme effective frame rate

- **Example:** Viola-Jones face detection
- **A compiler**: general code generator producing highly-optimised convolution implementations



CPU: INTEL i7-6700, GPU: NVIDIA TITAN X, CPA: SCAMP-5c estimate

Convolution filters on SCAMP 5

Easy filters

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

We can add repeatedly – so we can multiply by a constant

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Convolution filters on SCAMP 5

Harder filters

$$\begin{bmatrix} 0.125 & 0.25 & 0.125 \\ 0.25 & 0.5 & 0.25 \\ 0.125 & 0.25 & 0.125 \end{bmatrix}$$

Convolution filters on SCAMP 5

Harder filters – still easy

$$\begin{bmatrix} 0.125 & 0.25 & 0.125 \\ 0.25 & 0.5 & 0.25 \\ 0.125 & 0.25 & 0.125 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

We can divide by two repeatedly

Convolution filters on SCAMP 5

Hard filters

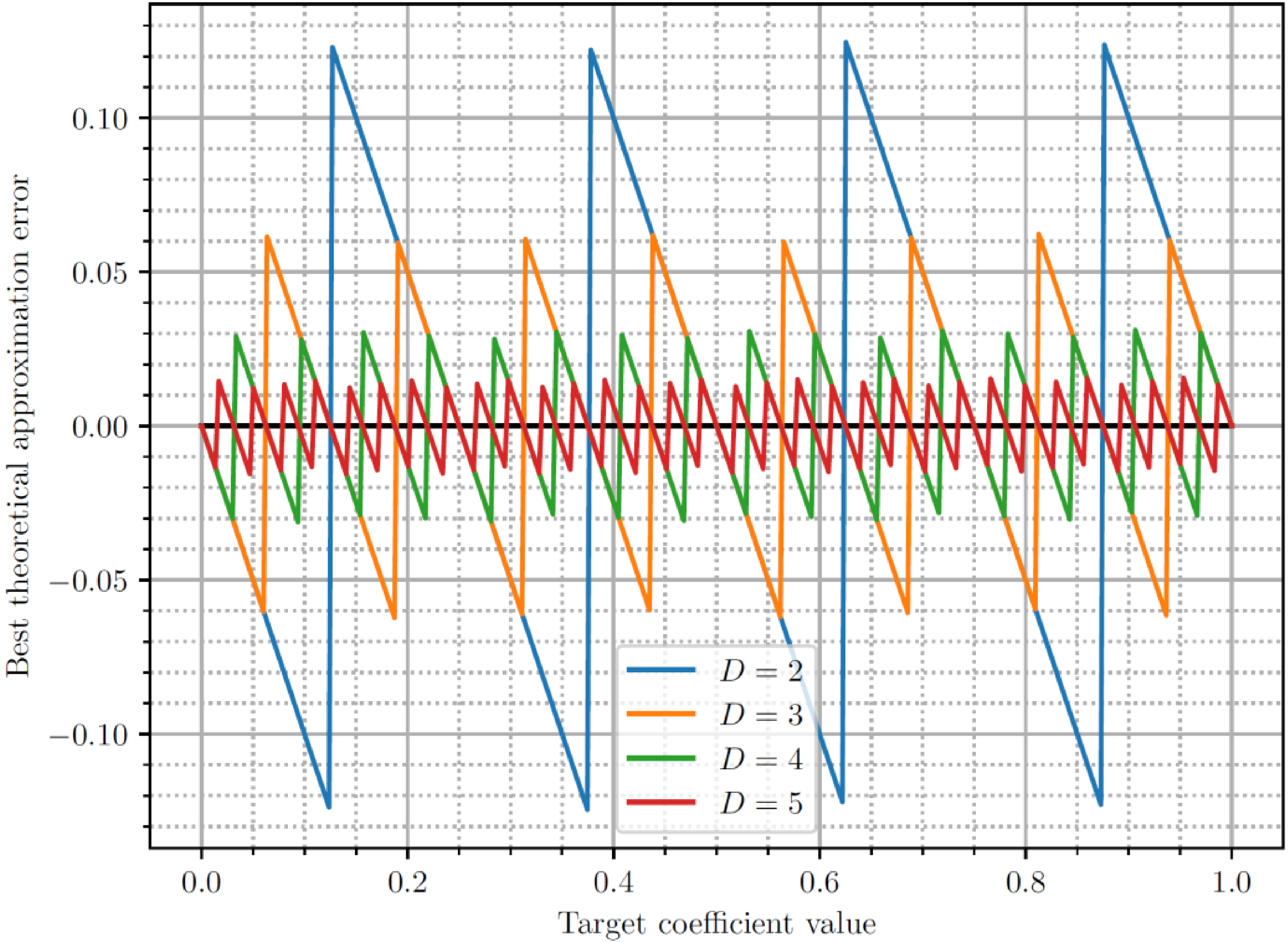
$$\begin{bmatrix} 1.12 & 0.23 & 0.88 \\ 0.36 & 0.51 & 0.89 \\ 0.16 & 0.13 & 0.73 \end{bmatrix}$$

Convolution filters on SCAMP 5

Hard filters – easy again

$$\begin{bmatrix} 1.12 & 0.23 & 0.88 \\ 0.36 & 0.51 & 0.89 \\ 0.16 & 0.13 & 0.73 \end{bmatrix} \approx \begin{bmatrix} 1.125 & 0.25 & 0.875 \\ 0.375 & 0.5 & 0.875 \\ 0.125 & 0.125 & 0.75 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 9 & 2 & 7 \\ 3 & 4 & 7 \\ 1 & 1 & 6 \end{bmatrix}$$

We can approximate



We can approximate

Filters often have repeated terms

$$\begin{bmatrix} 1.125 & 0.25 & 0.875 \\ 0.375 & 0.5 & 0.875 \\ 0.125 & 0.125 & 0.75 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 9 & 2 & 7 \\ 3 & 4 & 7 \\ 1 & 1 & 6 \end{bmatrix}$$

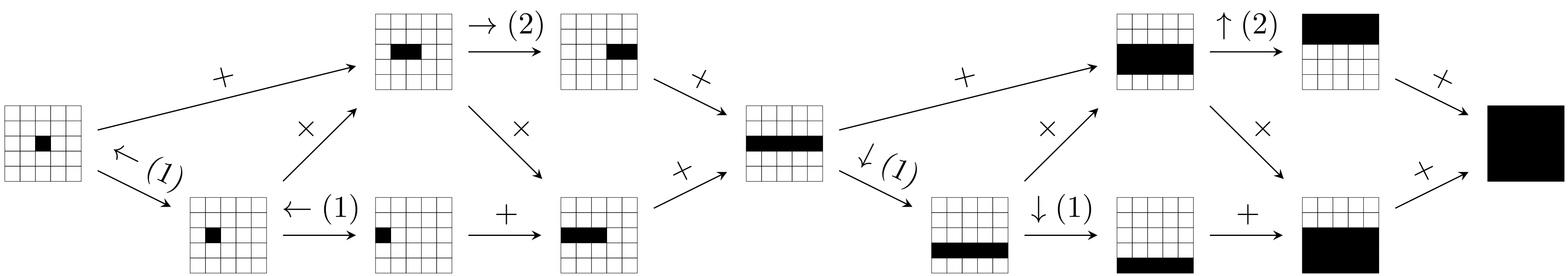
We implement multiplication using summations – so there are *lots* of common subterms

We can shift intermediate values to save redundant computation

Simple motivating (extreme) example

5x5 Box:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$




```
1 B = east(A)
2 A = add(A, B)
3 B = east(B)
4 B = add(B, A)
```

```
5 A = west(A)
6 A = west(A)
7 A = add(B, A)
8 B = north(A)
9 A = add(A, B)
```

```
10 B = north(B)
11 B = add(A, B)
12 A = south(A)
13 A = south(A)
14 A = add(B, A)
```

Finding a plan: End point

$$K = \begin{bmatrix} \frac{1}{8} & \frac{4}{8} & \frac{1}{8} \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 4 & 1 \end{bmatrix}$$

$$FS = \left\{ \begin{array}{ccc} (-1,0), & (0,0), & (0,0), \\ (1,0), & (0,0), & (0,0) \end{array} \right\}$$

"Final Set" (FS) of Partial Value Representatives (PVR)

The set of summands we need for the result of the filter application

Finding a plan: Starting point

$$S = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 0 & \underline{8} & 0 \end{bmatrix}$$

$$IS = \left\{ \begin{array}{cccc} \underline{(0,0),} & \underline{(0,0),} & \underline{(0,0),} & \underline{(0,0)} \\ \underline{(0,0),} & \underline{(0,0),} & \underline{(0,0),} & \underline{(0,0)} \end{array} \right\}$$

"Initial Set" (IS)

The set of summands of a fresh image

Objective

$$IS = \left\{ \begin{array}{llll} (0,0), & (0,0), & (0,0), & (0,0) \\ (0,0), & (0,0), & (0,0), & (0,0) \end{array} \right\}$$ *(Identity filter)*

$$FS = \left\{ \begin{array}{lll} (-1,0), & (0,0), & (0,0), \\ (1,0), & (0,0), & (0,0) \end{array} \right\}$$ *(desired filter)*

Find a sequence of operations to transform IS into FS

Instructions as transformations

Shifts:

(0 0)

(2 4)

$$\xrightarrow[(3\ 3)]{(1\ 1)}$$

$\rightarrow(1)\ \searrow(1)$

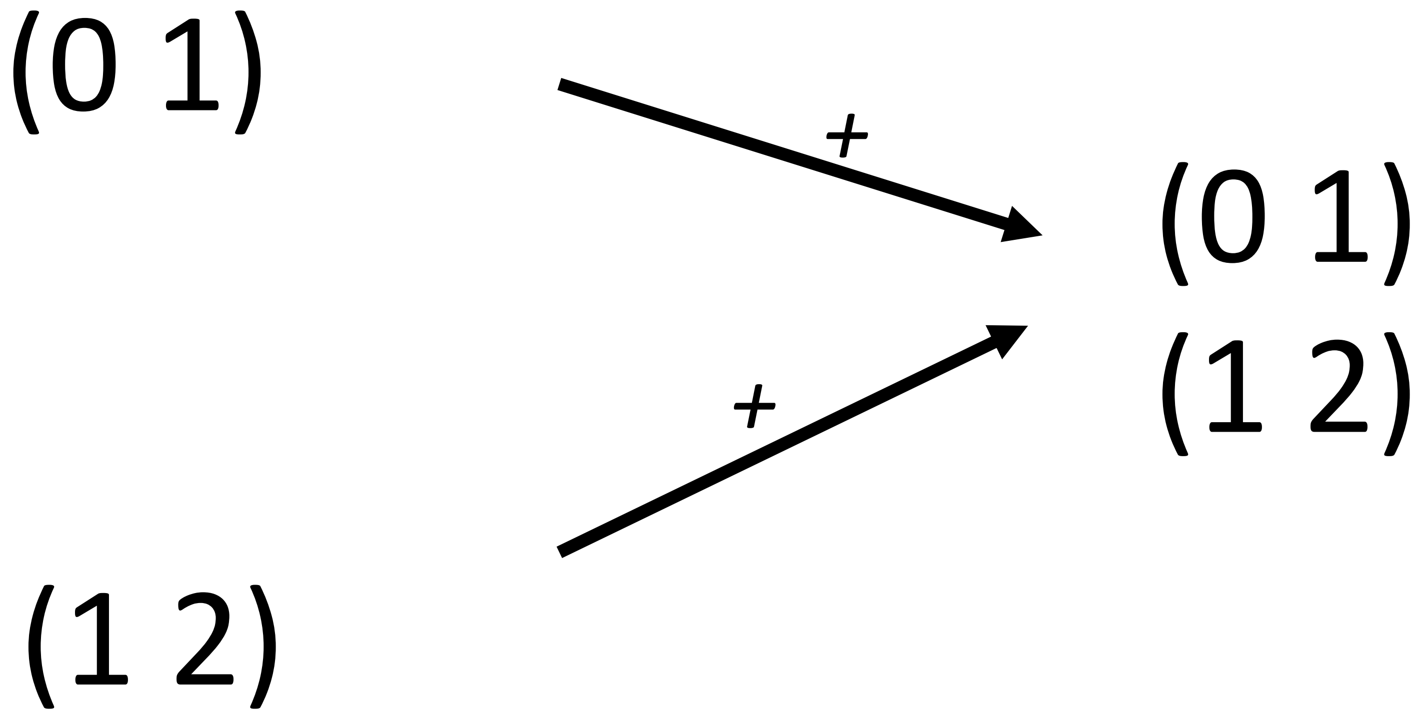
Instructions as transformations

Scales (Div2):

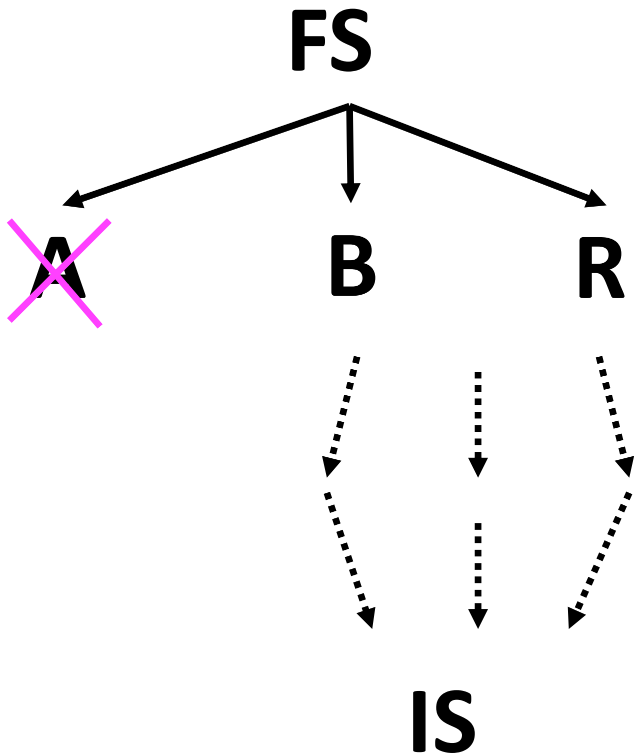
$$\begin{matrix}(0\ 0)\\(0\ 0)\end{matrix} \xrightarrow{\quad +(1) \quad} (0\ 0)$$

Instructions as transformations

Additions / Subtractions:



Reverse Split



A, B transformable

Recursive, continue with B, R

Reverse Split

Pruning

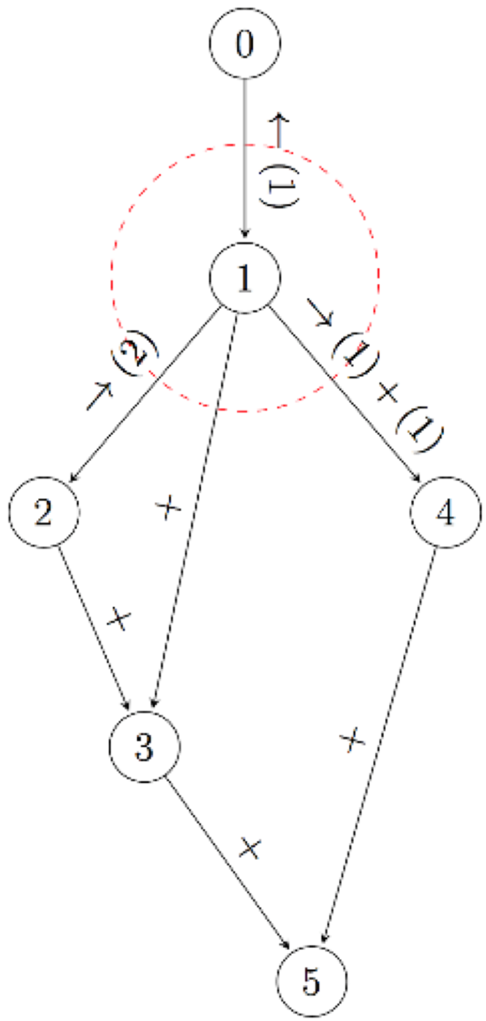
We prune splits that would exceed the number of registers in the SCAMP 5 device (seven)

We prune subtrees when the resulting instruction sequence is longer than the best so far

We attempt heuristically-promising splits first

Example

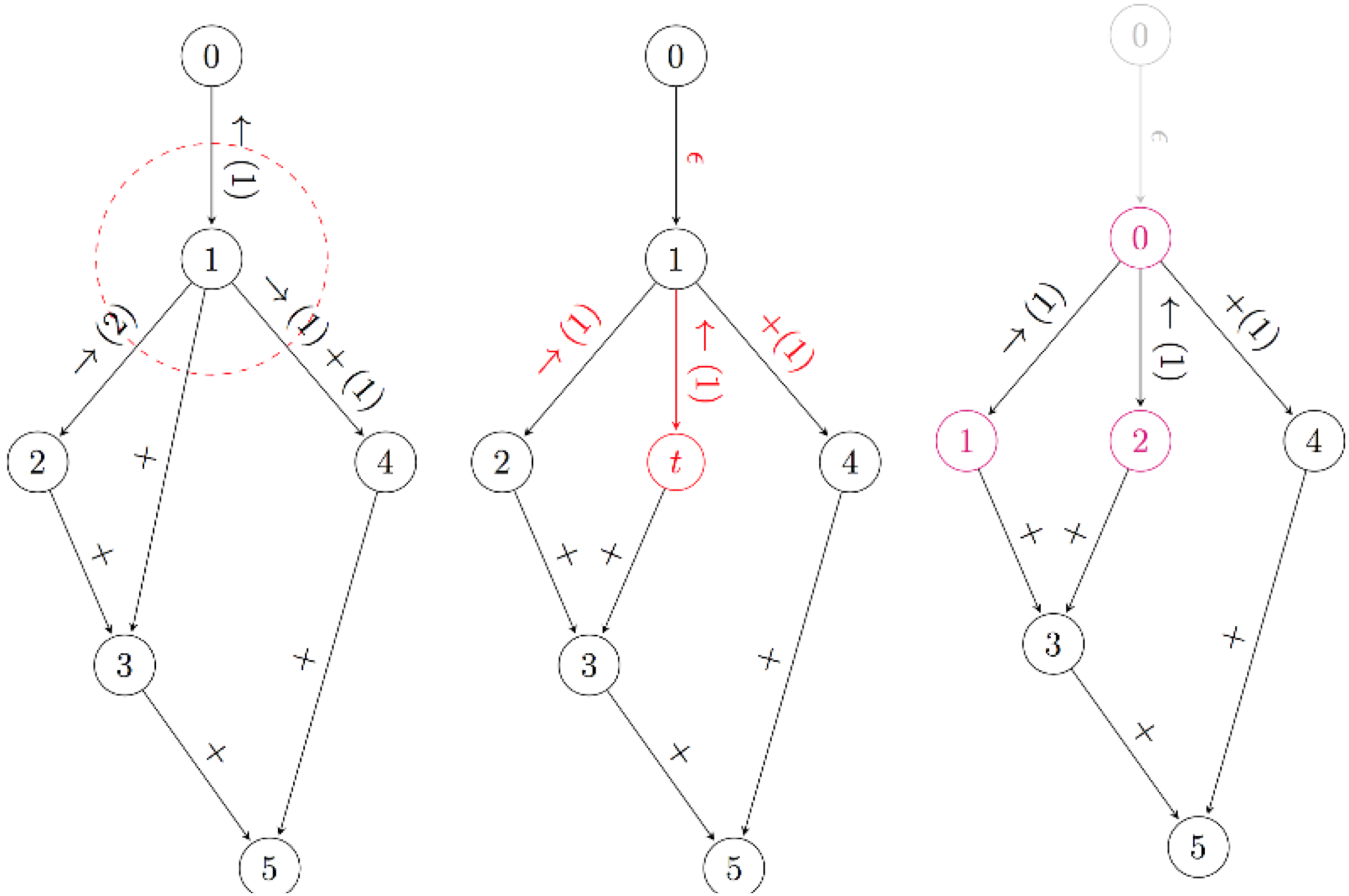
$$\begin{bmatrix} 1 & 0.5 & 1 \end{bmatrix}$$



node1 = east(node0)
node2 = west(node1)
node2 = west(node2)
node4 = west(node1)
node4 = div2(node4)
node3 = add(node2,node1)
node6 = add(node3, node4)

Graph Relaxation

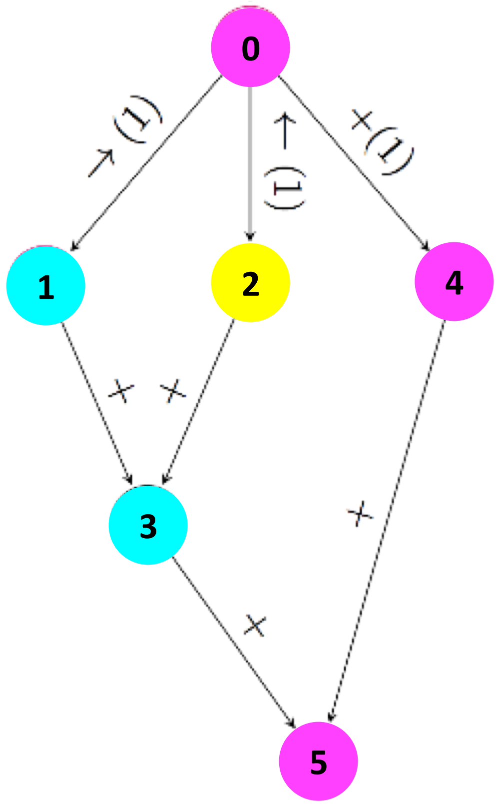
$$\begin{bmatrix} 1 & 0.5 & 1 \end{bmatrix}$$



Apply a systematic retiming to minimize shifts

Register Allocation

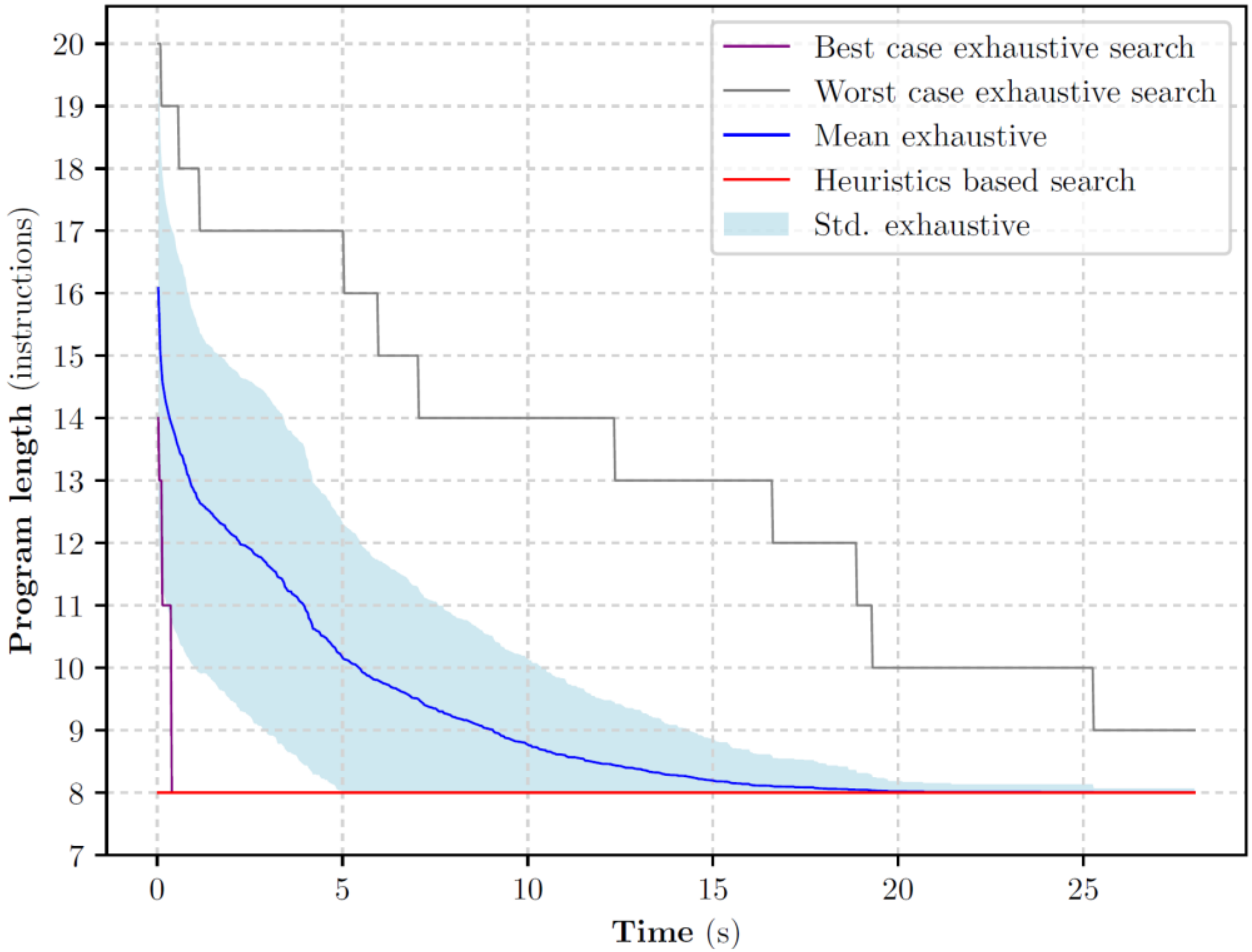
$$\begin{bmatrix} 1 & 0.5 & 1 \end{bmatrix}$$



Final resulting code:

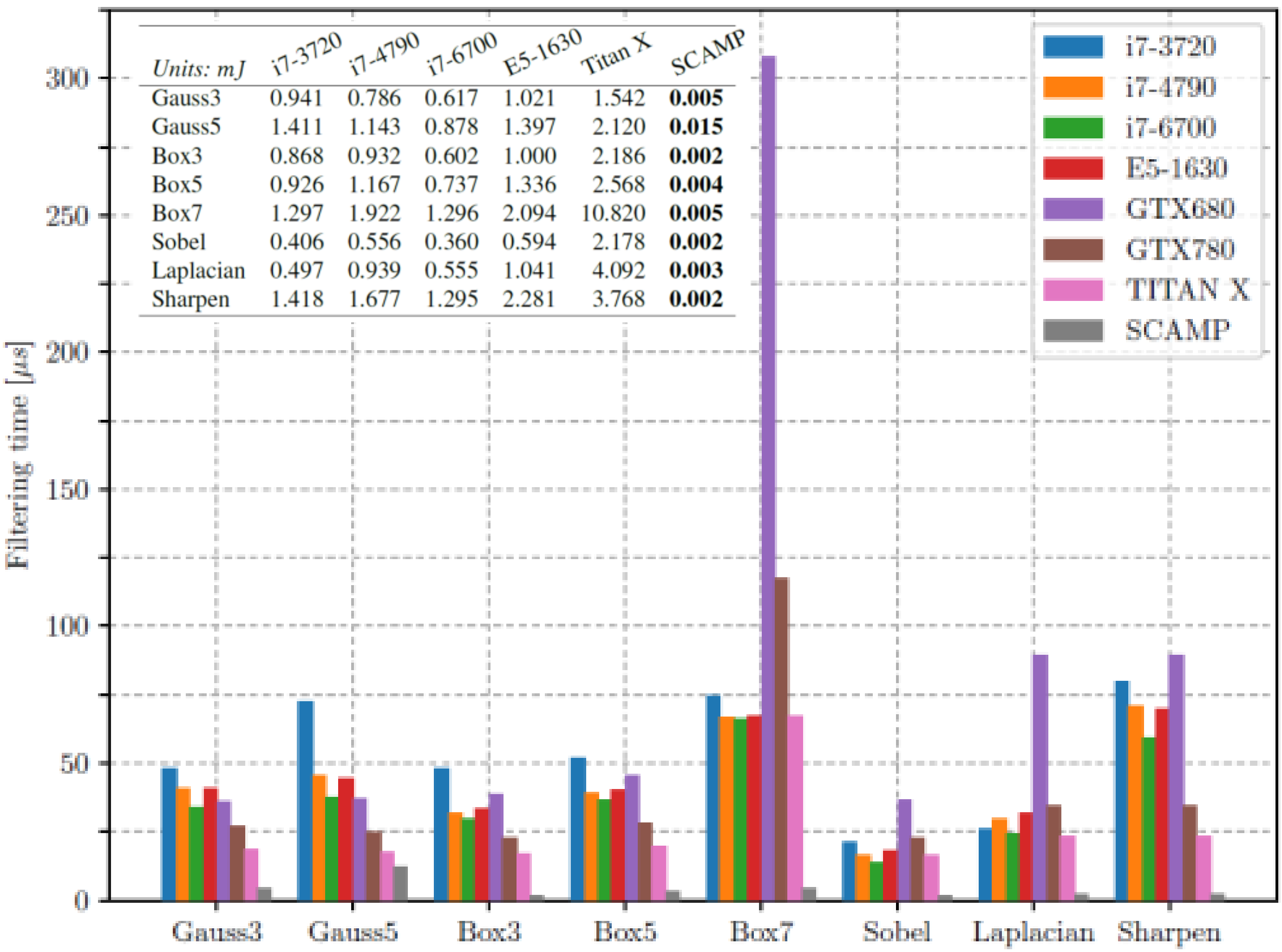
B = west(A)
C = div2(A)
B = add(C, B)
A = east(A)
A = add(B, A)

Evaluation



Full exhaustive search, compared to heuristic search on Sobel 3 × 3 filter (sampled over 256 runs)

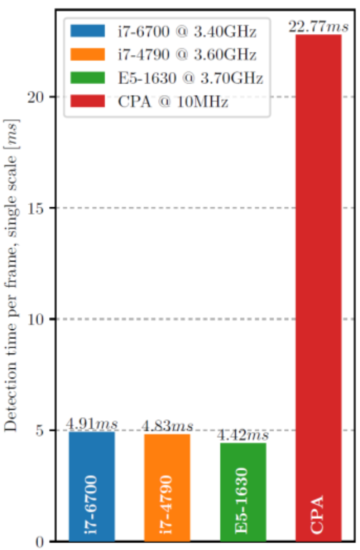
Evaluation

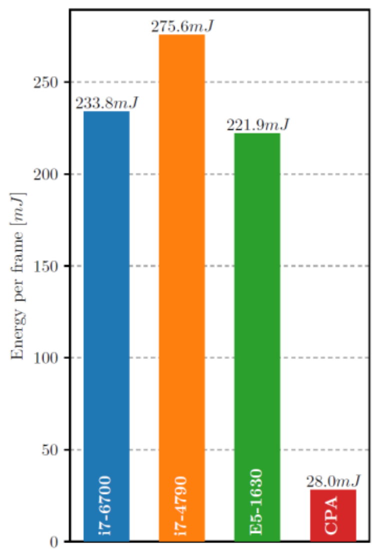


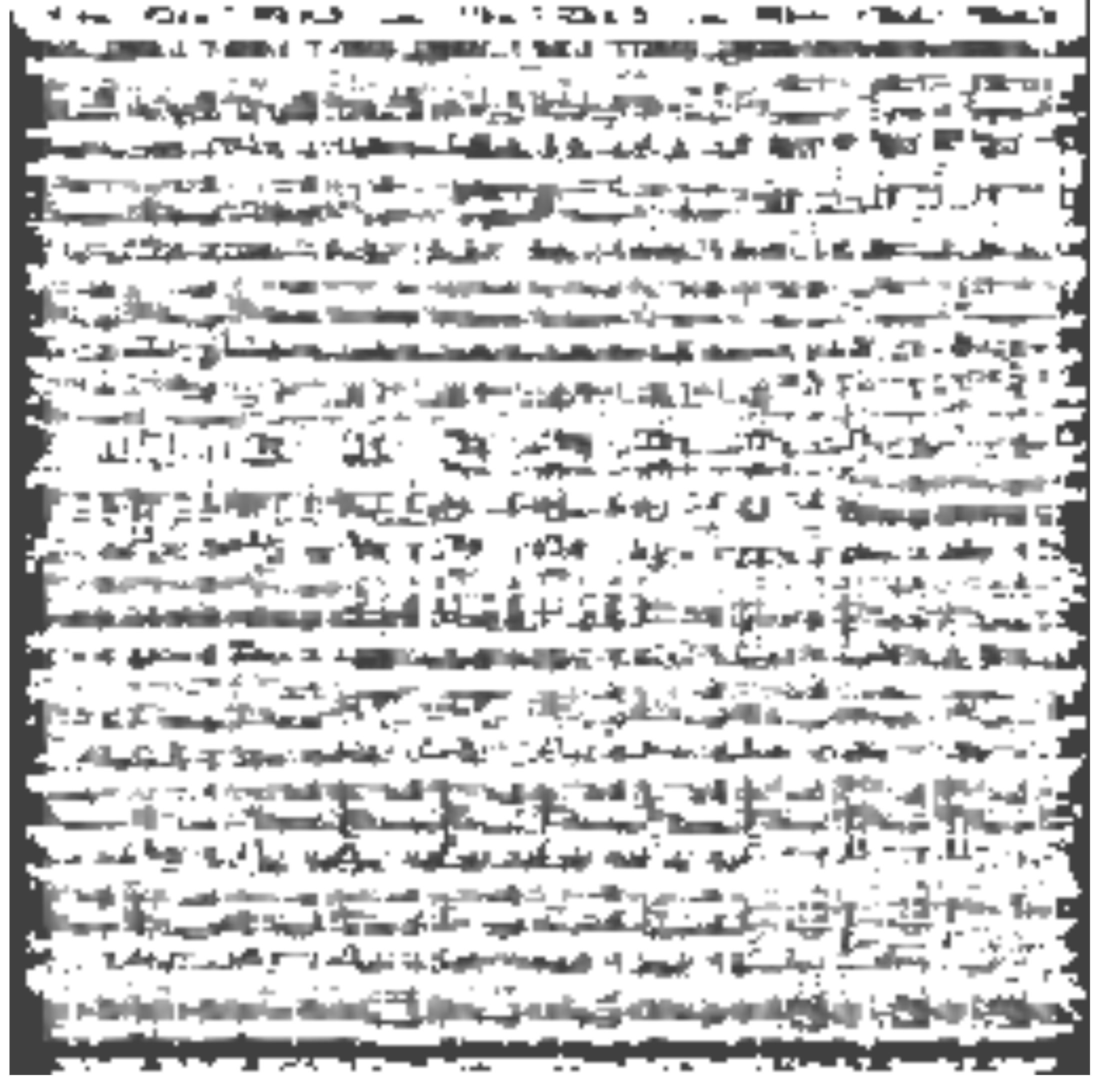
Units: mJ	i7-3720	i7-4790	i7-6700	E5-1630	Titan X	SCAMP
Gauss3	0.941	0.786	0.617	1.021	1.542	**0.005**
Gauss5	1.411	1.143	0.878	1.397	2.120	**0.015**
Box3	0.868	0.932	0.602	1.000	2.186	**0.002**
Box5	0.926	1.167	0.737	1.336	2.568	**0.004**
Box7	1.297	1.922	1.296	2.094	10.820	**0.005**
Sobel	0.406	0.556	0.360	0.594	2.178	**0.002**
Laplacian	0.497	0.939	0.555	1.041	4.092	**0.003**
Sharpen	1.418	1.677	1.295	2.281	3.768	**0.002**

- 8 common filter examples on 256×256 8-bit grayscale image
- CPU and GPU: default implementations shipped with OpenCV 3.3.0, with TPP and IPP enabled and with CUDA V8.0.61
- Power estimated based on TDP and time

7 Stage Viola-Jones Face Detector







- Due to code size and other limitations, we were only able to run a 7-stage Viola-Jones face detector
- It works as well as a 7-stage CPU implementation
- But for full accuracy, 25 stages are needed. SCAMP 5 would be slower than CPUs, but uses much less energy

Conclusions

Convolution filters are a key capability

- With a suitable code generator we can do a lot with very very simple hardware
- By trading approximation against efficiency we can do even more

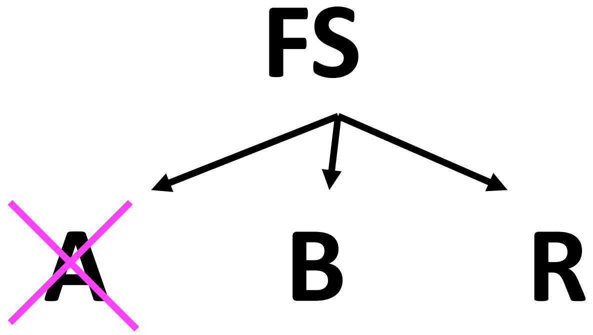
Near-camera processing is the only way we can approach biological levels of energy efficiency

There is a spectrum of design choices:

- How much to do in analogue
- Where to convert to digital
- How compute is distributed and connected to the sensors
- How to preprocess to reduce larger-scale data movement

Backup

Reverse Split



A, B transformable

Example

$$\begin{bmatrix} 1 & 0.5 & 1 \end{bmatrix}$$

FS

$(-1 \;\; 0)$

$(-1 \;\; 0)$

$(\;0 \;\; 0)$

$(\;1 \;\; 0)$

$(\;1 \;\; 0)$

A $\cancel{(1 \;\; 0)}$

$\cancel{(1 \;\; 0)}$

B $(-1 \;\; 0)$

$(-1 \;\; 0)$

R $(0 \;\; 0)$

FS

(-1 0)

(-1 0)

(0 0)

(1 0)

(1 0)

+ **A** ~~(1 0)~~

~~(1 0)~~

+ **B** (-1 0)

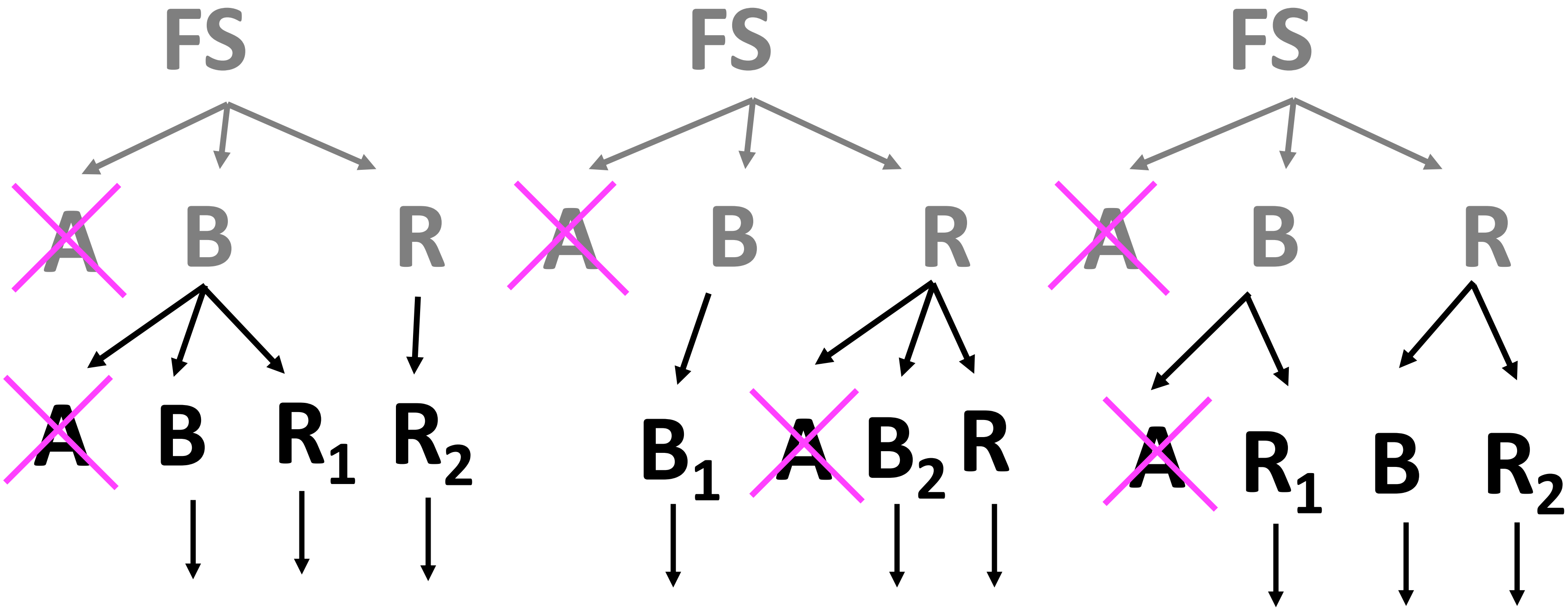
(-1 0)

+ **R** (0 0)

→(2)

R (0 0)

B (-1 0)
(-1 0)



FS
A
B
R
A
B
R_1
R_2
FS
A
B
R
B_1
A
B_2
R
FS
A
B
R
A
R_1
B
R_2

R (0 0)

B (-1 0)
(-1 0)

A ~~(0 0)~~

B (-1 0)
(-1 0)

→(1)
+(1)

B $\begin{matrix} (-1 & 0) \\ (-1 & 0) \end{matrix}$

$$\begin{array}{ccccc} & & & & \textbf{IG} \\ \mathbf{B} & \begin{matrix}(-1\ \ 0)\\(-1\ \ 0)\end{matrix} & & \xleftarrow{\quad\leftarrow(1)\quad} & \begin{matrix}(0\ \ 0)\\(0\ \ 0)\end{matrix}\end{array}$$