Filtrage d’images rapide sur GPU

Raphaël Couturier

IUT Belfort-Montbéliard – Institut FEMTO-ST – Université Bourgogne Franche-Comté

Overview of my talk

- Introduction on GPU
- Some elements on the GPU programming
- Results on:
  - Median filter
  - Convolution
  - PRNG
- Questions?
- Elements on fast GPU imaging algorithms
- More details on:
  - Median filter
  - Convolution
  - PRNG
- Questions?

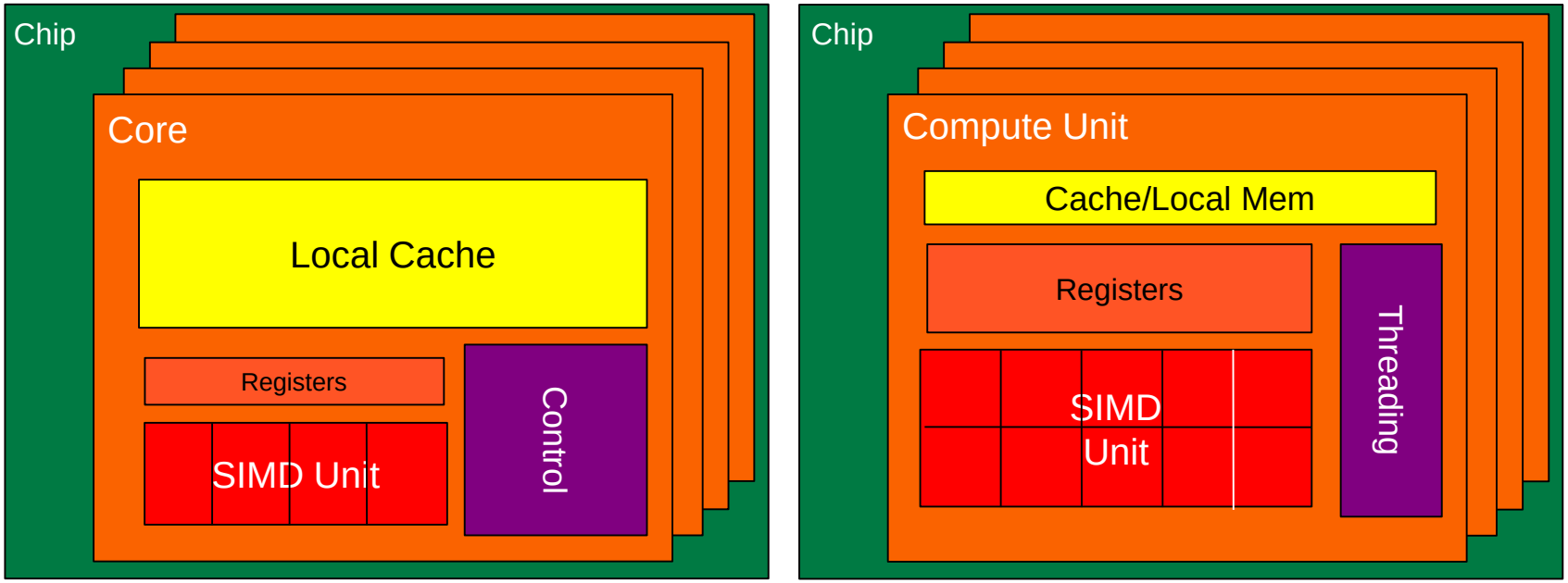
Some generalities about GPUs

- GPU : video card that can be used to make parallel computation
- For many kinds of applications
- Data parallelism model
- Number of cores: up to 5000
- Most hot topic domain: Deep learning

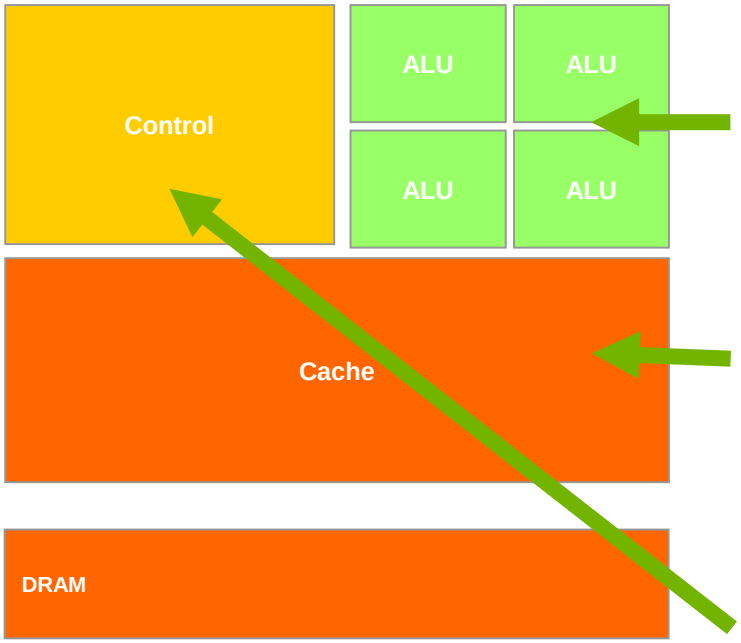
CPU and GPU are designed very differently

CPU
Latency Oriented Cores

GPU
Throughput Oriented Cores

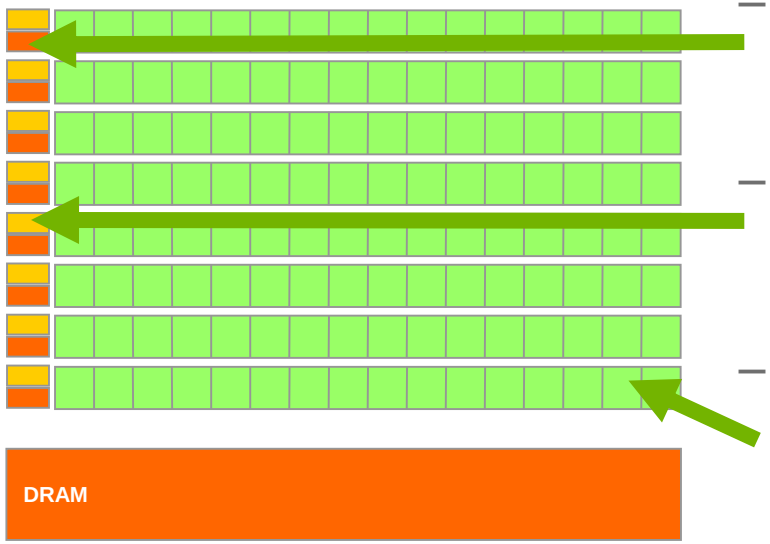


CPUs: latency oriented design



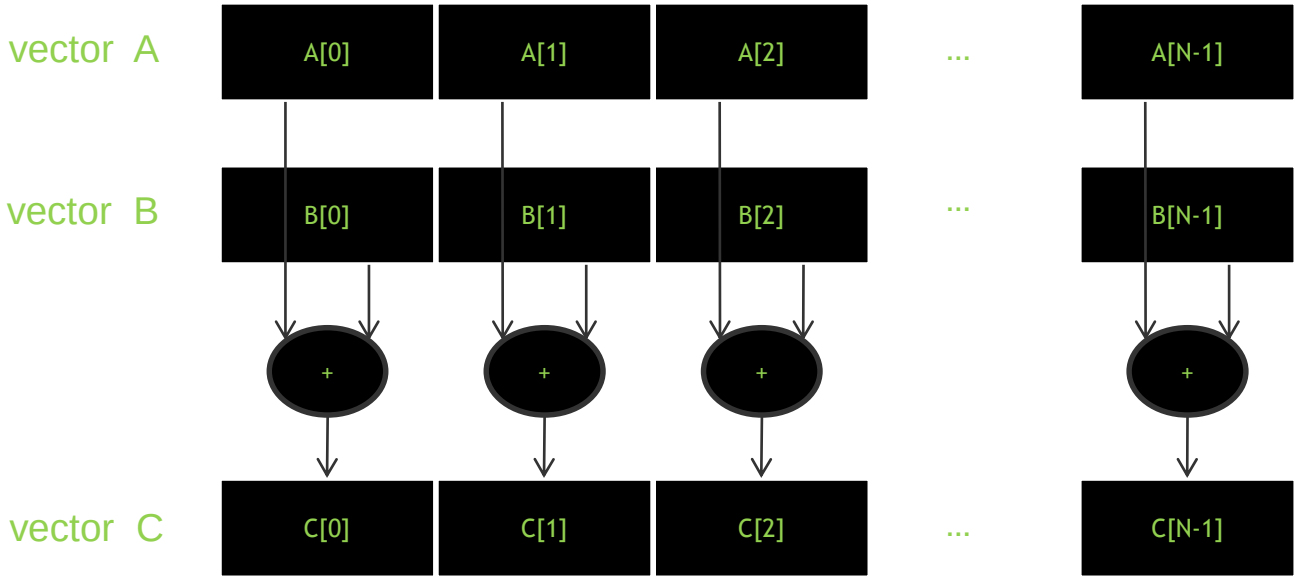
- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

GPUs: throughput oriented design



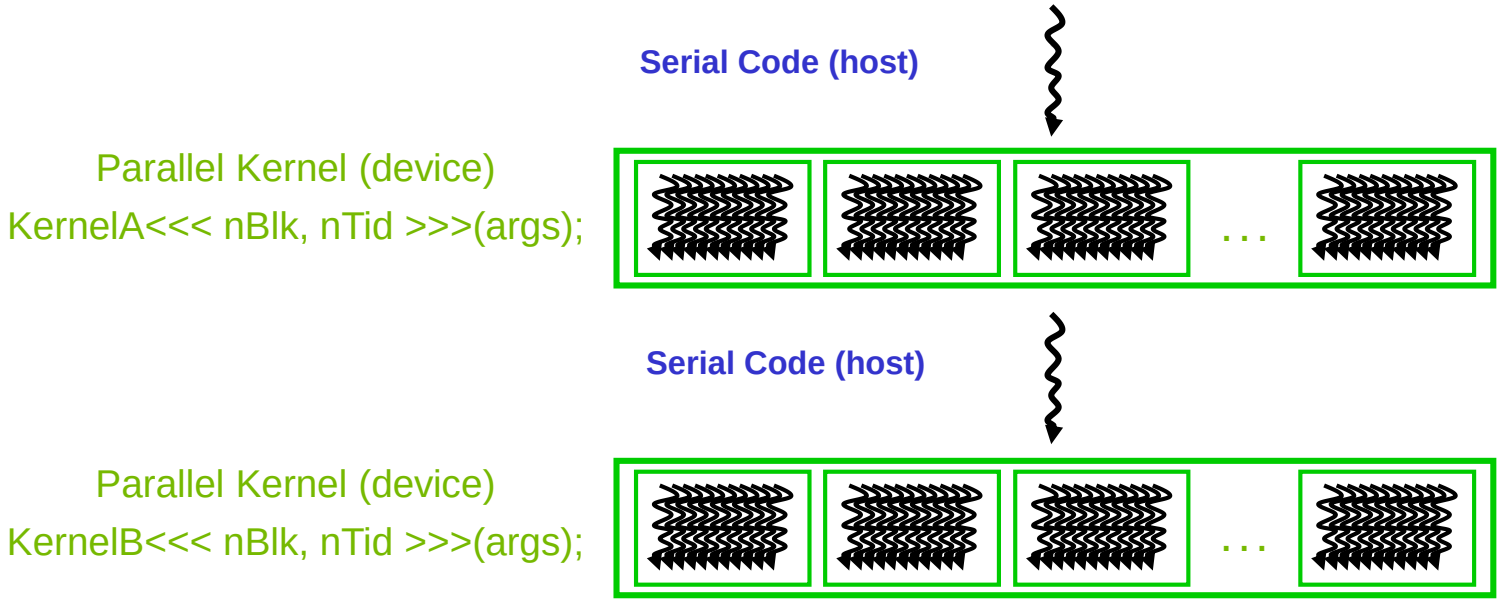
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies
  - Threading logic
  - Thread state

Data parallelism: vector addition example

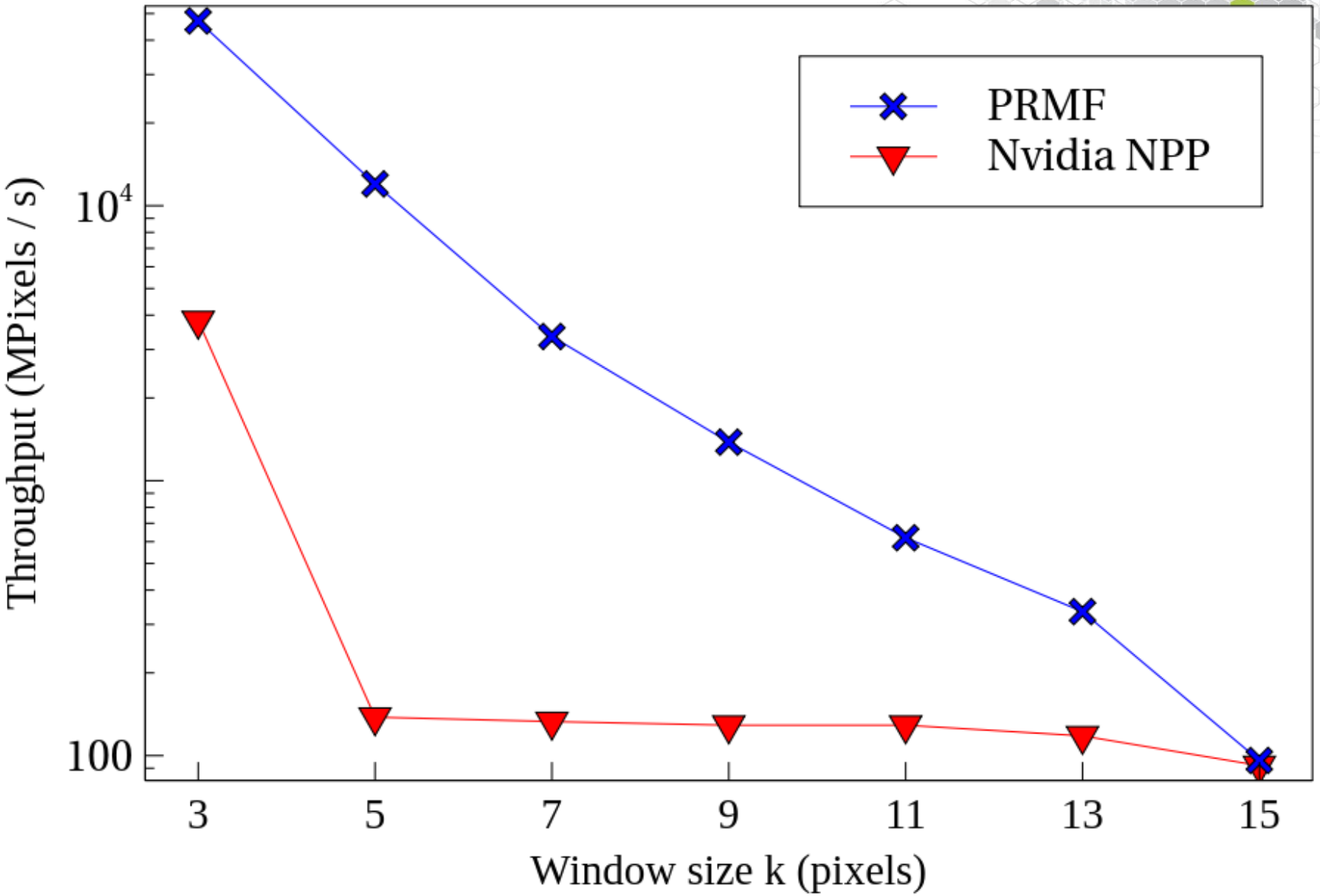


CUDA execution model

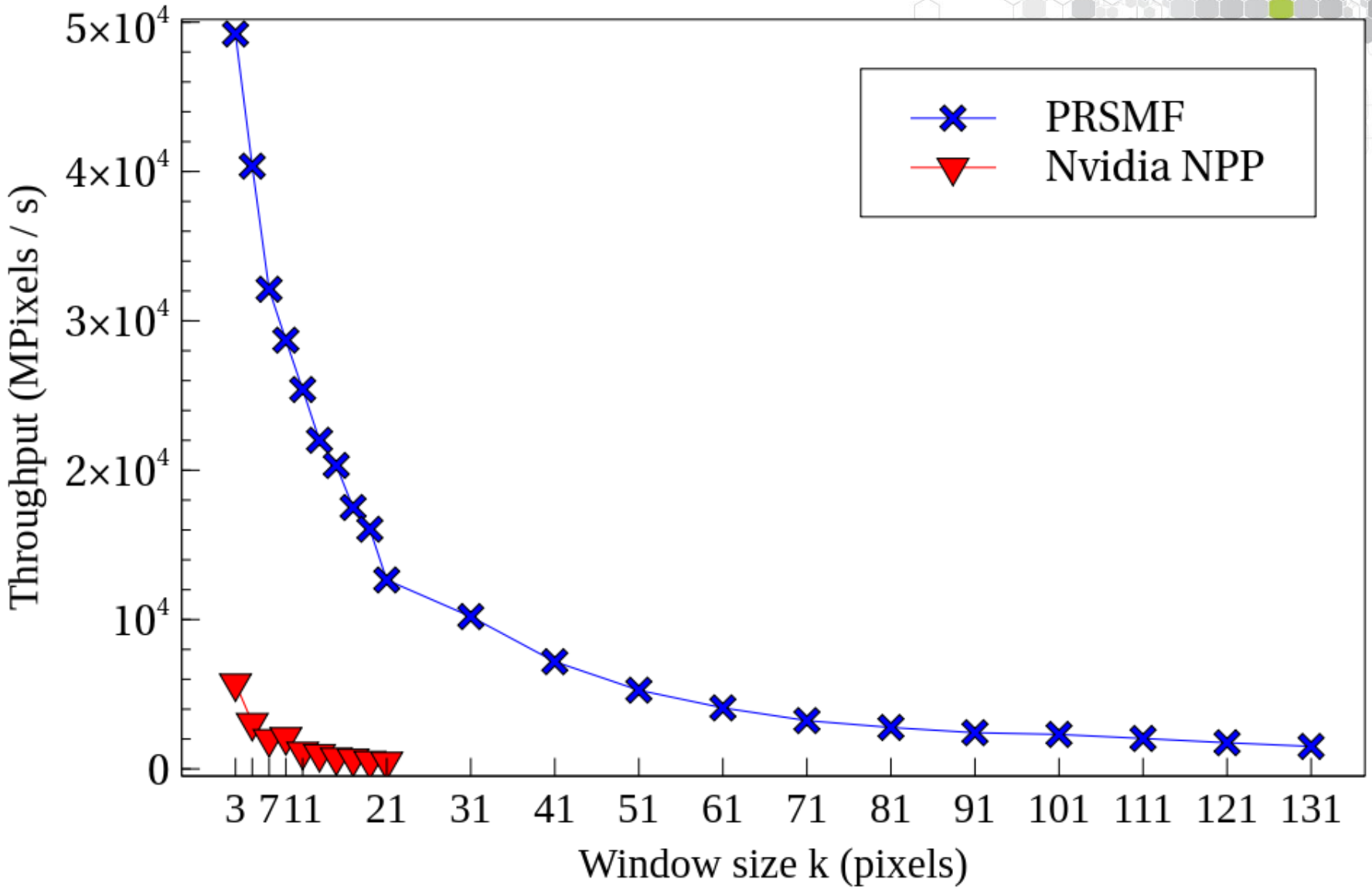
- Heterogeneous host (CPU) + device (GPU) application C program
  - Serial parts in **host** C code
  - Parallel parts in **device** SPMD kernel code



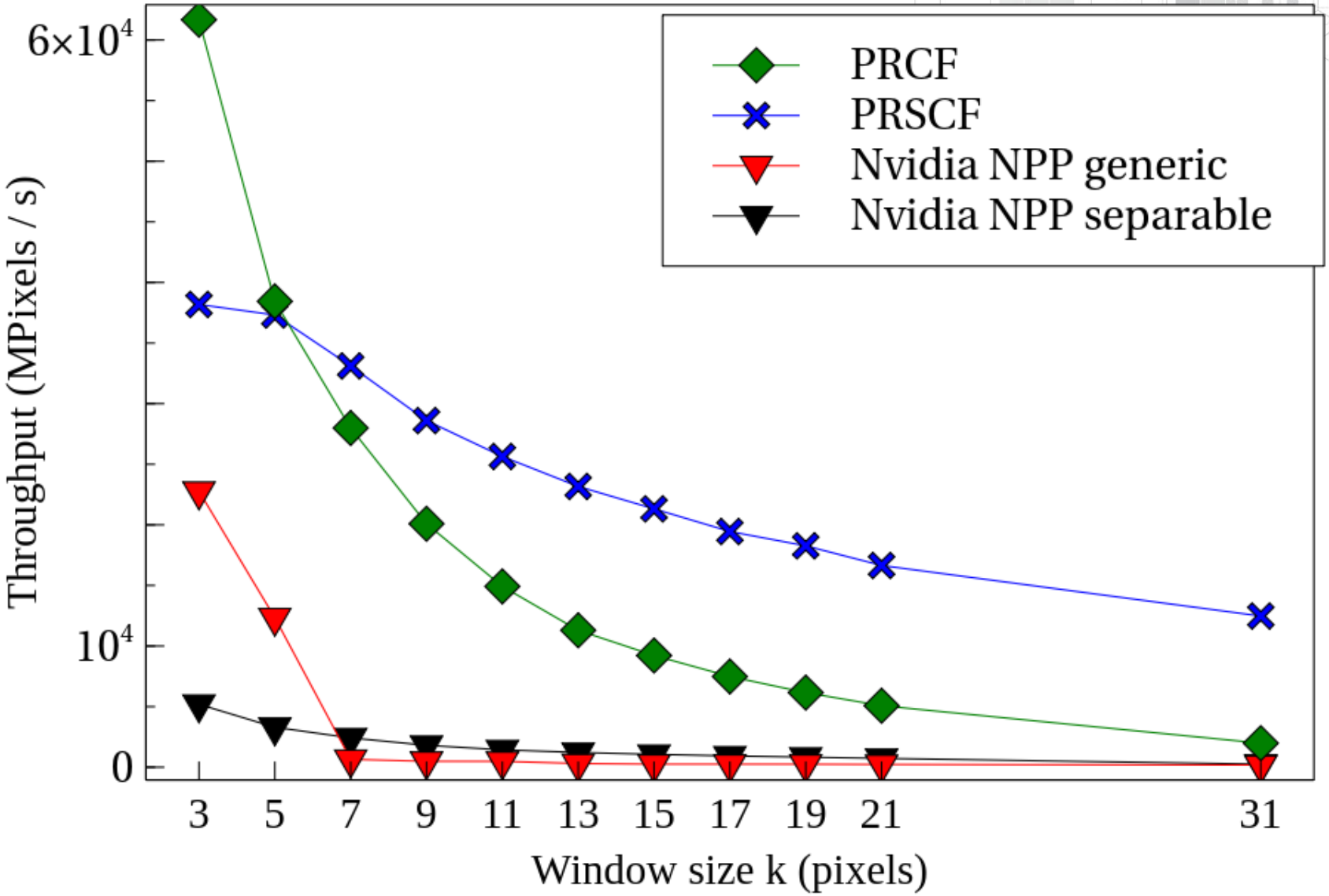
Result on median filter (full)



Separable median



Result on convolution filter



Results on pseudo random number generator

- Used in many applications: security, monte carlo applications, etc
- Example on a kernel used to make a stream cipher, application on images
- All the tests of TestU01 are successful
- All classical tests on image are also successful

Image size	ESSENCE Throughput (in GB/s)	AES-ECB Throughput (in GB/s)
512x512x3	35.1	20.3
1024x1024x3	71.5	36.6
2048x2048x3	105.7	52.1
4096x4096x3	115.7	58.3
8192x8192x3	108.6	65.8
16384x16384x3	110.6	70.2

Questions ?

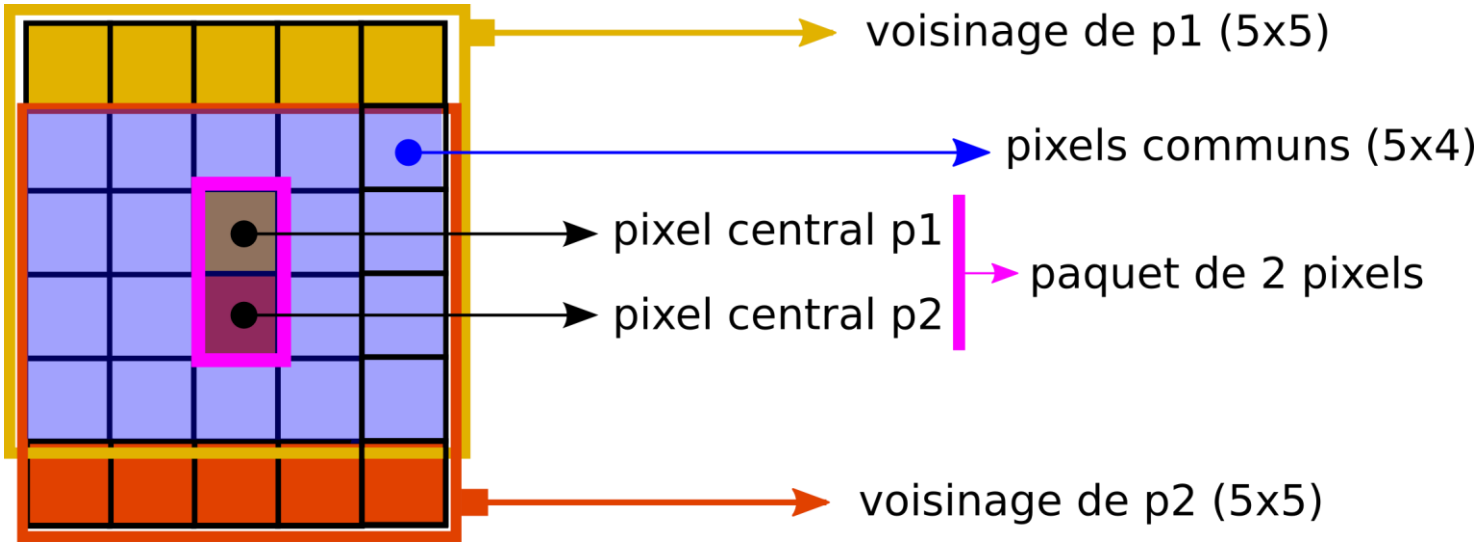
raphael.couturier@univ-fcomte.fr

How to generate efficient kernels?

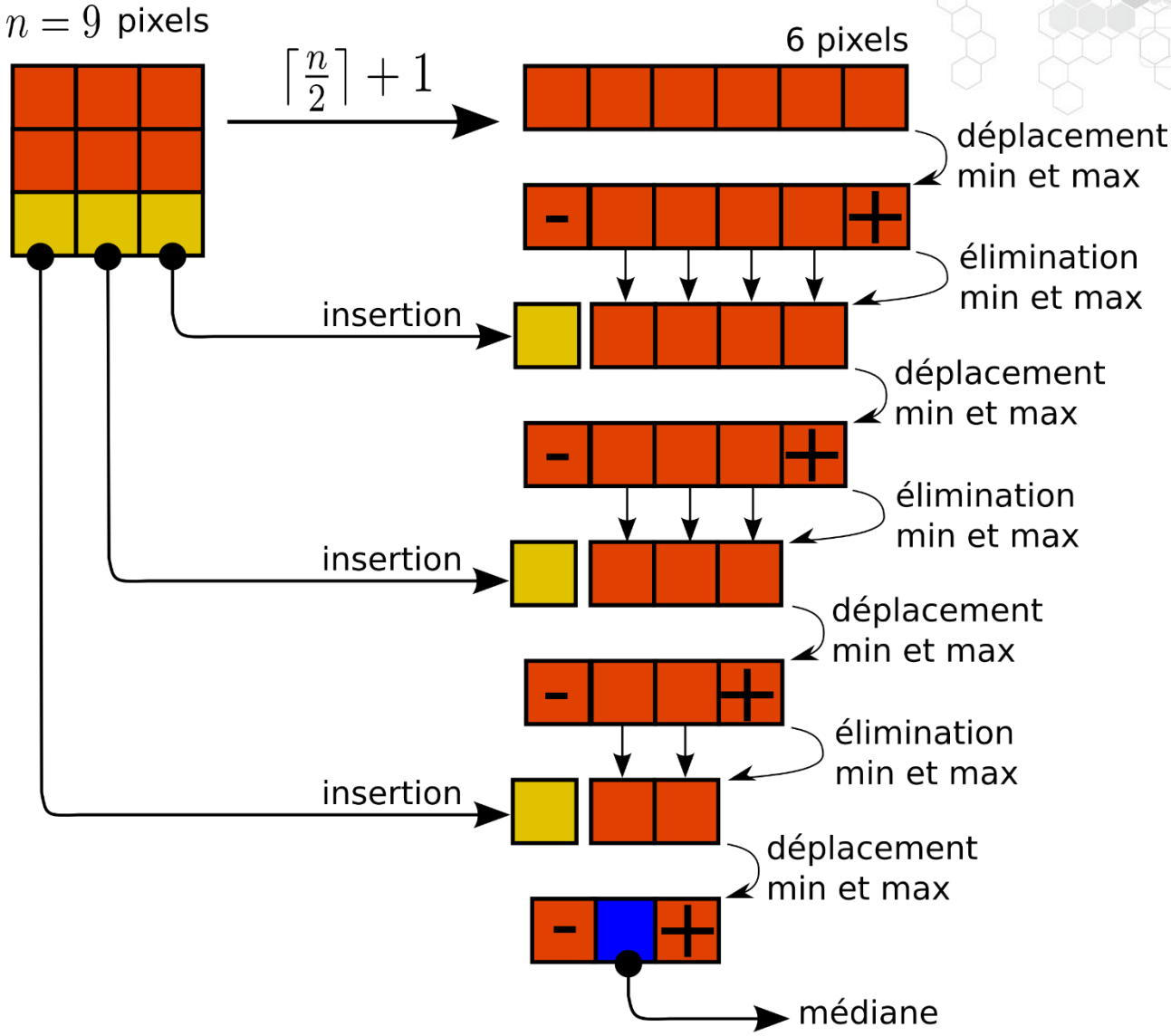
- Limitation of CPU-GPU communication
- Reducing the number of divergent branches
- Reducing the number of synchronizations
  - Only synchronization between threads in the same block
  - No synchronization between blocks
- Take care of memory access (depend on the harware)
- If possible use register
- Most of the time, don’t use too much shared memory
- Use small very specific kernels
- Necessity to write code generator ☺

Median filter

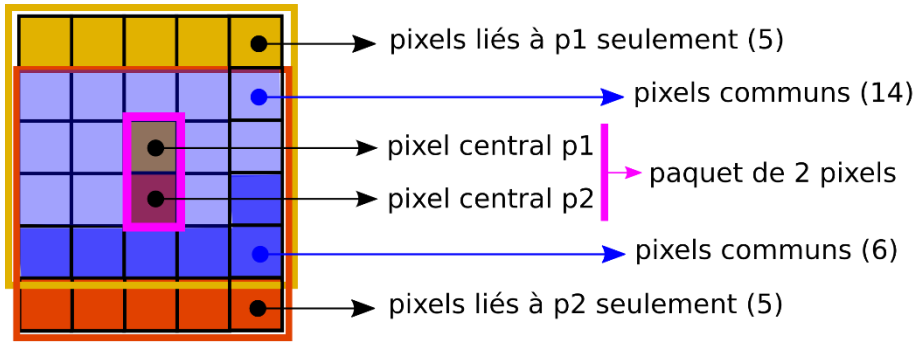
- Load of neighbor pixels in registers
- In order to take the median filter, we don’t use sort algorithm
- Selection filter
- Use of overlapping between adjacent pixels
  - Reducing the number of register => increasing of parallelism
  - Shared part of selection filter => less computation
  - Use of vertical pixels => contiguous memory access

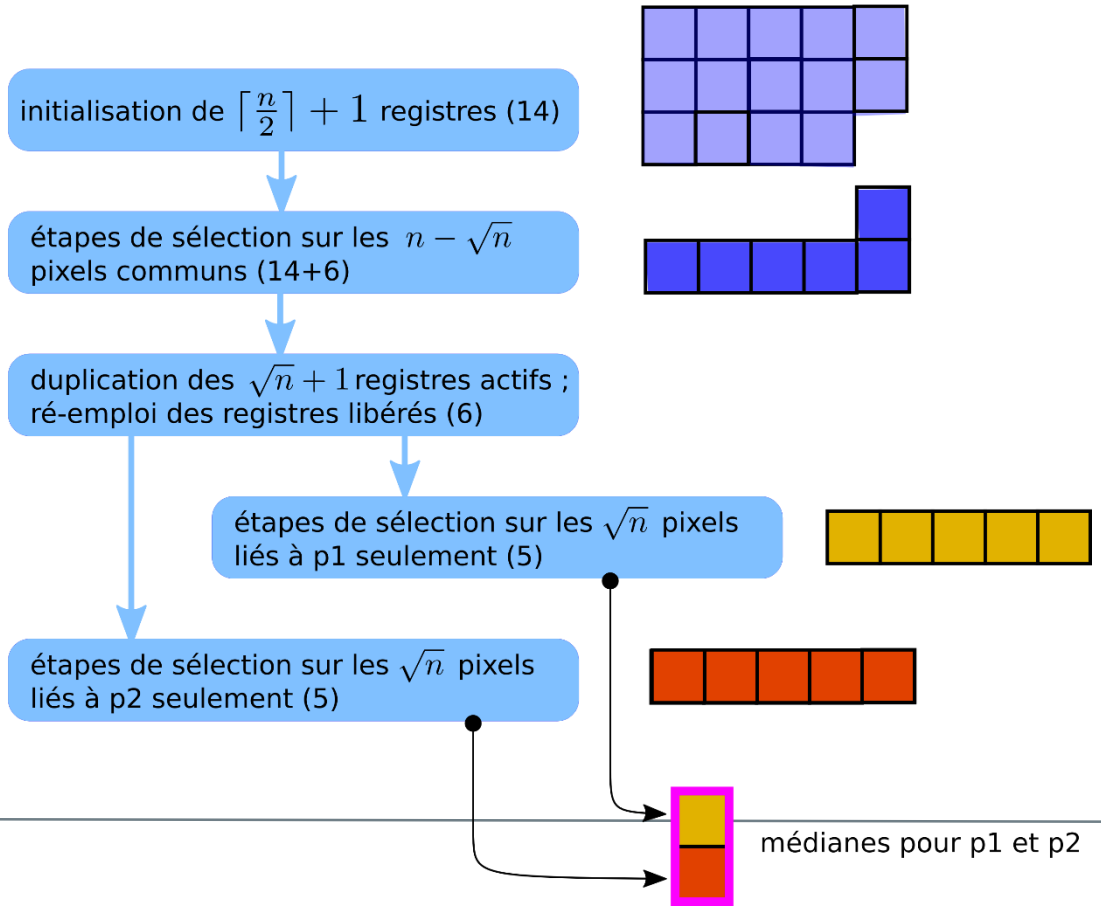


Median filter: selection filter



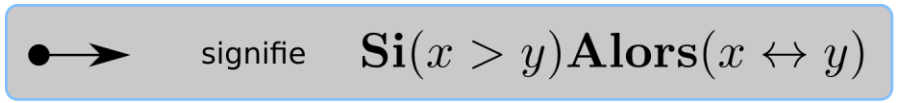
Median filter: use of overlapping

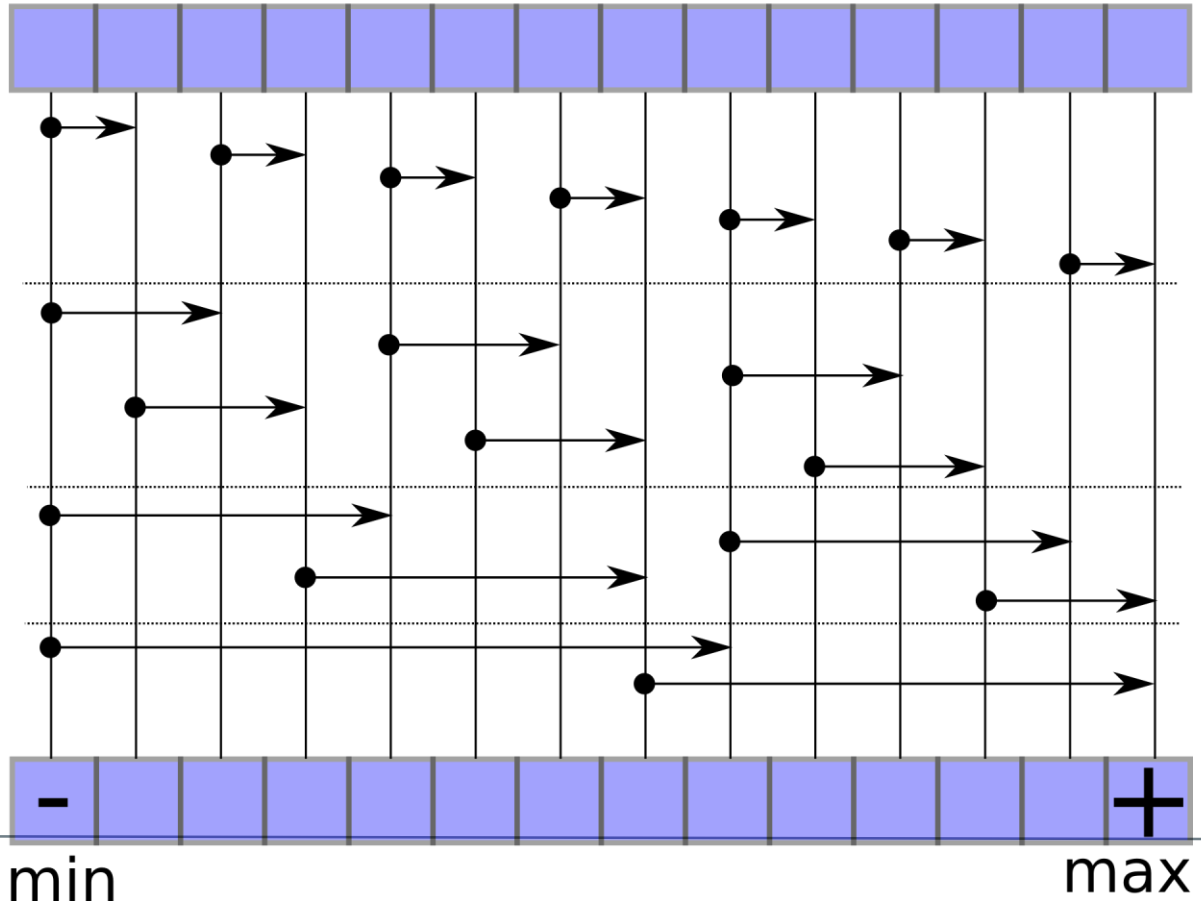




Median filter: maximizing parallelism (ILP)

- ILP: instruction level parallelism, example with 5x5 => 14 values





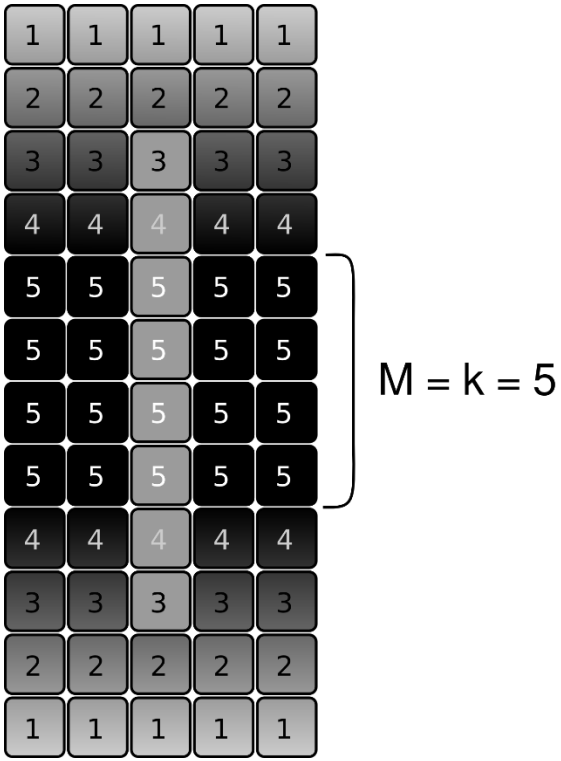
Convolution

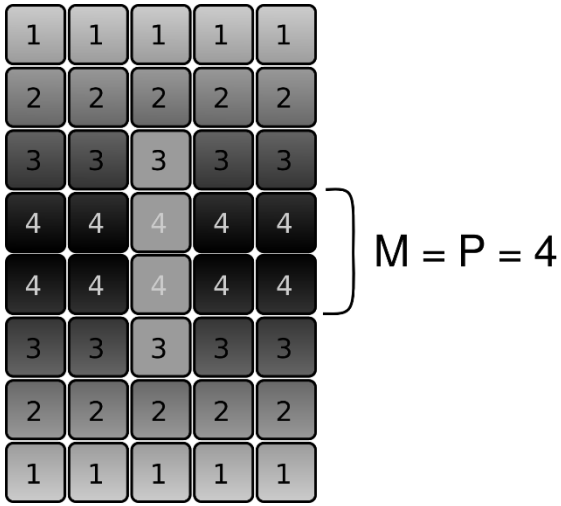
Principles:

- Only the case of impair windows size considered
- Use dedicated kernel
- One kernel for each size
- Multiple pixels per thread
- Use only register
- Use of ROI (region of interest)
- Generator of code because a human cannot write such codes

Convolution

- ROI of pixel packets
- Vertical packets ensure better memory accesses





PRNG: Naive option

- Each thread is responsible to compute a single PRNG
- Simple to implement
- Quite efficient
- But with several PRNG (even PCG), this fails to succeed with TestU01 tests

PRNG: improved version

- Use multiple PRNG (PCG32 and XORSHIFT)
- Mix their result using shared memory
- Compute multiple elements per thread
- Succeeded to pass all the TestU01 tests
- Only 20 lines of code in the kernel: paper in finalization

Questions ?

raphael.couturier@univ-fcomte.fr