Change Detection Algorithm with Hardware Constraints

Prof. Ingo Sander[1], Postdoc Marcello Costa[2]
October 31, 2024 — [1]KTH Royal Institute of Technology, [2]Cisb-Saab

Contents

"Parallelized n-lags Change Detection Model for rapid data analysis":

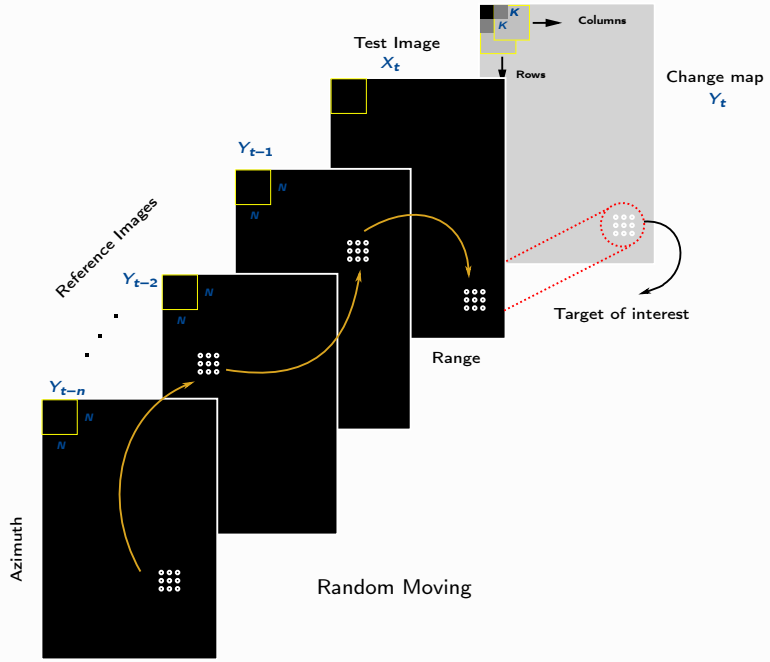
1. **Elementary Change Detection Model (bi-temporal):** → Low complexity-power-memory demands, and parallelized structure[1] (using Synchronous MoC)
2. **Temporal dataset: Ultra-wideband (UWB) SAR** → stable scatters in time, possible bi-temporal and multi-temporal analysis (We are assuming random motions of targets in the clutter as a stationary AR(1) process)
4. **Generalization** → change model extension: n-AR(1) in a Markov-chain, $n = 1, \ldots, N$ lags

[1] Rabhi *et al.*, "Patterns and skeletons for parallel and distributed computing," Springer Science & Business Media, 2003.

[2] Wu, Suya, et al. "Quickest Change Detection for Unnormalized Statistical Models," *IEEE Trans. on Inf. Theory*, v. 70, n. 2, pp. 1220-1232, 2024.

Change Detection System Model

Problem Overview:



Time Series approach:

$$\hat{Y}_t = f(X_t, Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}) + \epsilon_t,$$

Anomaly Detection Model:

$$\hat{Y}_t = \underbrace{\mu_Y}_{\text{clutter} \rightarrow \text{clutter}} + \underbrace{X_t}_{\text{clutter} \rightarrow \text{target}},$$

$$\begin{cases} Y_t = & aY_{t-1} + X_t \\ Y_t = & a^n Y_{t-n} + \sum_{i=0}^{n-1} a^i X_{n-i} \end{cases}$$

a is the lag-one autocorrelation between Y_t and Y_{t-1} and n is the time lag

Auto-regressive first order Model - AR(1)[3]: Detects uncorrelated samples between y_{t-1} and x_t as a predicted anomaly to compose the change map y_t, which describes the degree of retained memory of previous states.

$$\underbrace{\hat{y}_t[n,m]}_{\text{changes}} = \underbrace{\hat{\rho}}_{\text{corr.}} \underbrace{y_{t-1}[n,m]}_{\text{ref. signal}} + \sqrt{1-\hat{\rho}^2}\ \underbrace{x_t[n,m]}_{\text{test signal}}, \hat{\rho} \in [0,1] \tag{1}$$

$\Rightarrow$ Assuming 1-lag, the red noise sequence $\hat{y}_t \sim \mathcal{N}(0,1)$ is produced from white noise sequence x_t, assuming the random motion of anomalies on clutter; ρ is the cross-correlation between $y_{t-1}[n,m]$ and $x_t[n,m]$, normalized in the interval $(0,1)$.

[3] M. Davis and R. Vinter. "Stochastic Modelling and Control, Monographs on Statistics and Applied Probability," *Springer Netherlands*, 1985.

n-lags Model

Def. Markov chain: A stochastic process $\{X_n : n = 0, 1, 2, \dots\}$ is called a Markov chain if it has the following property:

$$\begin{aligned} &P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n1}, \dots, X_2 = i_2, X_1 = i_1, X_0 = i_0) \\ &= P(X_{n+1} = j | X_n = i) \end{aligned}$$

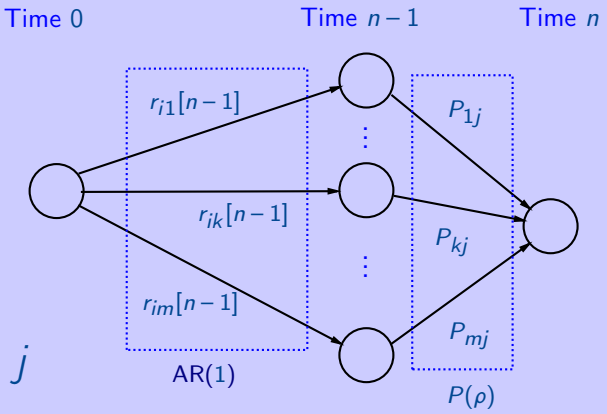
for all states $i_0, i_1, i_2, \dots, i_{n1}, i, j \in \mathcal{X}$ and $n \geq 0$.

$\Rightarrow$ Markov Chain is a stochastic process such that given the value of the current state, the distribution of the next state is independent of the past.

n-step transition probabilities:

$$r_{i,j}[n] = \sum_{k=1}^{m} r_{i,k}[n-1]P_{k,j}$$

$$\begin{cases} r_{i,j}[n] \rightarrow & \text{Prediction going to state } i \text{ to } j \\ r_{i,k}[n-1] \rightarrow & \text{recursion from state } i \text{ to } k \\ P_{k,j} \rightarrow & \text{Transition probability from state } k \text{ to } j \end{cases}$$



$$P = \begin{pmatrix} P_{0,0} & P_{0,1} & P_{0,2} & \cdots & P_{0,j} & \cdots \\ P_{1,0} & P_{1,1} & P_{1,2} & \cdots & P_{1,j} & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \\ P_{k,0} & P_{k,1} & P_{k,2} & \cdots & P_{k,j} & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \end{pmatrix} \rightarrow \text{transition probability matrix}$$

Random walk on integers: given $X_n, X_{n-1}, X_{n-2}, \ldots$ the distribution of X_{n+1} depends only on X_n but not $X_{n-1}, X_{n-2}, \ldots$ The state space is $\mathcal{X} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\} \in \mathbb{Z}$

$$X_{n+1} = \begin{cases} X_{n+1}, & \text{with } p \\ X_{n-1}, & \text{with } 1-p \end{cases}$$

The transition probability is

$$P_{i,j} = \begin{cases} p, & \text{if } j = i+1 \\ 1-p, & \text{if } j = i-1 \\ 0, & \text{Otherwise} \end{cases}$$

$$P = \begin{pmatrix} \ddots & & & & & & \\ & 0 & p & & & & \\ & 1-p & 0 & p & & & \\ & & 1-p & 0 & p & & \\ & & & 1-p & 0 & p & \\ & & & & 1-p & 0 & \\ & & & & & & \ddots \end{pmatrix}$$

$\Rightarrow p = \rho_{x,\hat{y}}$.

Proposition: n-lags 2D-AR(1) Model on Markov-Chain.

Let $r_{i,j}[n]$ be the prediction extracted from a Markov chain. For change detection, Let $\tilde{r}^{\ell}_{i,j}[n]$ denote the set of interest involving AR(1) subject to the lowest probability. Then we will the unique sequence

$$\tilde{r}^{\ell}_{i,j}[n] = \bigcup_{i=0}^{\ell-1} \min\{r_{i,j}[n] \backslash \tilde{r}^{\ell}_{i,j}[n]\}, \tag{2}$$

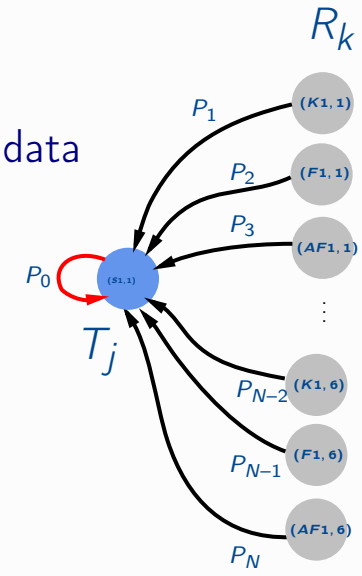
which represents the most likely available lags to detect changes anomalies in $r_{i,j}[n]$ with minimum false alarms.

□

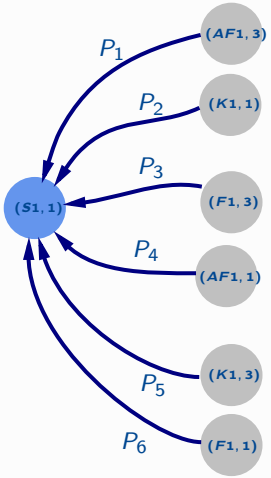
Change Detection System Model

R: reference data

T: test data

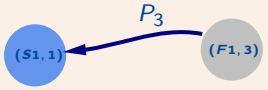


Best pairs



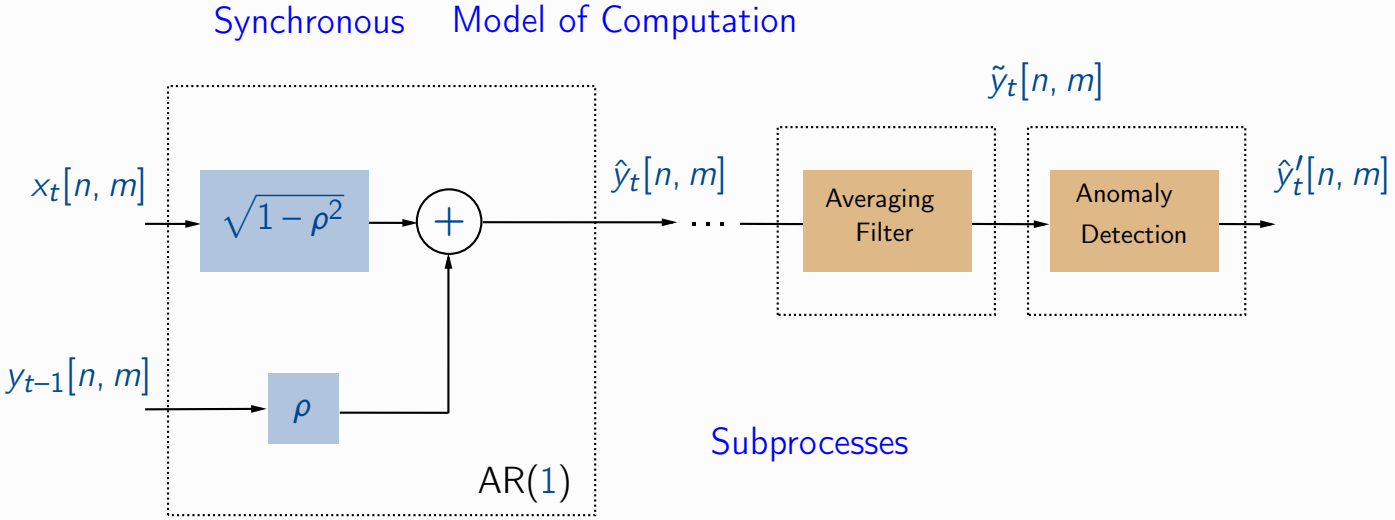
Decision Criteria

Min: $\hat{I}_{CD} = \tilde{r}^1_{i,j}[n]$



Change Detection Algorithm: 2D-*n*AR(1)/correlation on Synchronous MoC

Time-Spatial Implementation[4]:



[4]M. Costa *et al.* "A Fast 2D-AR(1) Filtering for Bitemporal Change Detection on UWB SAR Images," *Proceedings of SPIE remote sensing*, September 2024.

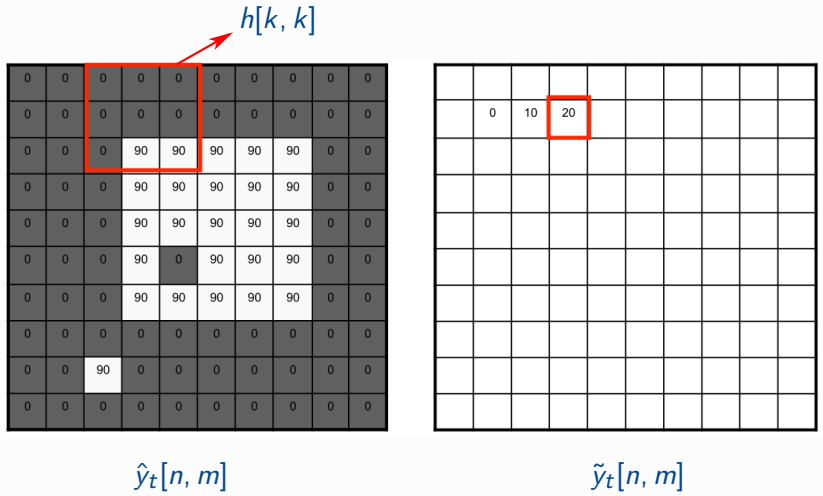
Spatial Filtering with Moving Average (FIR)

The smoothed function $\tilde{y}_t[n, m]$ under the neighborhood influence is determined by the kernel elements h through the original signal $\hat{y}_t[n, m]$, i.e.,

$$\tilde{y}_t[n, m] = (\hat{y}_t * h)[n, m]$$
$$\sum_{k,\ell} \hat{y}_t[k, \ell] h[n - k, m - \ell]$$

For a generic stencil kernel

$$h = \frac{1}{k} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & k \end{bmatrix}_{k \times k}$$



Change Detection Algorithm: 2D-*n*AR(1)/correlation on Synchronous MoC

```
-- Auxiliar: Stencil with Massiv
-------------------------------------------------------------------------------------------------------------
spatialFilter :: Int -> Int -> Signal (ForSyDe.Shallow.Matrix Double)  -> ForSyDe.Shallow.Matrix Double
spatialFilter dimx dimy img = matrix dimx dimy $ toList $ dropWindow $ mapStencil Edge (avgStencil 9) barImg
    where

        y_n' = fromSignal img  !! 0

        imG = fromMatrix y_n'

        barImg = fromLists' Par [imG] :: Array U Ix2 Double
```

Constructors

`Fill e` Fill in a constant element.

```
                outside |  Array  | outside
(Fill 0) : 0 0 0 0 | 1 2 3 4 | 0 0 0 0
```

`Wrap` Wrap around from the opposite border of the array.

```
                outside |  Array  | outside
Wrap :     1 2 3 4 | 1 2 3 4 | 1 2 3 4
```

`Edge` Replicate the element at the edge.

```
                outside |  Array  | outside
Edge :     1 1 1 1 | 1 2 3 4 | 4 4 4 4
```

`Reflect` Mirror like reflection.

```
                outside |  Array  | outside
Reflect :  4 3 2 1 | 1 2 3 4 | 4 3 2 1
```

`Continue` Also mirror like reflection, but without repeating the edge element.

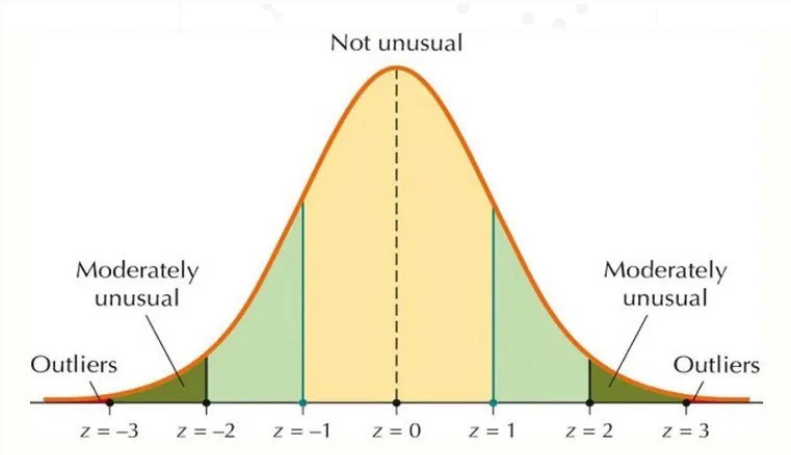
```
                outside |  Array  | outside
Continue : 1 4 3 2 | 1 2 3 4 | 3 2 1 4
```

Anomaly Detection

The empirical cumulative distribution function (ECDF), $\hat{F}(\cdot)$, is obtained from $[\tilde{y}_1 \ldots, \tilde{y}_n]$ i.i.d samples of output, making

$$\hat{F}(\tilde{y}) = \frac{1}{n}\sum_{i=1}^{n} I(\tilde{y}_i \leq \tilde{y}) \qquad (3)$$

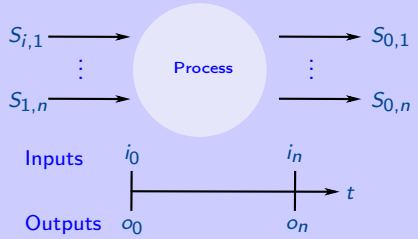
$$\hat{y}'_t[n, m] = \hat{F}^{-1}(p),$$

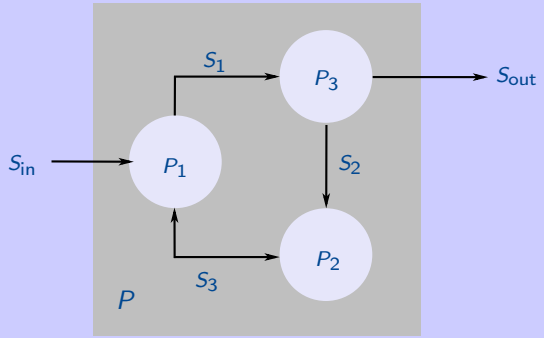


$\Rightarrow$ The quantile establishes the threshold on $\tilde{y}_t[n, m]$ when $x_t[n, m]$ has an anomaly of interest (potential target) against the lower clutter anomalies.

Synchronous MoC[5]: Synchronized parallel components, running in successive computation steps, where all components perform some quantum of computation.

1. Output is synchronous with input
2. Internal actions are instantaneous





[5]Albert Benveniste and Gérard Berry. "The synchronous approach to reactive and real-time systems," *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

Change Detection with AR Model

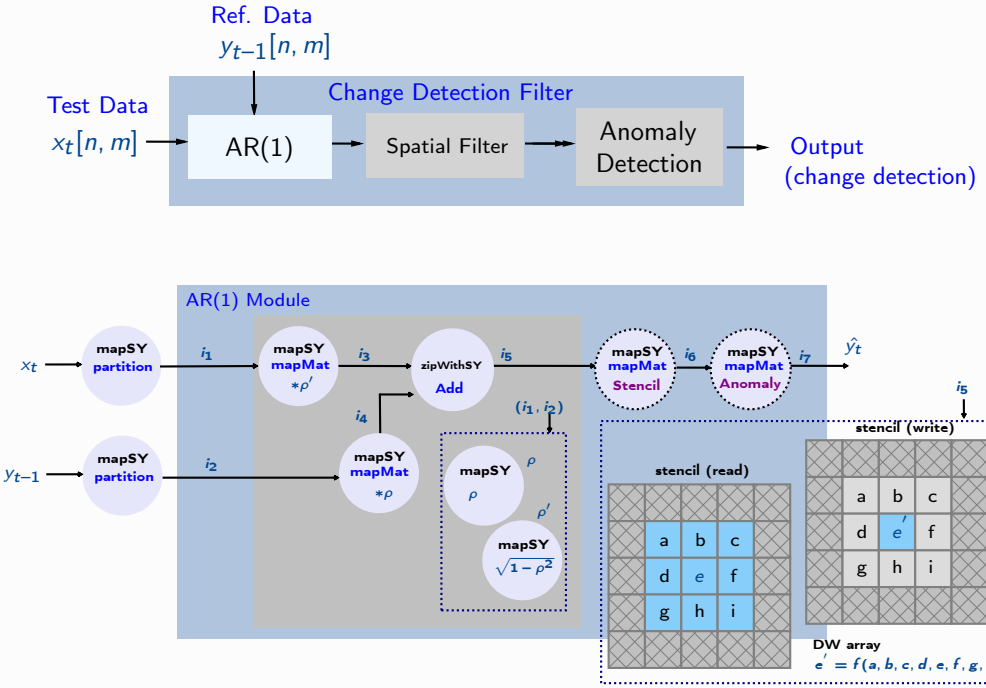
- ForSyDe Framework[5]/Haskell

Synchronuous MoC constructors:

- Combinational → `zipWithSY`, `mapSY`
- Delay → `delaySY`
- Sequential → `mooreSY`, `mealySY`

Parallel stencil computation

- Data.Massiv.ArrayFramework[6]

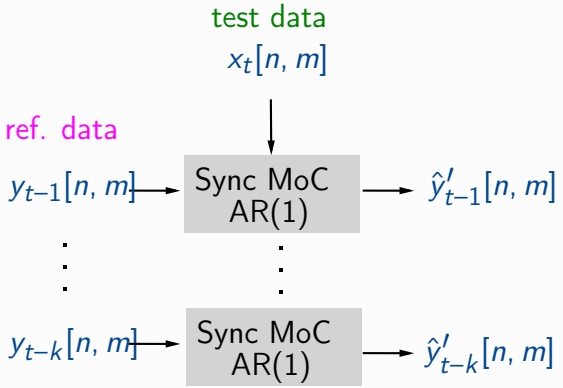


[5] https://forsyde.github.io/

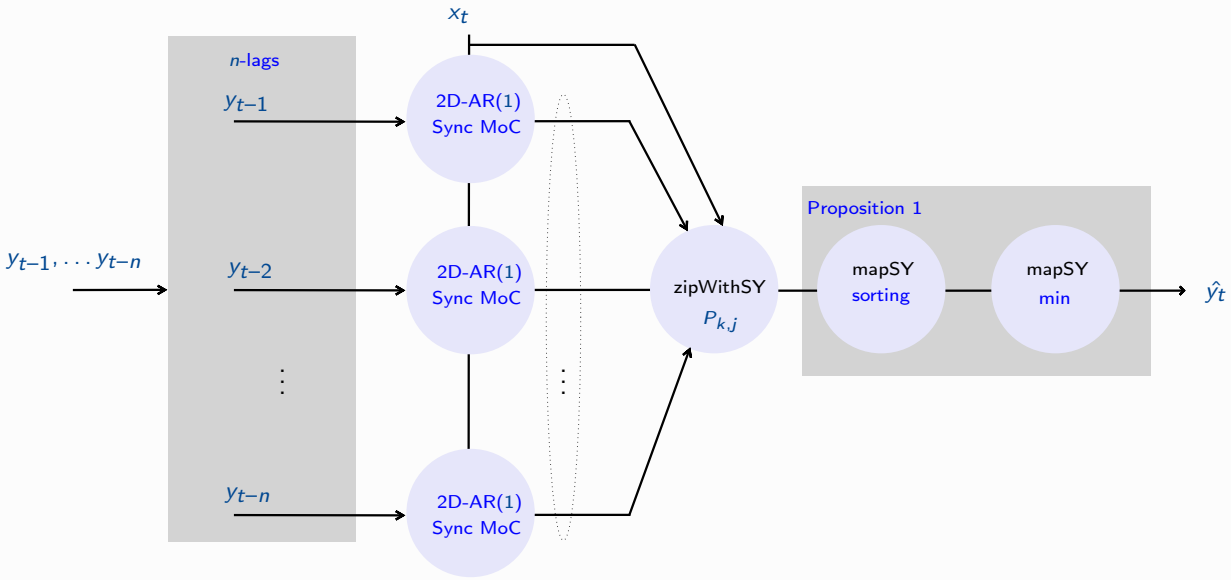
[6] https://hackage.haskell.org/package/massiv

Extension: n-lags Model

- Hardware efforts $\times$ detection performance are balanced over high-parallelized Sync MoC in terms of $[N, k]$ selection.
- Extension for a Markov chain in multitemporal (n-lags) improves the detection performance using a parallel structure over the elementary MoC structure.



Synchronous MoC on ForSyDe



Change Detection Algorithm: 2D-*n*AR(1)/correlation on Synchronous MoC

```
----- AR(n) + MC ---------------------------------------------------------------------------------------------

-- Charalampos suggestion

let u1 = vector [intest,inref1]; u2 = vector [intest,inref3]; u3 = vector [intest,inref3]; u4 = vector [intest,inref5]

let m1 = zipWithxSY (procAR1 dimx dimy) u1
    m2 = zipWithxSY (procAR1 dimx dimy) u2
    m3 = zipWithxSY (procAR1 dimx dimy) u3
    m4 = zipWithxSY (procAR1 dimx dimy) u4
```

```
import Control.Parallel.Strategies (parMap, rpar, parListChunk, parBuffer, runEval)

parMap rpar  (\x -> (f x)) [0 ..100]
```

Parallel scheduling implementation

```
-- Probabilities Matrices (skeleton)
let out = vector [intest,m1,intest, m2, intest,m3,intest, m4]
let res = zipWithxSY (procMatrix4 dimx dimy) out
```

Vector of Processes skeletons

```
----- Output File --------------------------------------------------------------------------------------------
writeFile "/home/marcello-costa/workspace/Demos/Out/CD0.txt" (show res)
writeFile "/home/marcello-costa/workspace/Demos/Out/Test1.txt" (show intest)
writeFile "/home/marcello-costa/workspace/Demos/Out/Iref2.txt" (show inref1)
```

Change Detection Algorithm: 2D-*n*AR(1)/correlation on Synchronous MoC

```haskell
procAR1 :: Int -> Int -> ForSyDe.Shallow.Vector (Signal(ForSyDe.Shallow.Matrix (ForSyDe.Shallow.Matrix Double)))-> Signal (ForSyDe.Shallow.Matrix (ForSyDe.Shallow.Matrix Double) )
procAR1 dimx dimy dat = out
      where

      t = dat `atV` 0; st = fromSignal t !! 0

      ref1 = dat `atV` 1; sr1 = fromSignal ref1 !! 0; ssr1 = vector [sr1]
      sv = signal [ssr1]

      m = (zipxSY . mapV (mapSY (zipWithMat(\ x y -> arSystem dimx dimy (signal [y]) (signal [x]) ) st )) . unzipxSY) sv
      m1 = fromSignal m !! 0; cm1 = m1 `atV` 0

      out = signal [cm1]
```

AR(1)

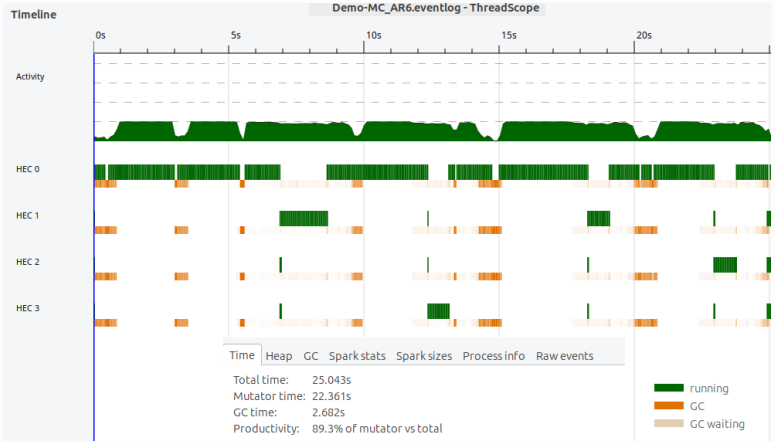
→ External Parallelism (Haskell)

```haskell
procMatrix4 :: Int -> Int -> ForSyDe.Shallow.Vector (Signal(ForSyDe.Shallow.Matrix (ForSyDe.Shallow.Matrix Double)))-> Signal (ForSyDe.Shallow.Matrix Double)
procMatrix4 dimx dimy dat = signal [out1]
      where

      t = dat `atV` 0; st = fromSignal t !! 0
      ref1 = dat `atV` 1; sr1 = fromSignal ref1 !! 0; ssr1 = vector [sr1]
      ref2 = dat `atV` 3; sr2 = fromSignal ref2 !! 0; ssr2 = vector [sr2]
      ref3 = dat `atV` 5; sr3 = fromSignal ref3 !! 0; ssr3 = vector [sr3]
      ref4 = dat `atV` 7; sr4 = fromSignal ref4 !! 0; ssr4 = vector [sr4]

      sv = signal [ssr1, ssr2, ssr3, ssr4]
      c = (zipxSY . mapV (mapSY (zipWithMat(\ x y -> mcSystem dimx dimy (signal [y]) (signal [x]) ) st )) . unzipxSY) sv

      p1 = fromSignal c !! 0; p1mat = fromMatrix p1 !! 0; p1List = p1mat `atV` 0
      p2 = fromSignal c !! 1; p2mat = fromMatrix p2 !! 0; p2List = p2mat `atV` 0
      p3 = fromSignal c !! 2; p3mat = fromMatrix p3 !! 0; p3List = p3mat `atV` 0
      p4 = fromSignal c !! 3; p4mat = fromMatrix p4 !! 0; p4List = p4mat `atV` 0

      pLists = [p1List] ++ [p2List] ++ [p3List] ++ [p4List]

      cm1 = sr1 `atV` 0
      cm2 = sr2 `atV` 0
      cm3 = sr3 `atV` 0
      cm4 = sr4 `atV` 0

      cms = [cm1, cm2, cm3, cm4]

      ---- Sorting Lists ---
      revpLists = reverseOrder pLists
      pOrder = qsort pLists
      sorList = findIndices (\l -> l <= pOrder !! 0) pOrder

      out = cms !! head sorList
      out1 = out `atV` 0
```

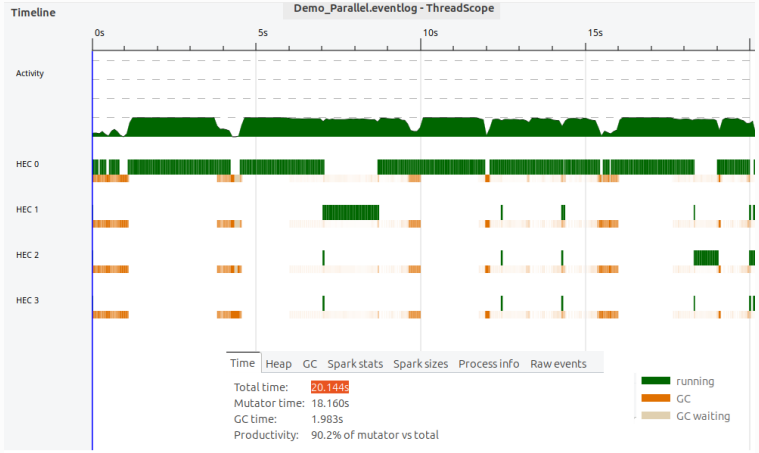
MC
Vector of Processes

→ Data Parallelism Skeletons (ForSyDe)

Performance using 4 threads (threadscope), and 4-lags



(a) Data parallelism with Skeletons

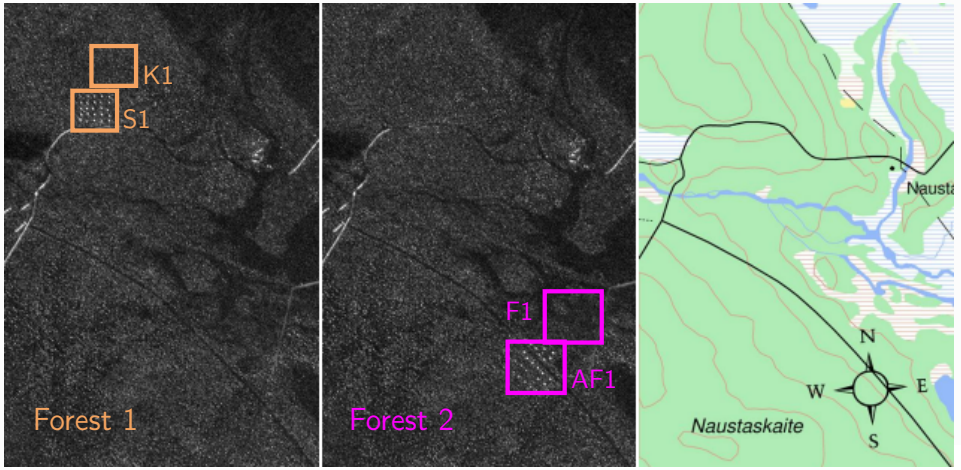


(b) Parallel Evaluation Strategies

CASE: Change Detection on CARABAS-II

CARABAS-II dataset[7]:

- 24 images: magnitude VHF (20 – 90 MHz) SAR HH-polarized with 1 m resolution (2×3 km^2)
- 4 Targets position: S1, K1, F1, and AF1
- 6 passes: 3 Flight directions under ground RFI sources ON/OFF
- Application: Through-foliage detection



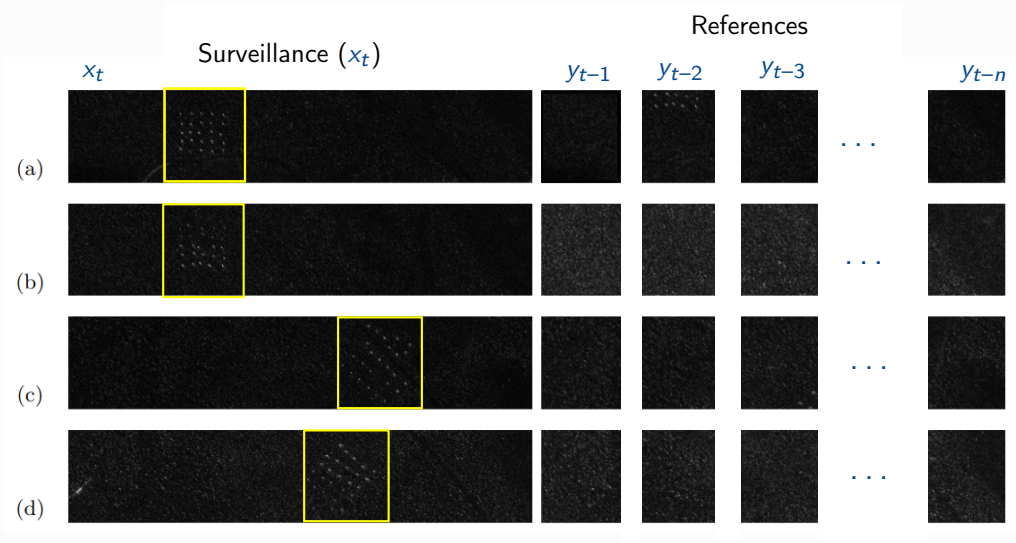
[7] Ulander, L.M, et al. "Change detection for low-frequency sar ground surveillance," *IEE Proceedings-Radar, Sonar and Navigation* , 152(6), pp. 413–420, 2005.

CASE: Change Detection on CARABAS-II

	Exp. No.	Surveillance image		Reference image		Known Targets	Area [km^2]
		Mission	Pass	Mission	Pass		
(a)	1	2	1	3	1	25	6
(b)	2	3	1	4	1	25	6
(c)	3	4	1	5	1	25	6
(d)	4	5	1	2	1	25	6
	5	2	2	4	2	25	6
	6	3	2	5	2	25	6
	7	4	2	2	2	25	6
	8	5	2	3	2	25	6
	9	2	3	5	3	25	6
	10	3	3	2	3	25	6
	11	4	3	3	3	25	6
	12	5	3	4	3	25	6
	13	2	4	3	4	25	6
	14	3	4	4	4	25	6
	15	4	4	5	4	25	6
	16	5	4	2	4	25	6
	17	2	5	4	5	25	6
	18	3	5	5	5	25	6
	19	4	5	2	5	25	6
	20	5	5	3	5	25	6
	21	2	6	5	6	25	6
	22	3	6	2	6	25	6
	23	4	6	3	6	25	6
	24	5	6	4	6	25	6
	Total					600	144

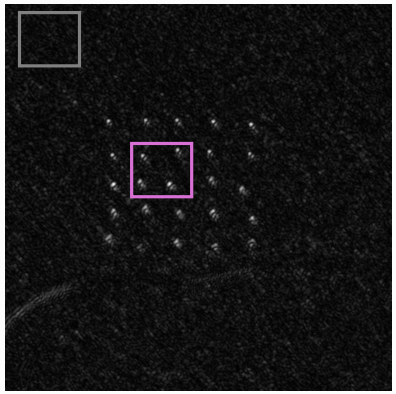
Except the same mission

Campaign sample data: 0.5×0.5 km (4%)



Analytical Block

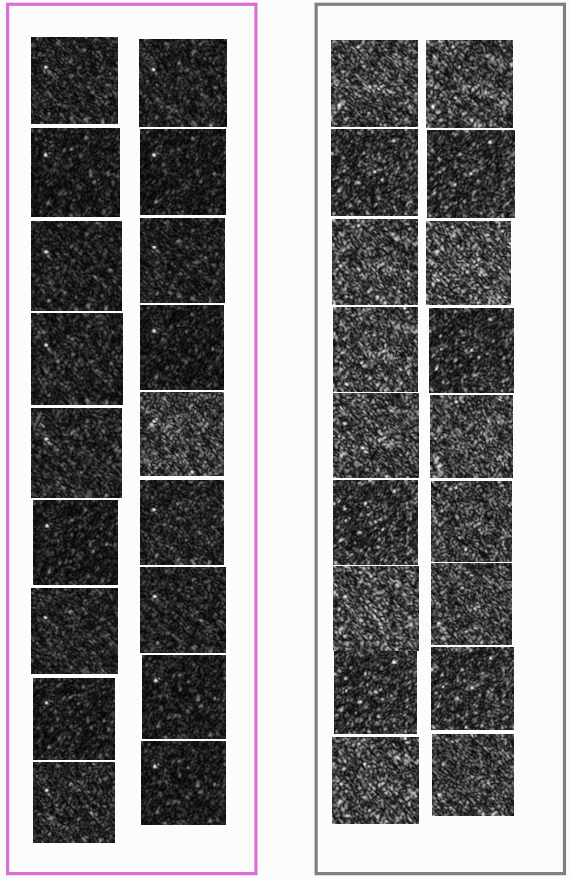
surveillance x_t



sample 100×100 : target-clutter

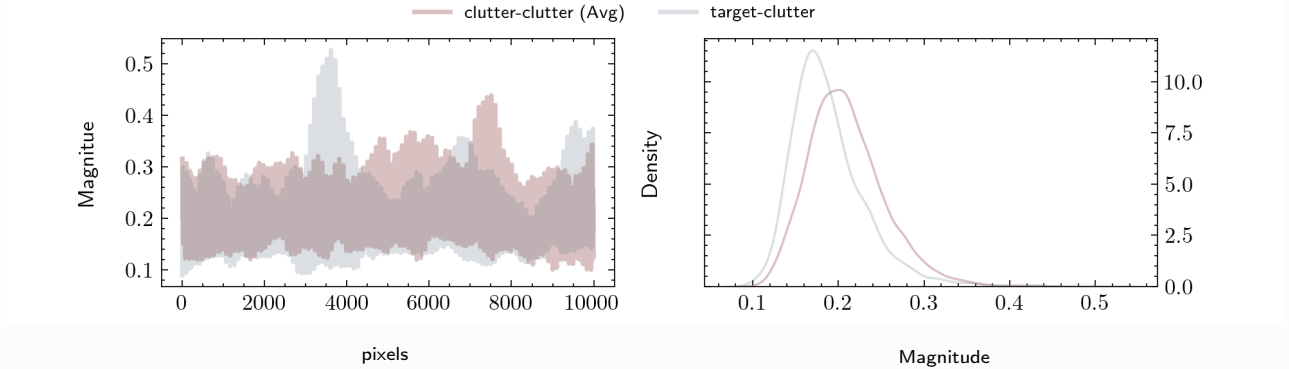
sample 100×100 : clutter-clutter

references y_{t-n}

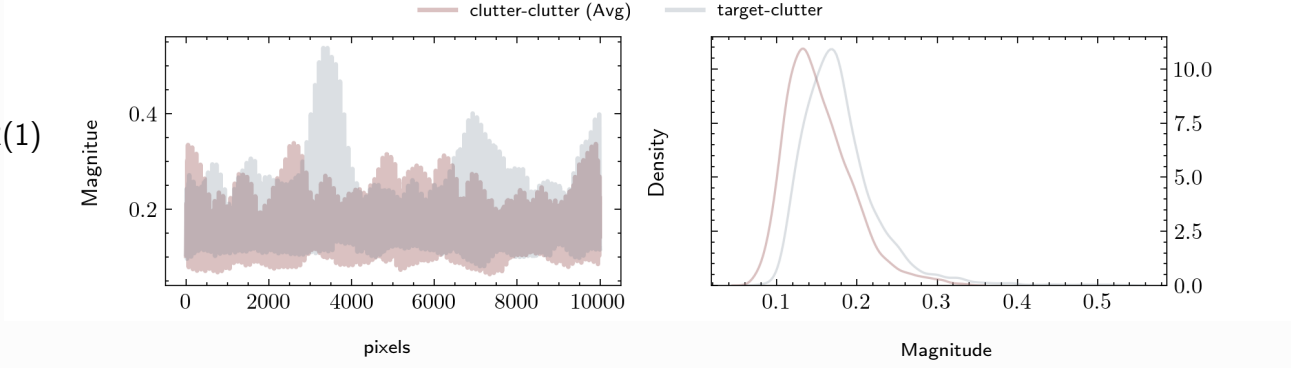


Signal separation: 0.100×0.100 km^2

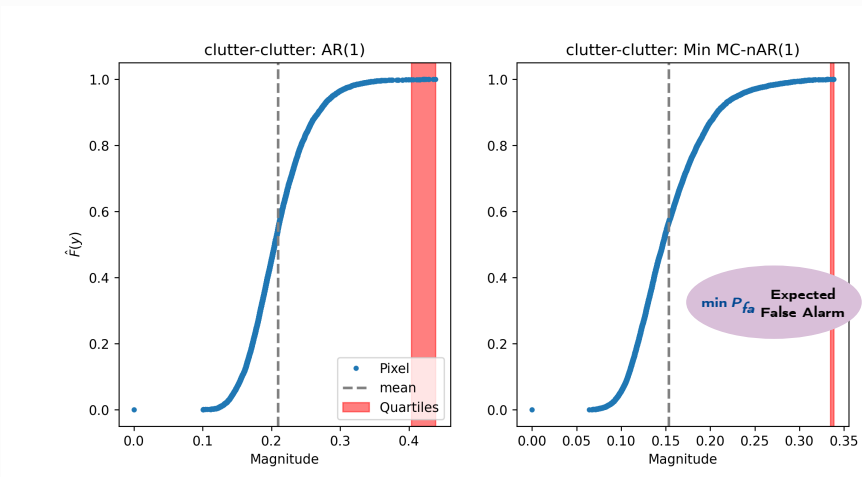
AR(1)



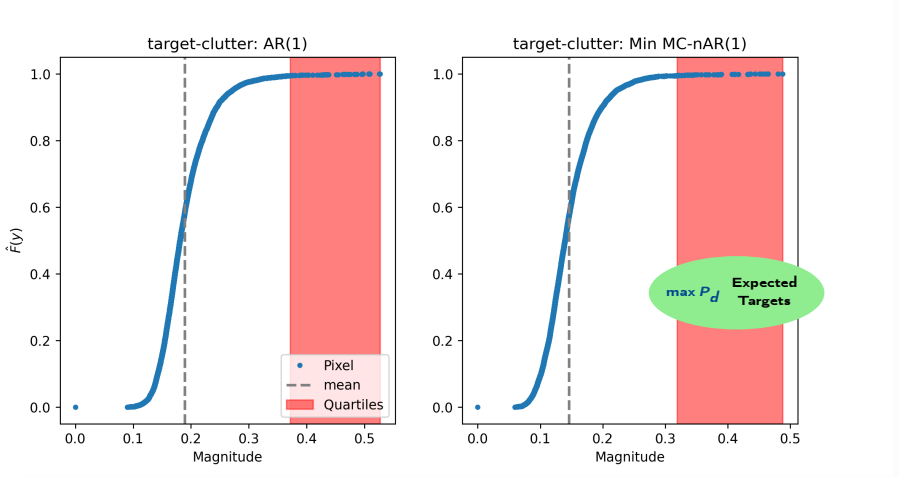
Min nAR(1)



Anomaly Detection: 0.100×0.100 km^2



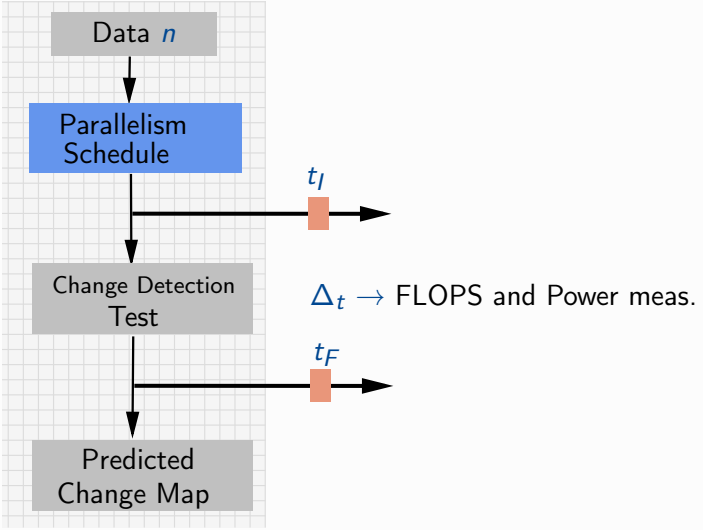
(a) Clutter area



(b) Target area

Test Campaign

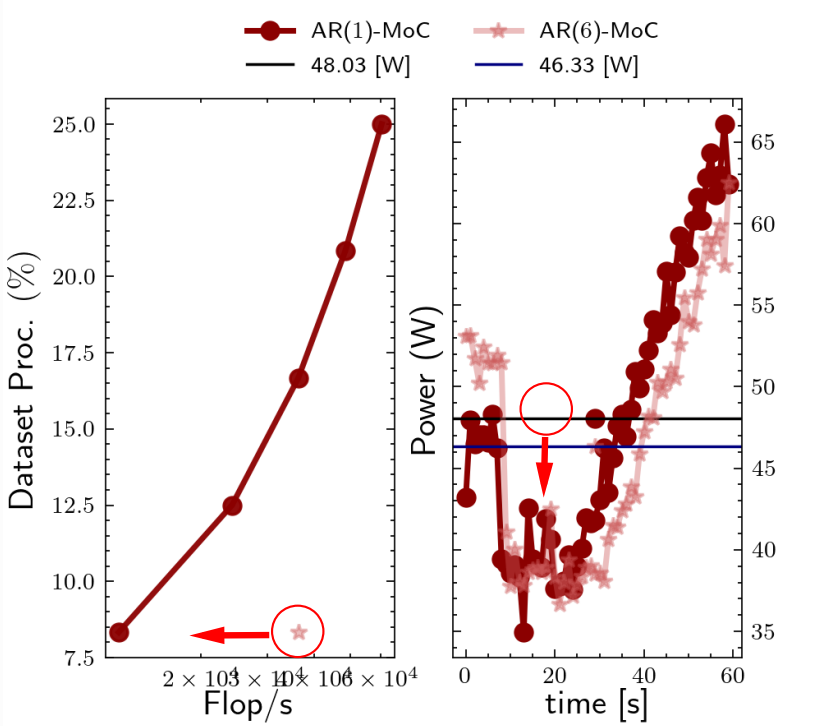
- Scenarios: Sample dataset 4% (and 4 test groups)
- Tests: [AR(1), 3-AR(1), 6-AR(1), 9-AR(1)]
- Analytical block N: [500]
- Parallelism scheduling: $[2, 3, 6, 9]$ cores/threads
- Convolution block k: [9].
- perfs[9]: `htop`, `powerstat`, `GHC`



[9] The CPU processor Intel Core i9-12900KF, 16-Core, 24-threads, clock of 3.2 GHz and 32 GB DDR5 of memory.

0.5×0.5 km^2 (4%), $N = 500$, $K = 9$, $n = 6$

- Sync MoC implementation
- Deterministic parallelism using skeletons on AR(1) + MC
- Efficient consumption to improve the runtime for 6-lags performance
- **Parallel Evaluation Strategies**



Architectures Analysis (Detection vs HW Efforts)

1. CPU Multicores

2. GPU cores

3. FPGA


KTH
VETENSKAP
OCH KONST