PACT
Munich, Germany
August 2002

Image Processing on the eXtreme Processing Platform®

Robert Strzodka
Numerical Analysis and Scientific Computing
University of Duisburg
http://www.numerik.math.uni-duisburg.de
strzodka@math.uni-duisburg.de

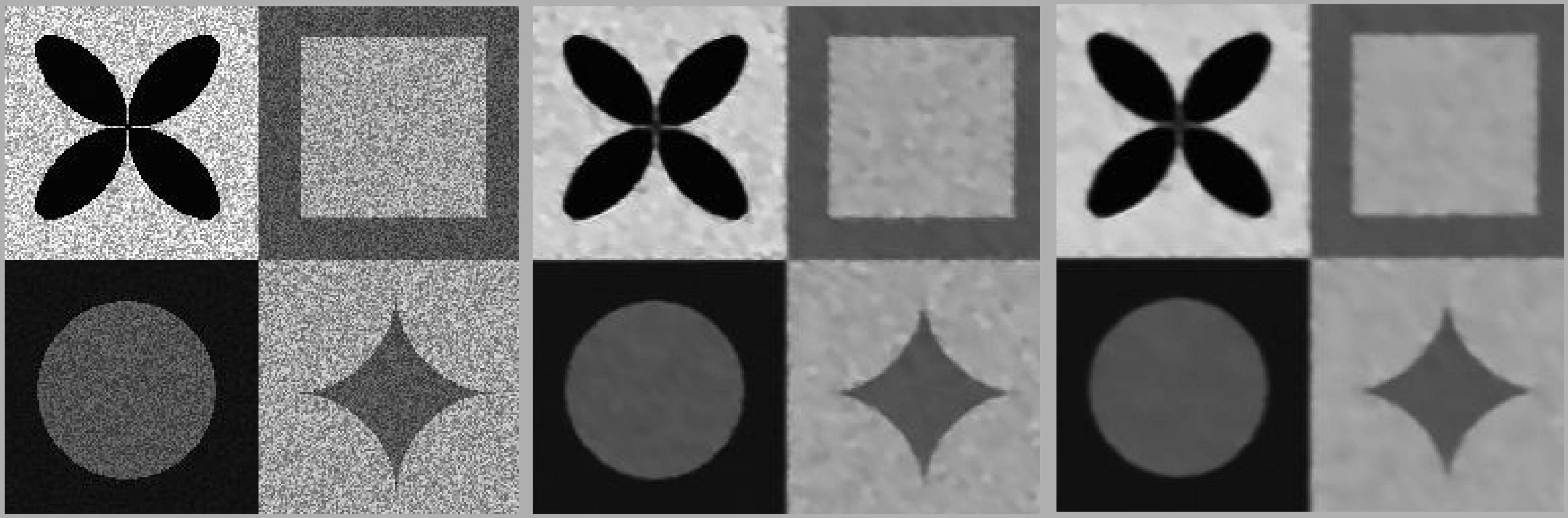
Overview

- Introduction
- Data-Flow & Architectures
- Implementations on the XPP
- Performance & Configurations
- Conclusions

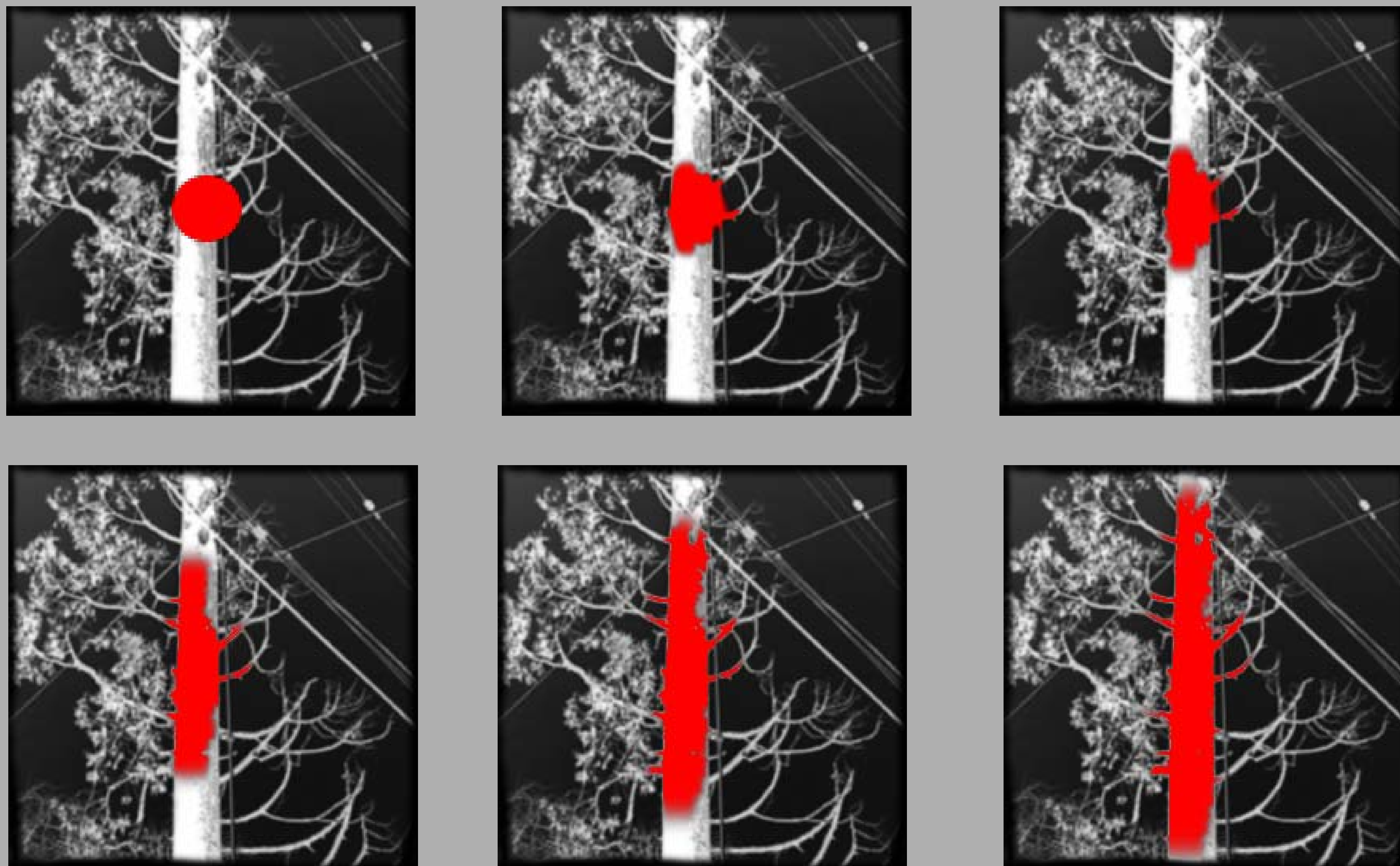
Some Key-Tasks in Image Processing

- Denoising
- Segmentation
- Matching
- Visualization

Denoising by anisotropic diffusion



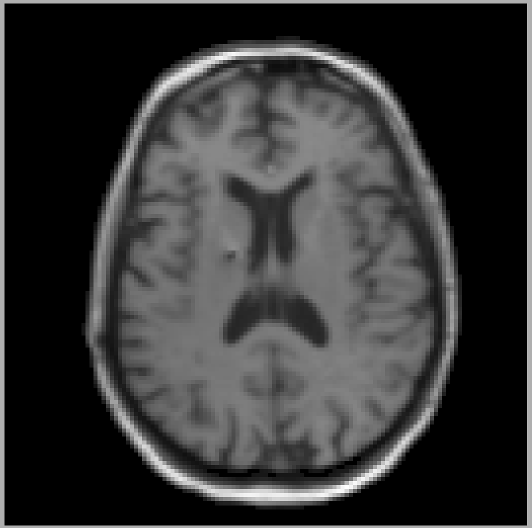
Segmentation by the level-set method



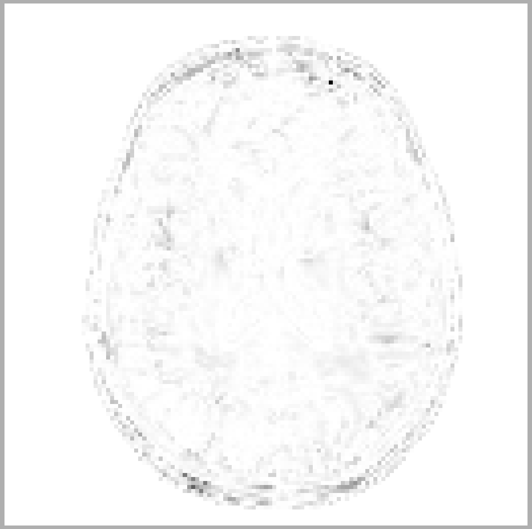
Matching by a gradient-flow method

original image

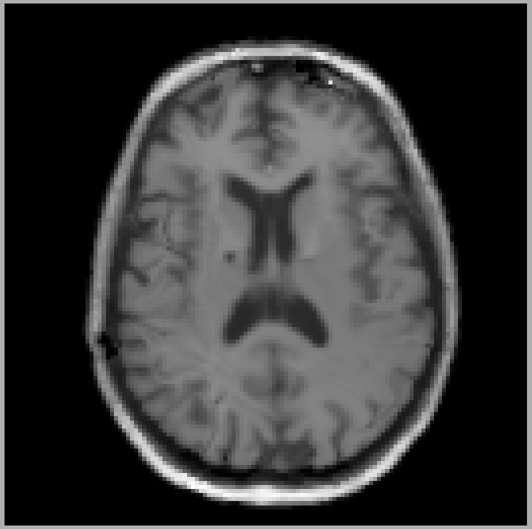
deformed image



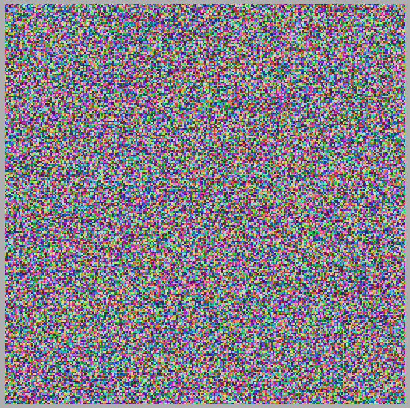
matching error

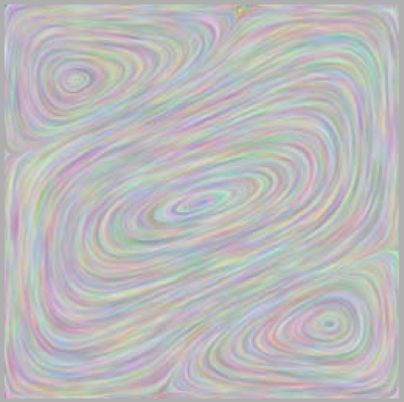


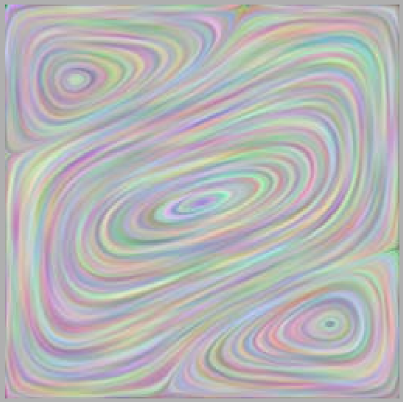
matching result

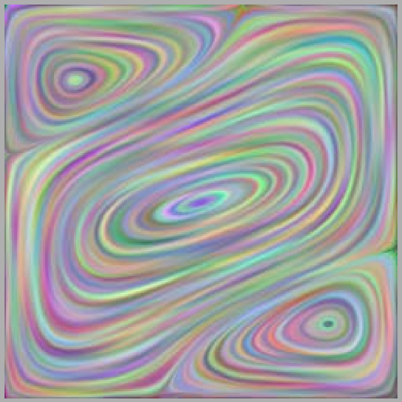


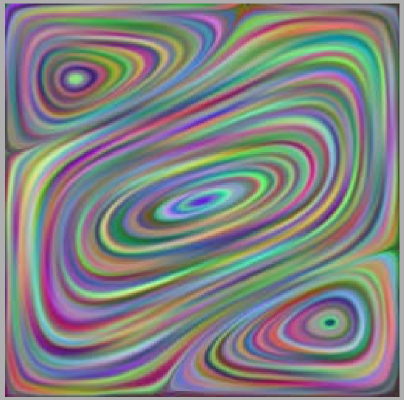
Visualization of a vector field


Initial image


Step 1


Step 2


Step 3


Step 4

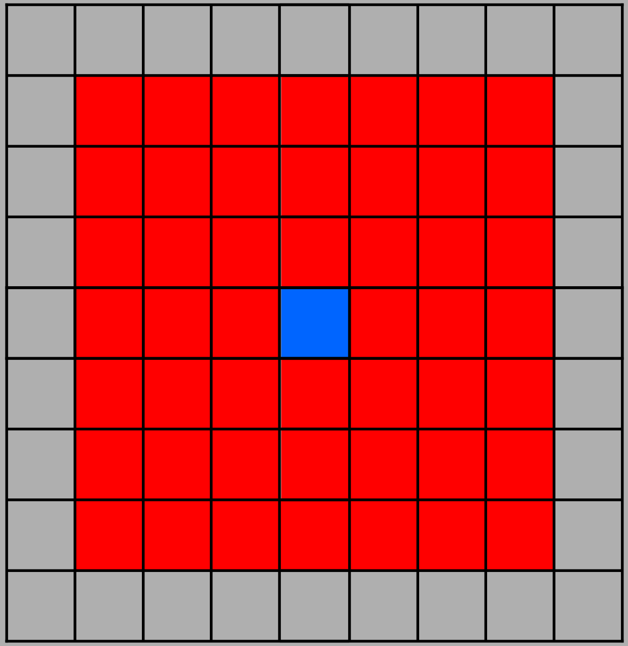

Step 5


Step 7


Step 10

Data-Flow in One Iteration

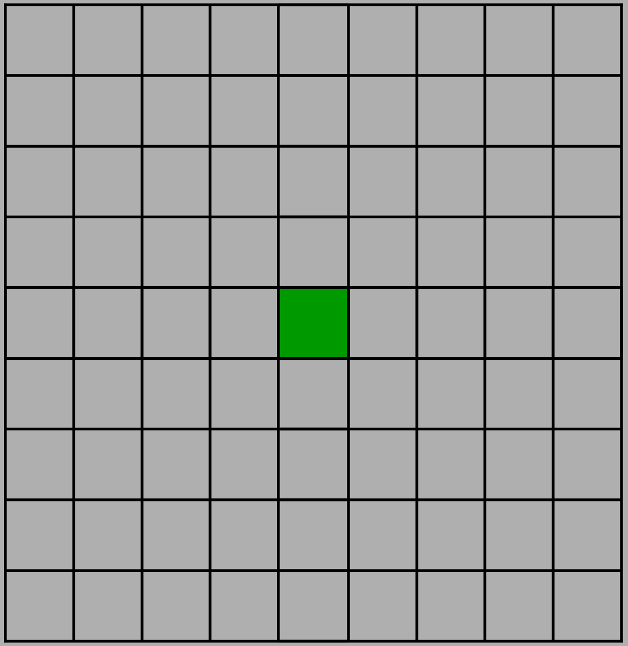
Step n



$$F\left(\left(X_{\beta}^{n}\right)_{|\beta-\alpha|\le C}\right)$$

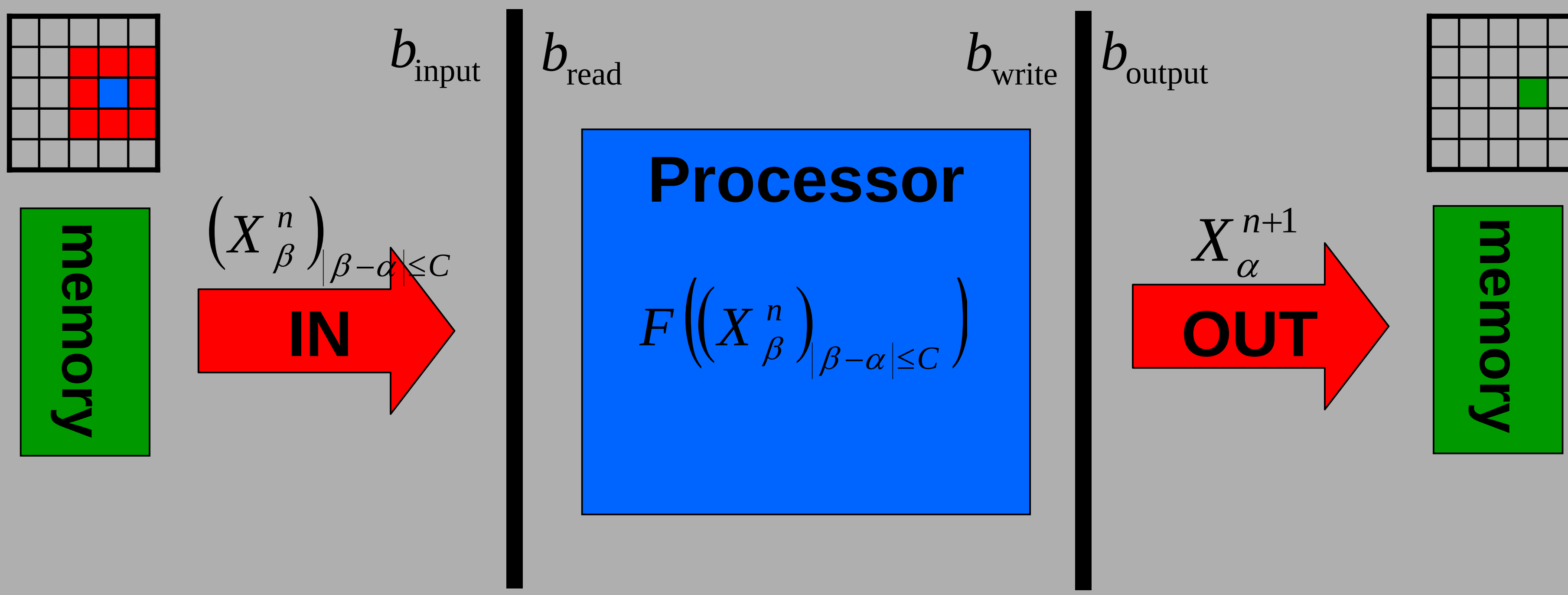
$$\sum_{\beta:|\beta-\alpha|\le C} W_{\alpha,\beta-\alpha} X_{\beta}^{n}$$

Step n+1



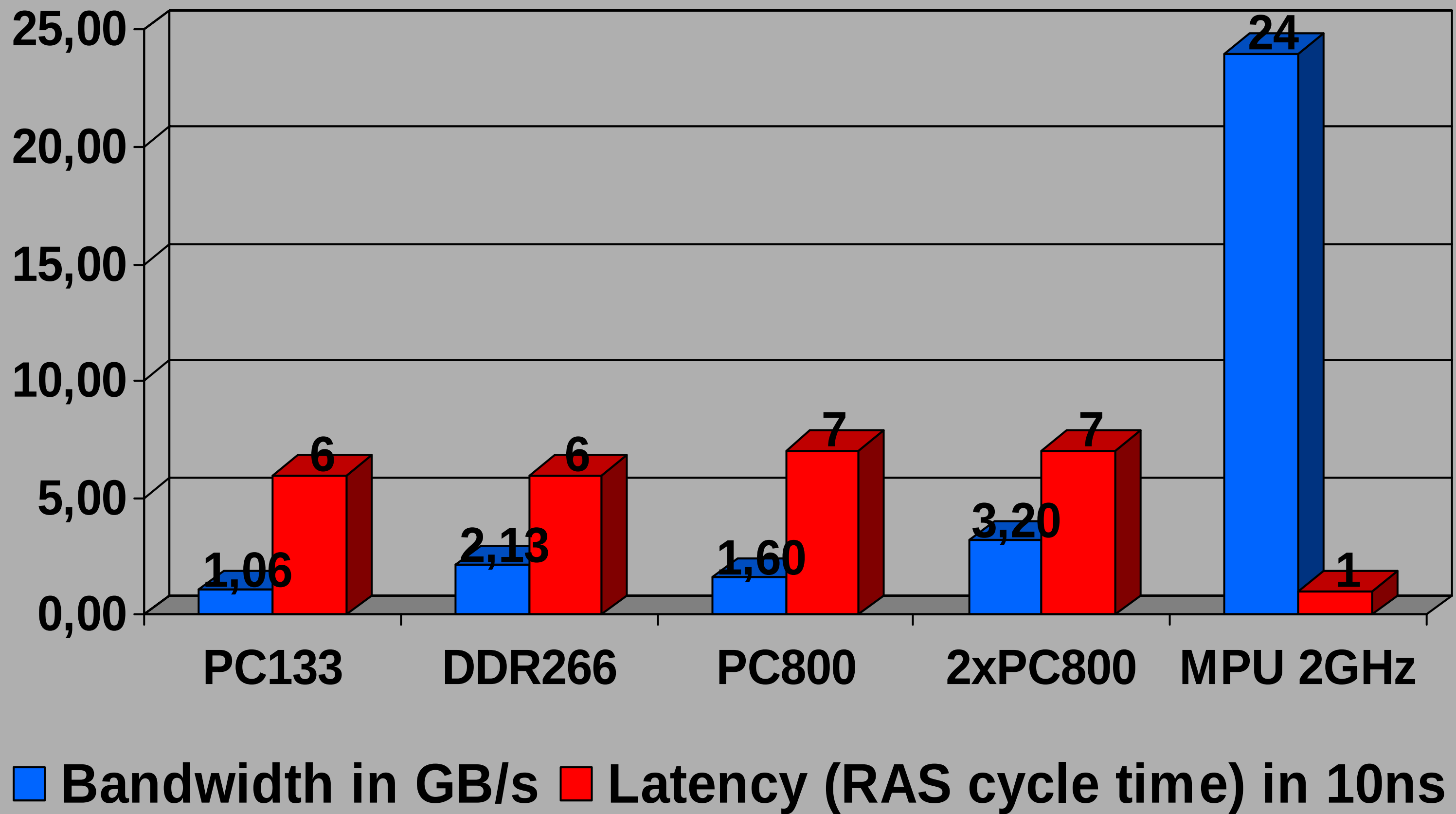
Data Processing-Bandwidth

$$\text{total - bandwidth } \; b_{\text{total}} = \min\left\{ b_{\text{input}}, b_{\text{read}}, b_{\text{write}}, b_{\text{output}} \right\}$$



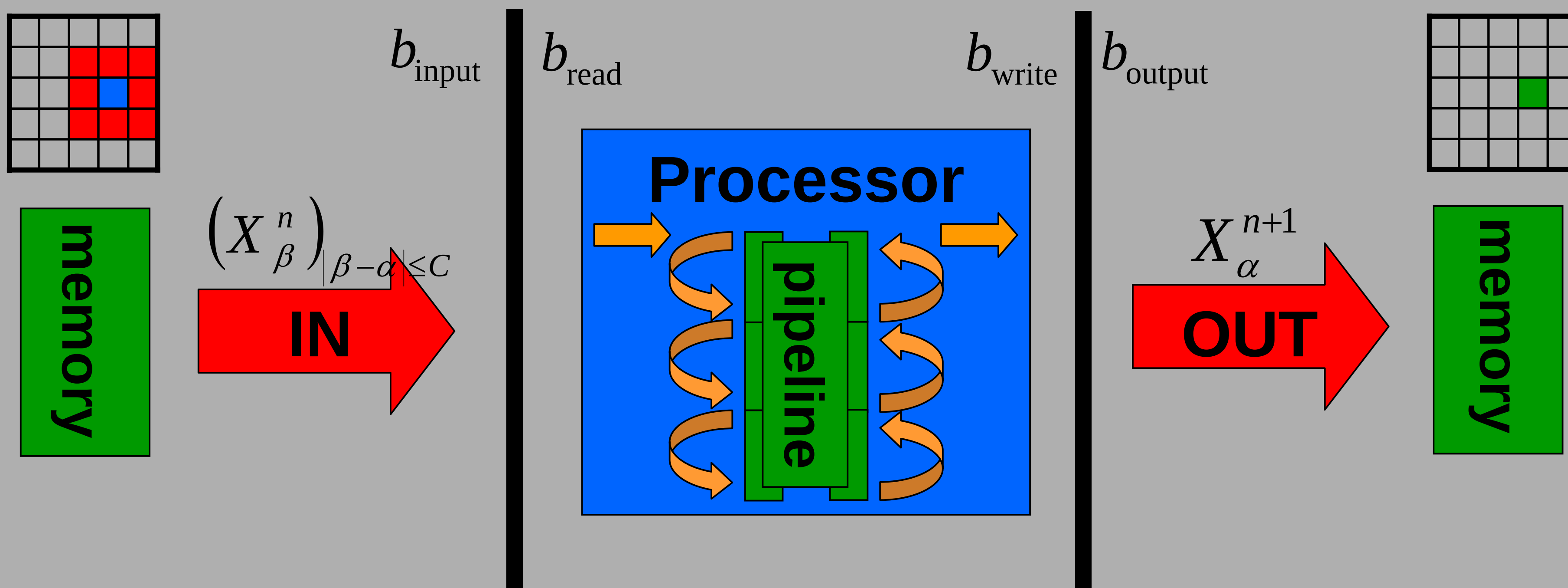
Task : Maximize total - bandwidth b_{total}

Current SD and RD RAM-Types for MPUs



Data Processing-Bandwidth

$$\text{total-bandwidth}\ \ b_{\text{total}} = \min\{b_{\text{input}}, b_{\text{read}}, b_{\text{write}}, b_{\text{output}}\}$$



Tasks: 1. Keep the whole pipeline busy

2. Maximize total-bandwidth b_{total}

Comparison XPP-FPGA

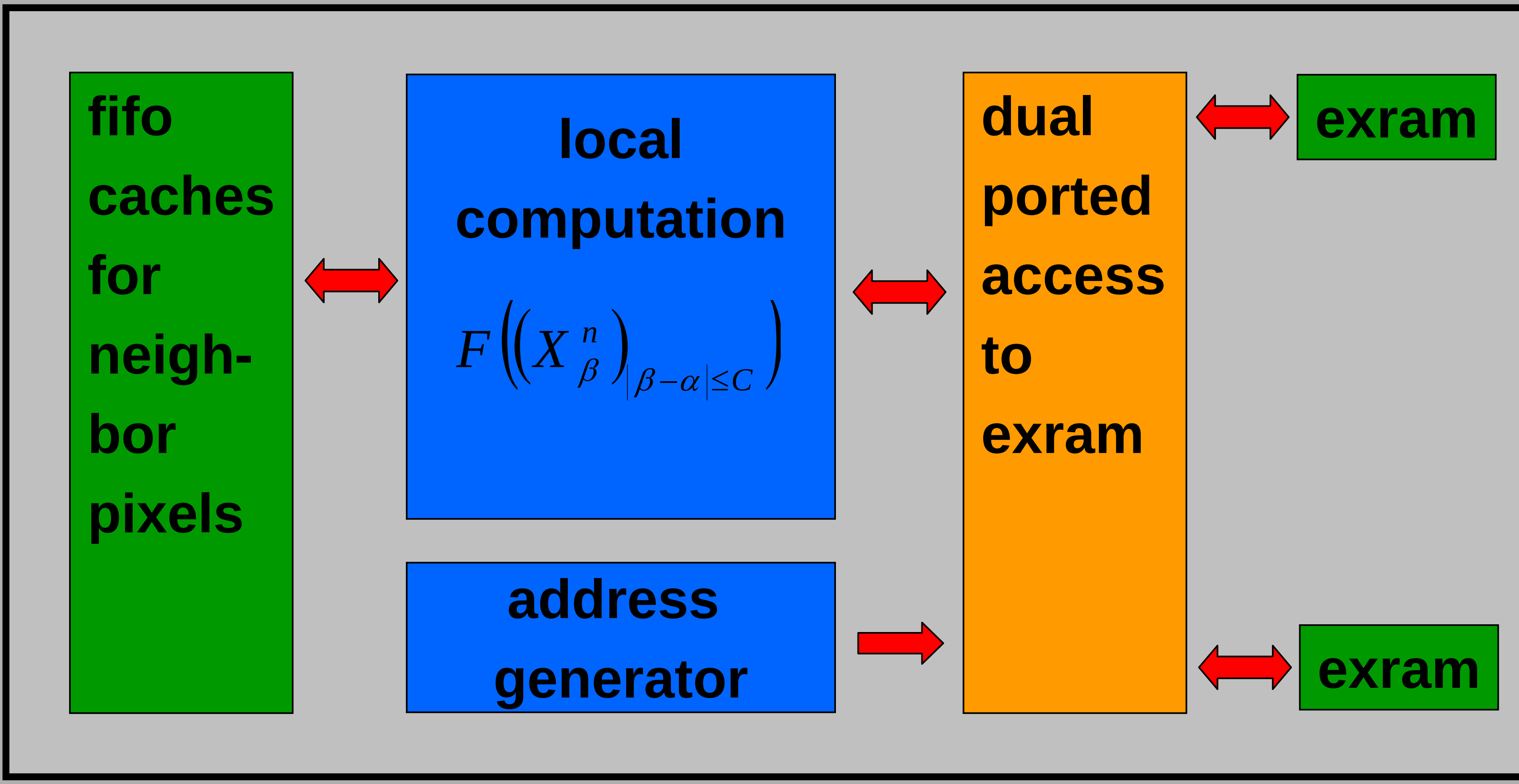
XPP	FPGA
+ **fast reconfigurability**	+ low level optimization
+ **implicit synchronization**	+ **large local memory with variable access**
+ higher level programming	+ **many IO channels**
+ **easier debugging**	

Overview

- Introduction
- Data-Flow & Architectures
- Implementations on the XPP
- Performance & Configurations
- Conclusions

Implementations on the XPP

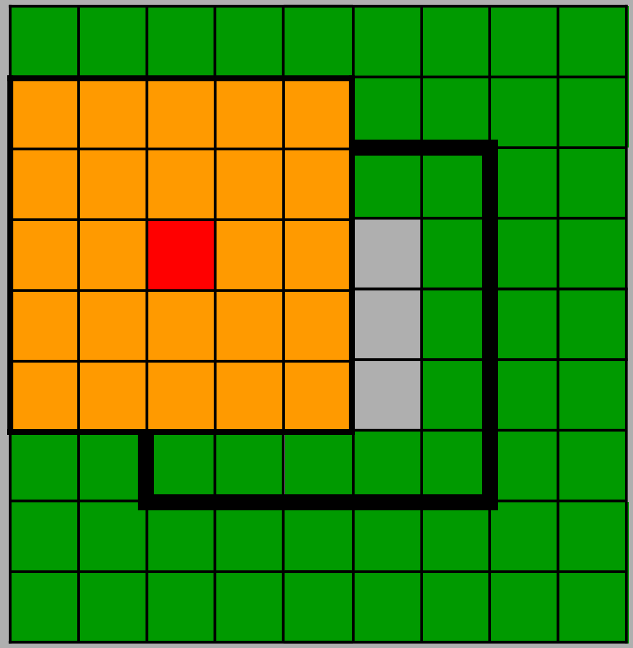
Aim : Maximal total - bandwidth,
i.e. : one input and output in each clock cycle



Boundry Conditions

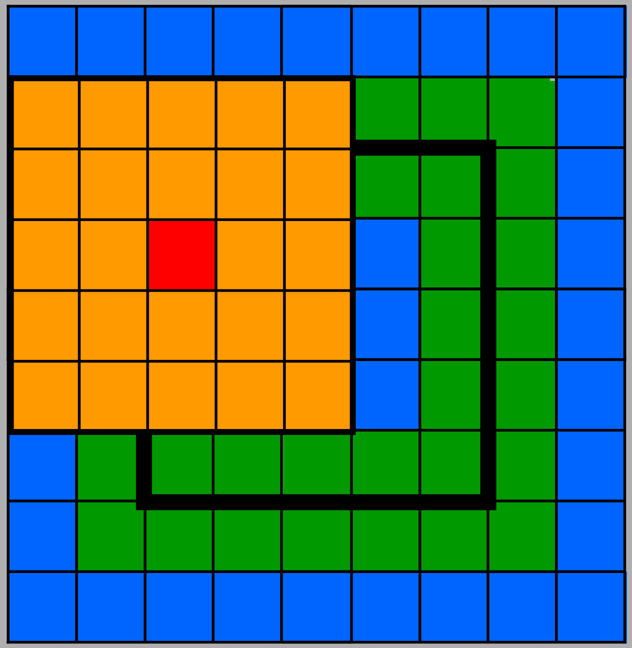
constant

boundry condition



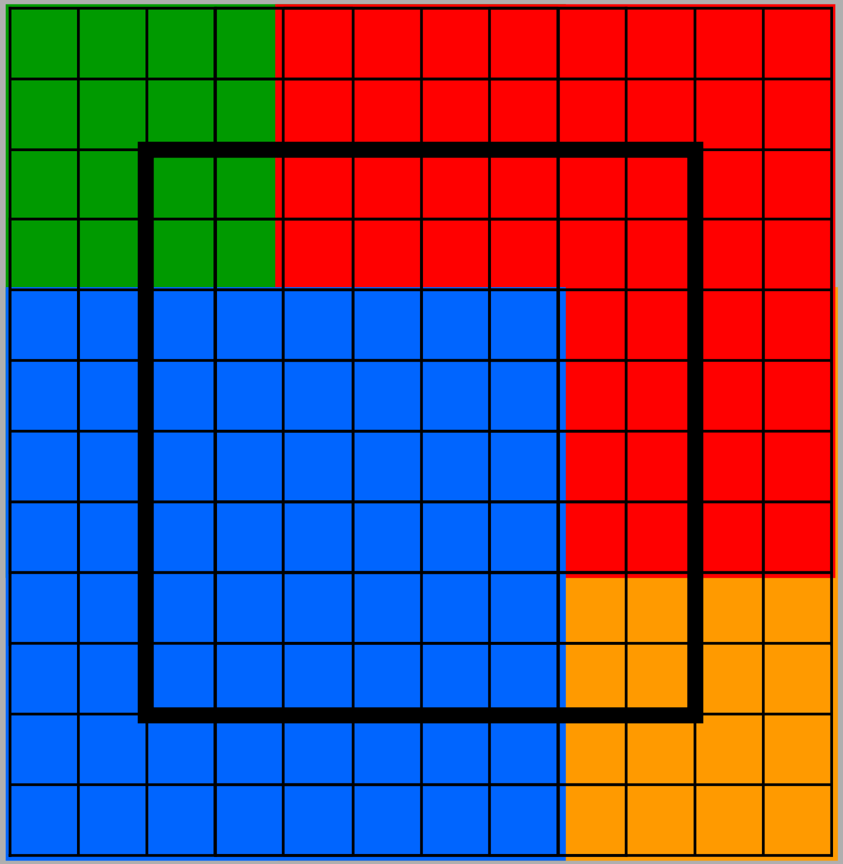
natural

boundry condition



Data Travers

If local memory is too small to cache all neighbour pixels, traverse the data in smaller subvolumes.



Costs: multiple transfer of border elements.

Implemented Filters in 2D and 3D in a 8+4x8 XPP array

$$X_{\alpha}^{n+1} = F\left(\left(X_{\beta}^{n}\right)_{|\beta-\alpha|\le C}\right) := \sum_{\beta:|\beta-\alpha|\le C} W_{\alpha,\beta-\alpha} X_{\beta}^{n}$$

2D Stencil	3D Stencil
3x3	3x3x3
5x5	5x5x5
	in array
7x7	10+4x15

Performance

	stencil 7x7 XPP 12x8	**stencil 3x3x3** XPP 12x8	**stencil 5x5x5** XPP 14x15
operations per clock cycle	49 MAC	27 MAC	125 MAC
output pixel per clock cycle	**1**	for 2^9 fifos **0.73**	for 2^9 fifos **0.58**
number of passes at 100MHz	256^2 data 1525	256^3 data 4.373	256^3 data 3.454

$\Rightarrow$ real-time for 2d applications

$\Rightarrow$ interactivity for 3d applications

Configuration for an explicit solver

for each timestep n {

configure the array for weight computation

compute weights for each $|\gamma| \le C$

$$W^{n}_{\alpha,\gamma} = G_{\gamma}\left(\left(X^{n}_{\beta}\right)_{|\beta-\alpha|\le C}\right)$$

configure the array for data computation

apply weights to data

$$X^{n+1}_{\alpha} = \sum_{\beta:|\beta-\alpha|\le C} W^{n}_{\alpha,\beta-\alpha} X^{n}_{\beta}$$

}

Configuration for an implicit solver

for each timestep n {

configure the array for weight computation

compute weights for each $|\gamma| \le C$

$$W_{\alpha,\gamma}^{n} = G_{\gamma}\left(\left(X_{\beta}^{n}\right)_{|\beta-\alpha|\le C}\right)$$

configure the array for data computation

for each iteration k {

apply weights to data

$$X_{\alpha}^{n+1,k+1} = \sum_{\beta:|\beta-\alpha|\le C} W_{\alpha,\beta-\alpha}^{n} X_{\beta}^{n+1,k}$$

}

}

Solving the weight transmission problem

1. Instead of pre-computing 27 weights $W_{\alpha,\gamma}, |\gamma| \leq 1$ for a 3x3x3 stencil, pre-compute only a smaller vector of intermediate results $\vec{w}_{\alpha}$ from which all the weights can be quickly evaluated $W_{\alpha,\gamma}\left(\vec{w}_{\alpha}\right)$.

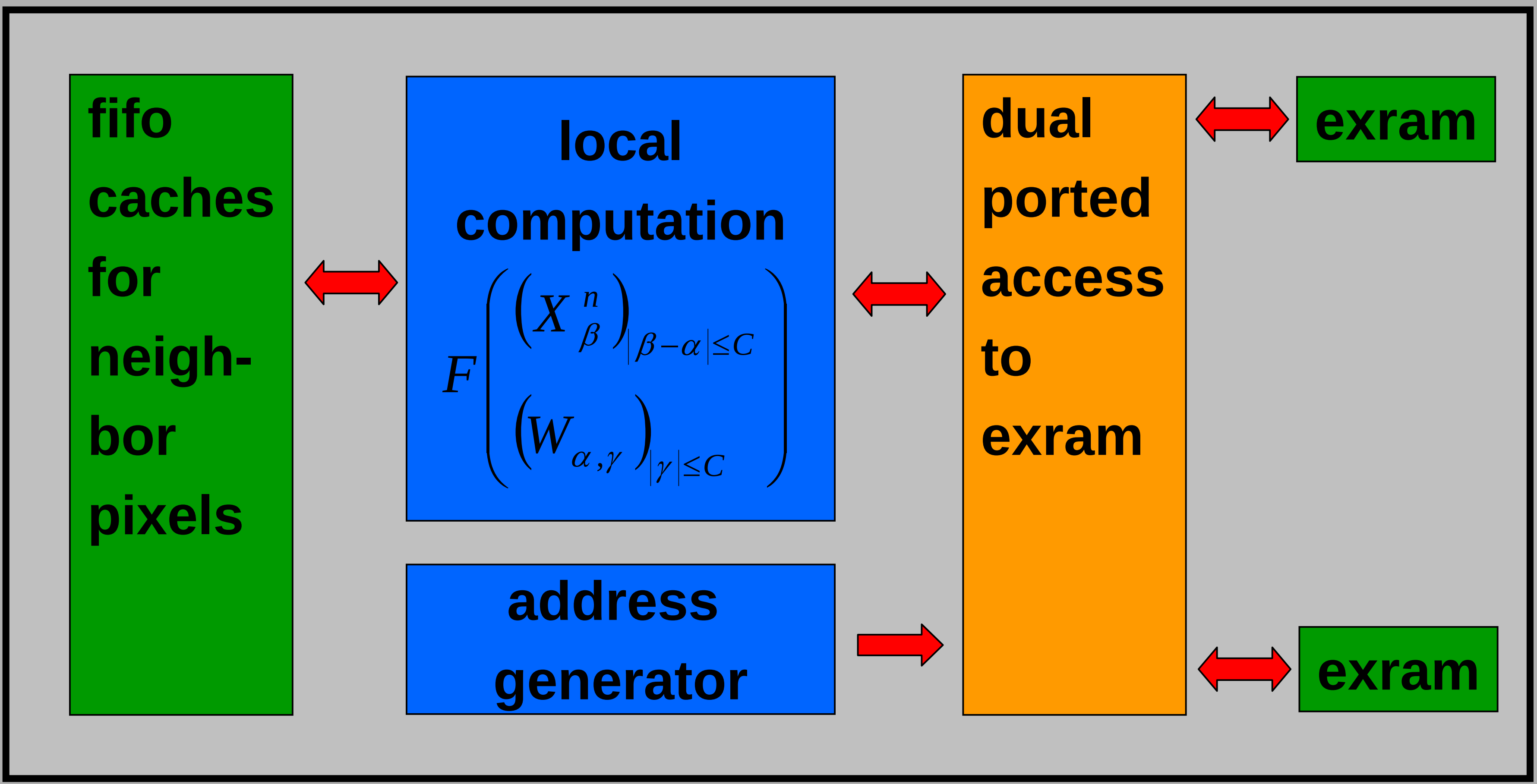
2. Increase the number of available IO channels by shifting the task of address generation and memory access to a processor outside of the XPP array, such that all the available 8 IO channels can be used for data input or output.

3. Increase the overall number of IO channels, such that applications will be able to access more than 6 intermediate results simultaneously.

Solving the weight transmission problem

Aim : Maximal total - bandwidth,

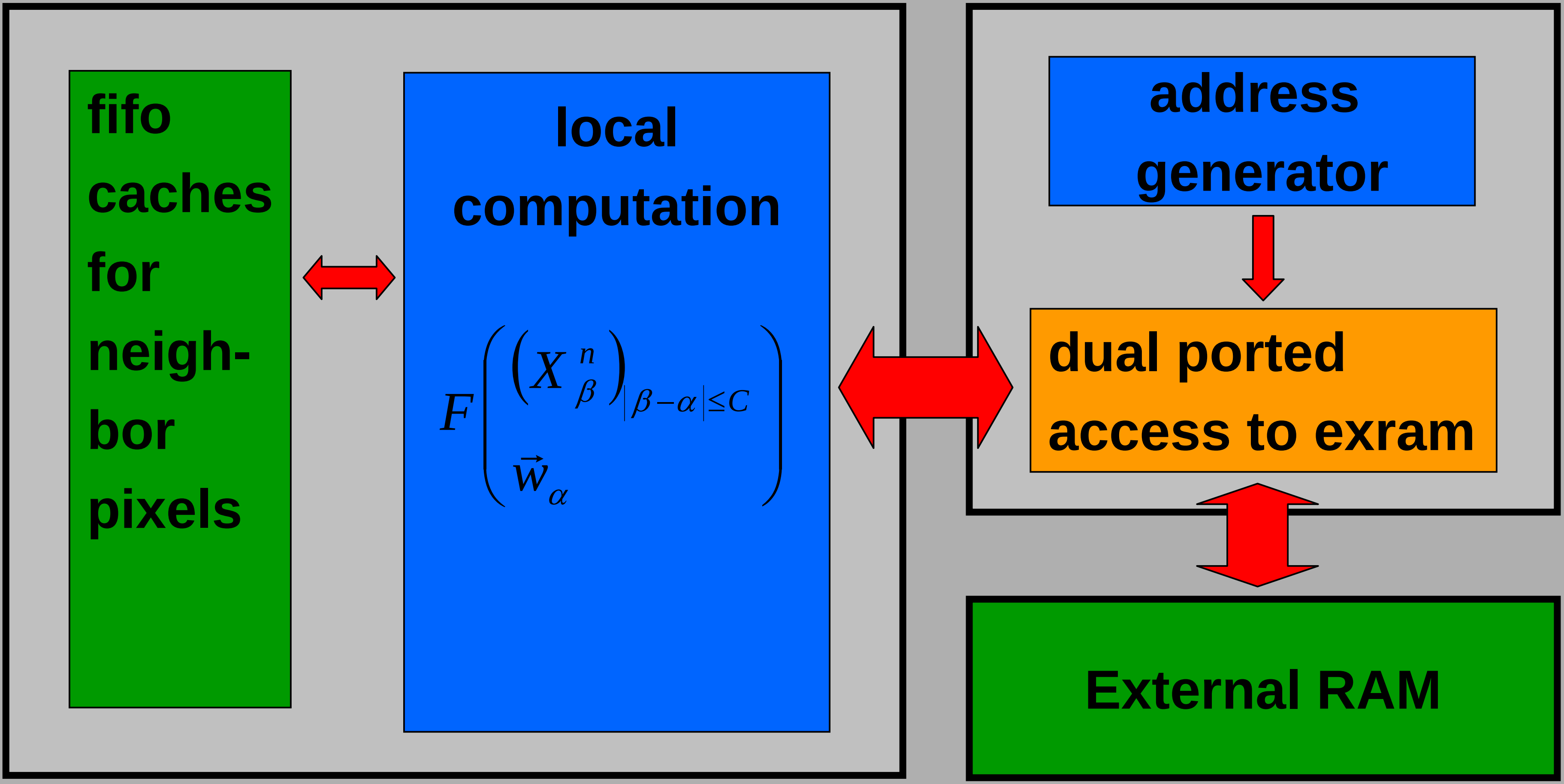
i.e. : one input and output in each clock cycle



Solving the weight transmission problem

Aim : Maximal total - bandwidth,

i.e. : one input and output in each clock cycle



Overview

- Introduction
- Data-Flow & Architectures
- Implementations on the XPP
- Performance & Configurations
- Conclusions

Conclusions

- A wide range of image processing application could be accelerated.
- The test implementations at estimated 100MHz suggest a performance gain of 10-20 over common PC solutions in full-grown applications.
- In our experience the XPP wins over other architectures such as GPUs or FPGAs either in speed or programmability.
- Finally, improved memory availability and IO access would further facilitate and accelerate image processing on the XPP.