Signal-to-Memory Mapping Analysis for Multimedia Signal Processing

Ilie I. Luican   *Hongwei Zhu*   *Florin Balasa*

University of Illinois at Chicago

Outline

- Memory management for signal processing
- Signal-to-memory mapping
- An efficient mapping algorithm that covers the [De Greef 1997] and [Troncon 2002] mapping models
- Comparative analysis of mapping models
- Experimental results
- Conclusions

Memory management
for signal processing applications

Real-time (multi-dimensional) signal processing systems

(video and image processing, real-time 3D rendering, audio and speech coding, medical imaging, etc.)

data transfer and data storage

↓

system performance
power consumption
chip area

→

The designer must focus on the exploration of the memory subsystem

Memory management for signal processing applications

```
T[0] = 0;
for ( j=16; j<=512; j++ )  {
   S[0][j-16][0] = 0;
   for ( k=0; k<=8; k++ )
      for (i=j-16; i<=j+16; i++ )
         S[0][j-16][33*k+i-j+17] = S[0][j-16][33*k+i-j+16] + A[4][j] – A[k][i];
   T[j-15] = S[0][j-16][297] + T[j-16];
}
out = T[497];
```

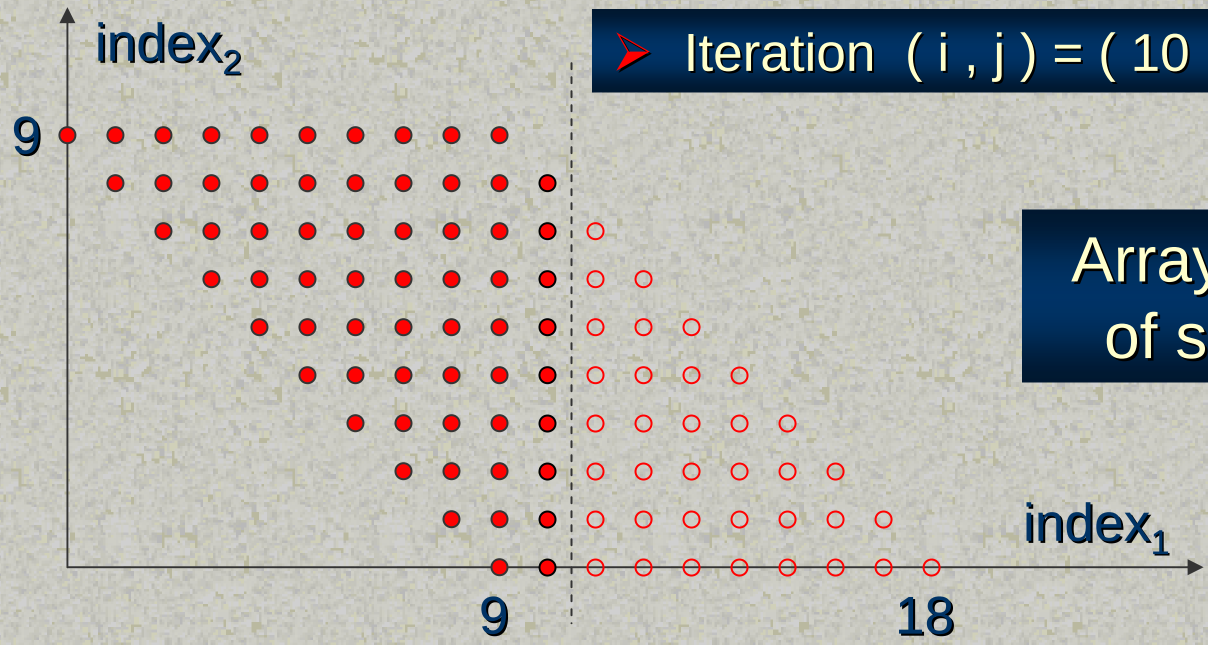
- Affine algorithmic specifications

- Loop-organized algorithmic specification

- Main data structures: multi-dimensional arrays

Signal-to-Memory Mapping

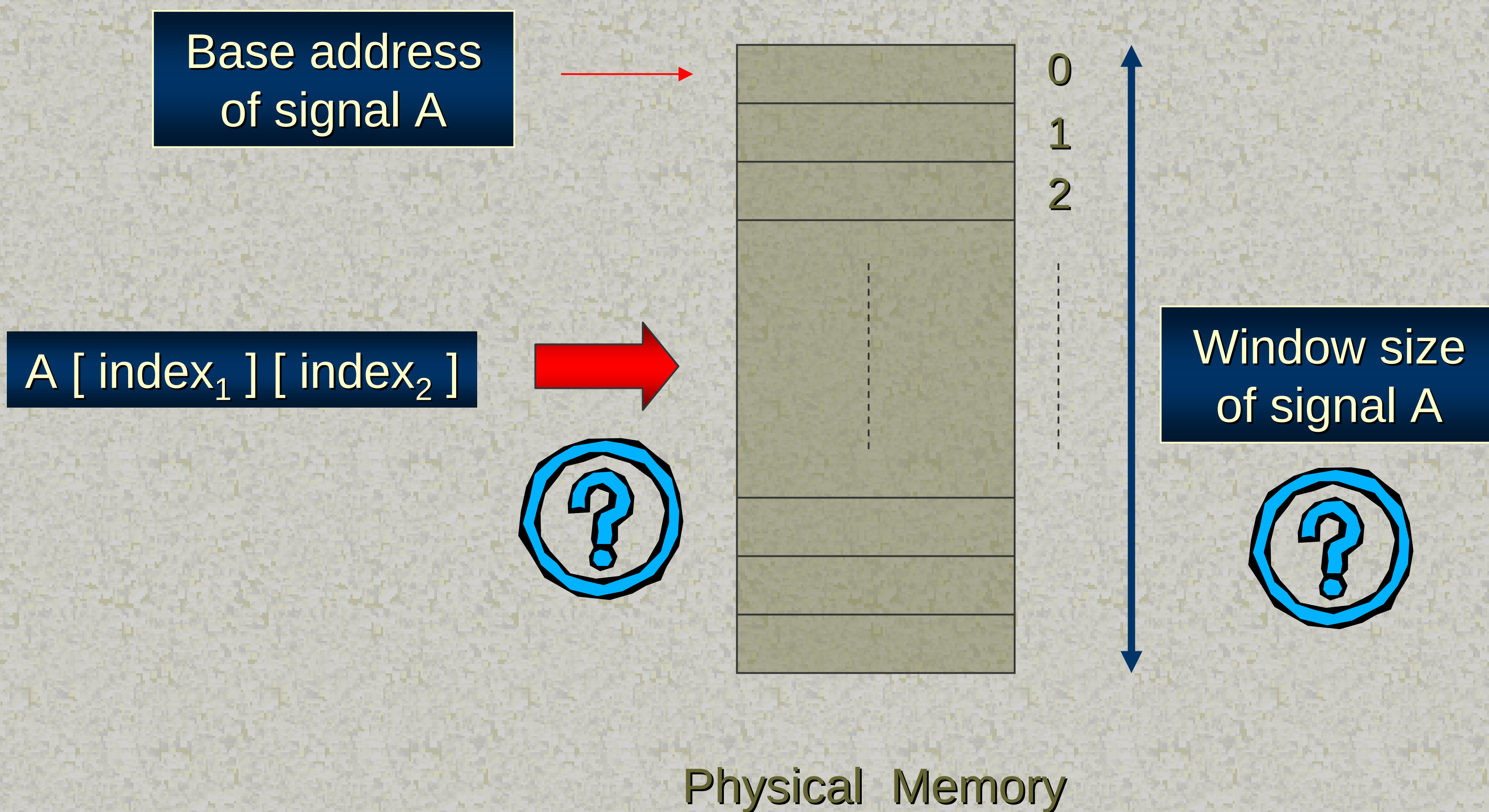
➤ Iteration (i , j) = (10 , 9)



Array space of signal A

```
for ( i = 0; i < 29; i++ )
  for ( j = 0; j < 10; j++ )  {
    if ( i+j >=  9  &&  i+j <= 18 )   A[i][j] = … ;
    if ( i+j >= 19  &&  i+j <= 28 )        … = A[i -10][j] ;
  }
```

Signal-to-Memory Mapping



Signal-to-Memory Mapping

[De Greef 1997] mapping model

m-dim. array → $2^m \cdot m!$ canonical array linearizations

window size = Min Max { dist. simultaneously alive elements } + 1

All linearizations

Array element → (Index in the minimizing linearization)

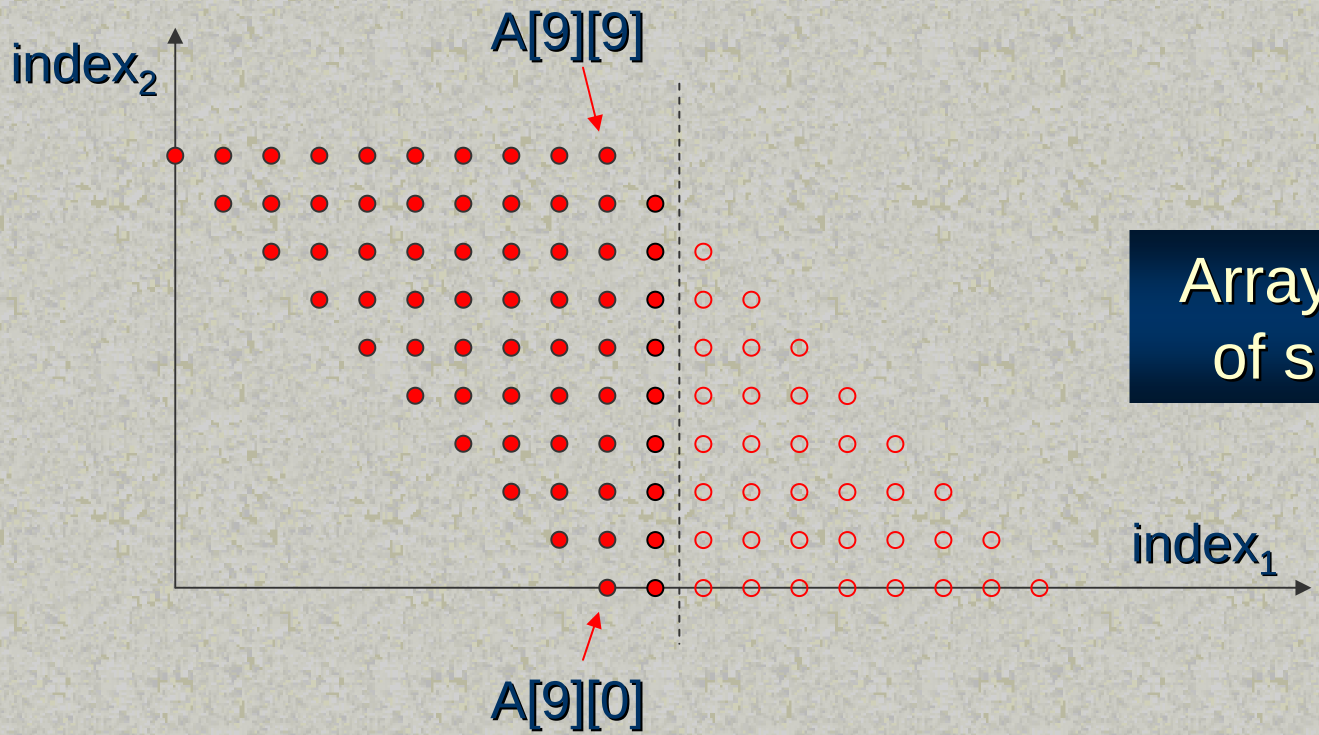
mapped to

modulo

(window size)

Signal-to-Memory Mapping

[De Greef 1997] mapping model

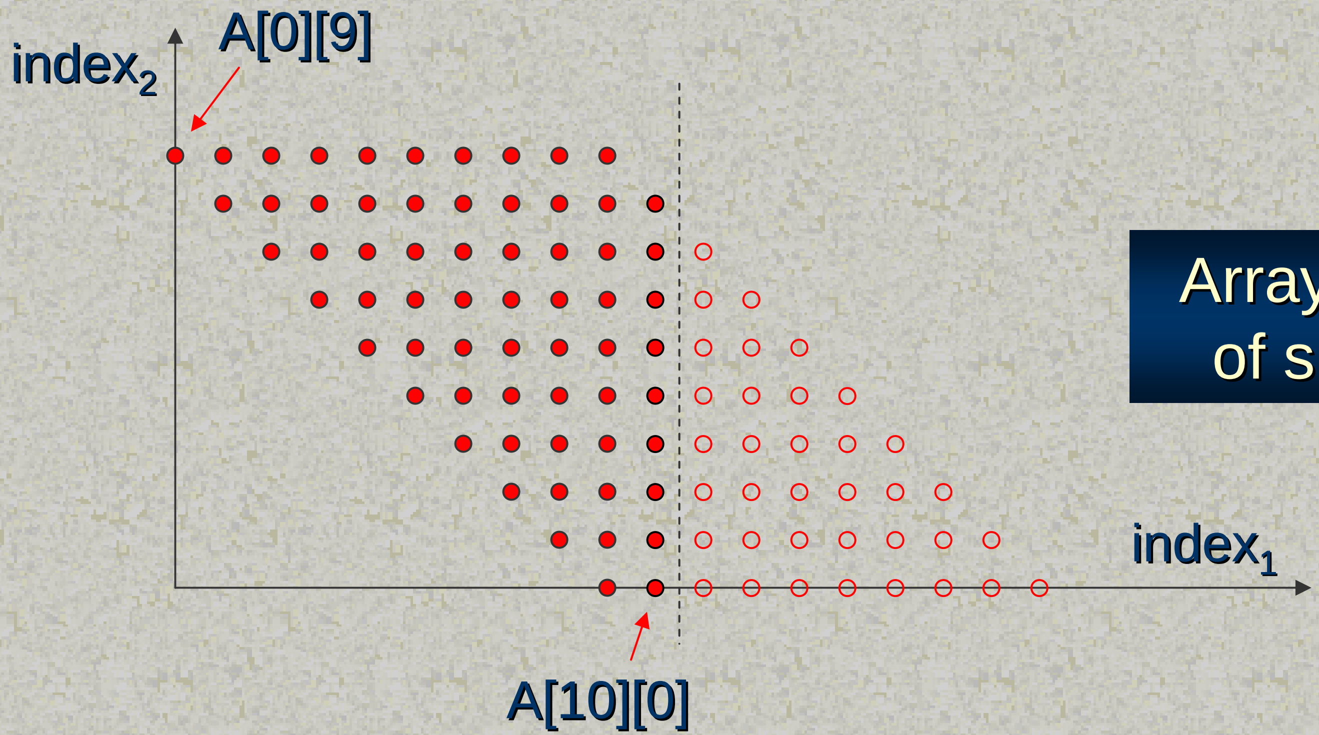


Array space of signal A

Column concatenation (direct order) ➡ Max {dist} = 9 x 19 = 171

Signal-to-Memory Mapping

[De Greef 1997] mapping model

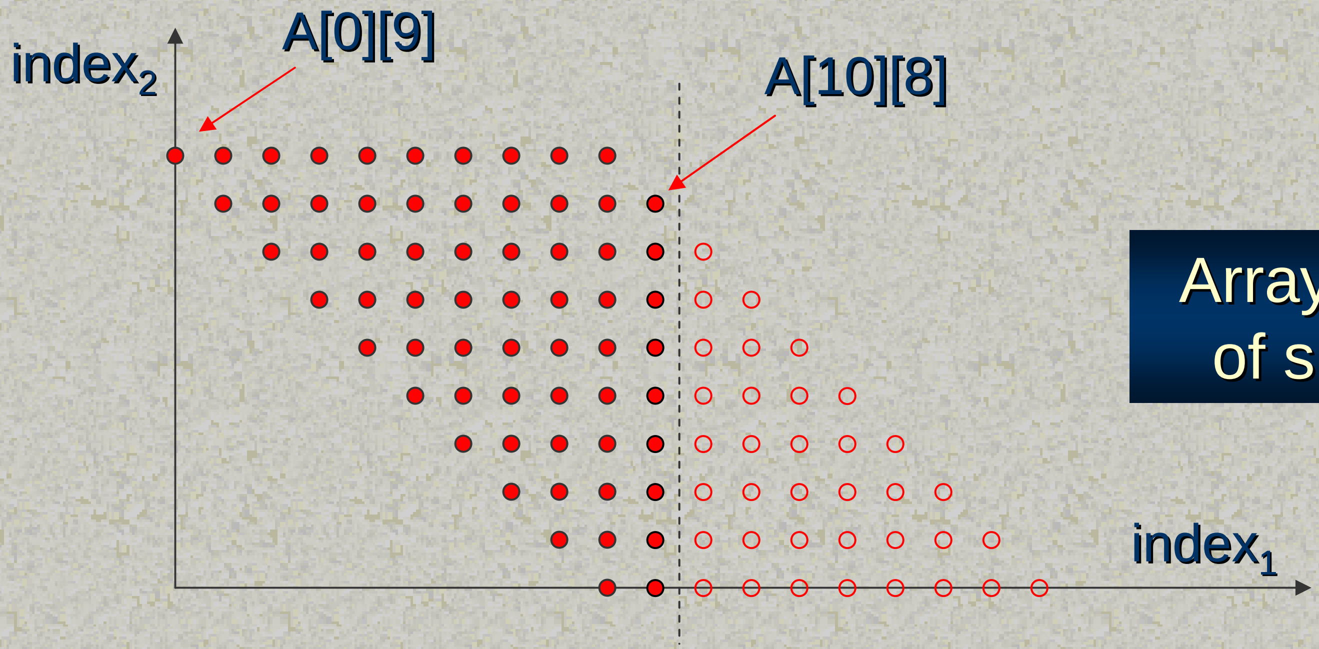


Array space of signal A

Column concatenation (reverse order) ➡ Max {dist} = 181

Signal-to-Memory Mapping

[De Greef 1997] mapping model

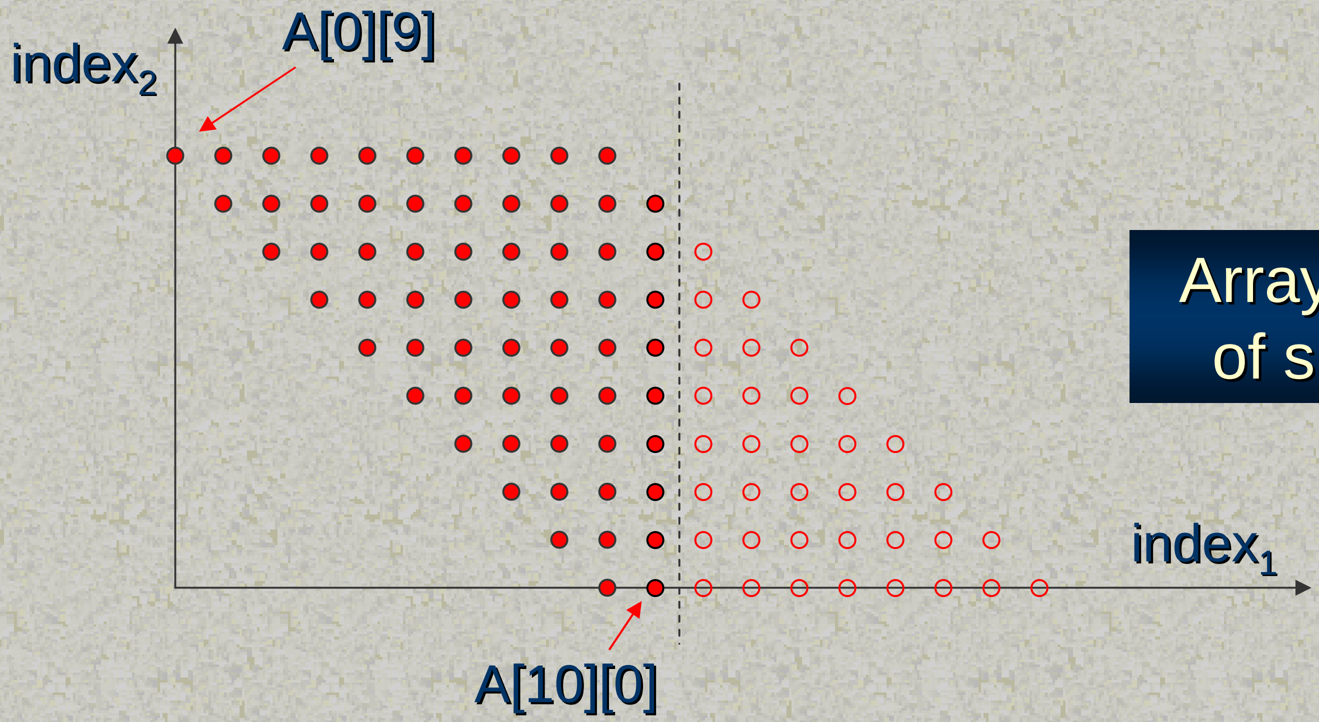


Array space of signal A

Row concatenation (direct order) ➡ Max {dist} = 10 x 10 – 1 = 99

Signal-to-Memory Mapping

[De Greef 1997] mapping model



Array space of signal A

Row concatenation (reverse order) ➡ Max {dist} = 109

Signal-to-Memory Mapping

[Troncon 2002] mapping model

m-dim. array → m-dim. window (w_1 , … , w_m)

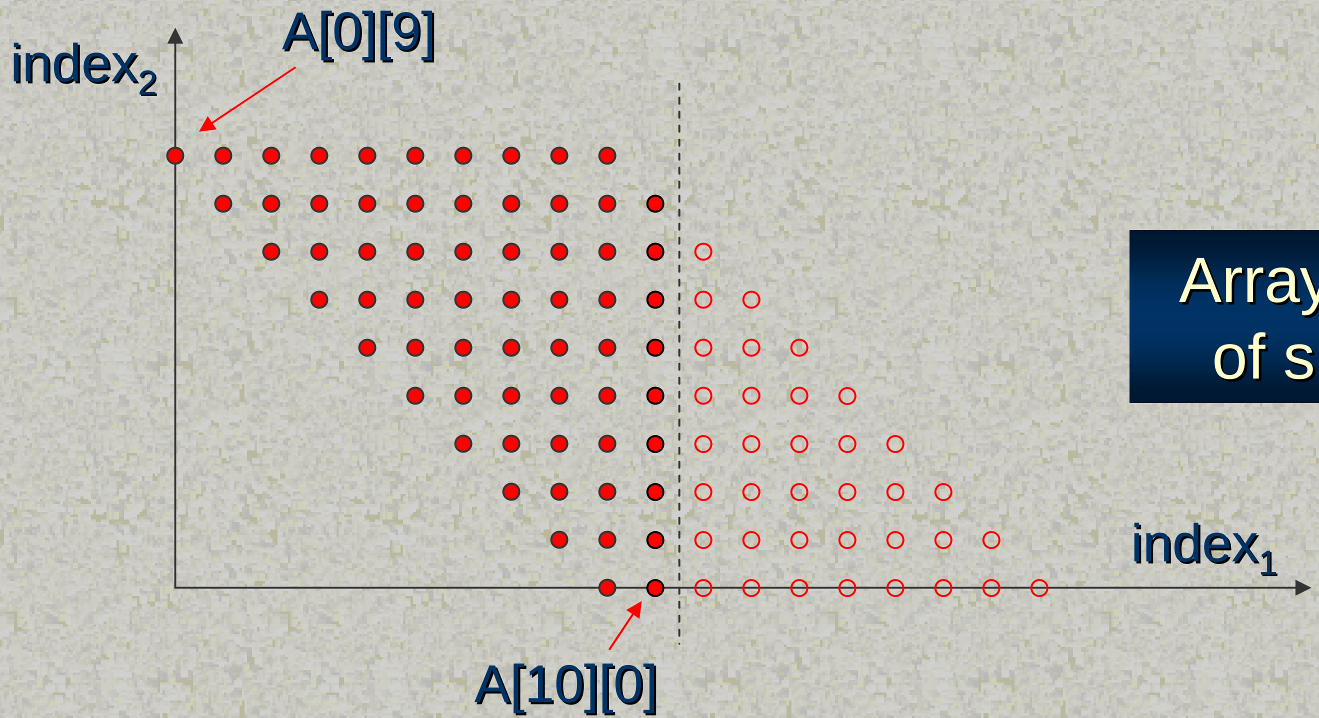
w_i = Max { dist. alive elements having same index i } + 1

A [index_1] … [index_m] → mapped to

A [index_1 *mod* w_1] … [index_m *mod* w_m]

Signal-to-Memory Mapping

[Troncon 2002] mapping model



Array space of signal A

2-D window ($w_1 = 11$, $w_2 = 10$)

Signal-to-Memory Mapping

```
for ( i = 0; i < 29; i++ )
  for ( j = 0; j < 10; j++ )  {
    if ( i+j >=   9  &&  i+j <= 18 )   A[i][j] = ... ;
    if ( i+j >= 19  &&  i+j <= 28 )        ... = A[i -10][j] ;
  }
```

Window size of signal A

[De Greef 1997] model

- 100 storage locations (row concatenation)

[Troncon 2002] model

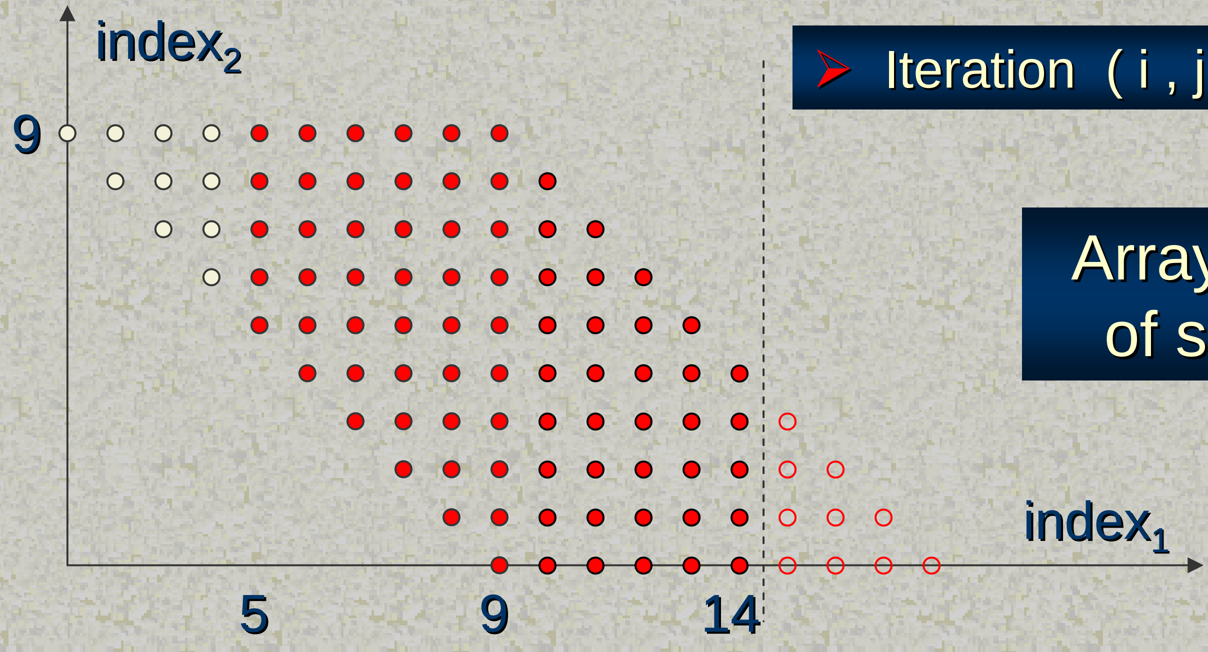
- 110 storage locations window = (11,10)

Signal-to-Memory Mapping

```
for ( i = 0; i < 29; i++ )
  for ( j = 0; j < 10; j++ )  {
    if ( i+j >=   9  &&  i+j <= 18 )   A[i][j] = … ;
    if ( i+j >= 19  &&  i+j <= 28 )         … = A[i -10][j] ;
  }
```

➢ Minimum storage (A) = 80 storage locations

➢ Iteration (i , j) = (14 , 5)



Array space of signal A

Comparative Analysis of Mapping Models

- Signal-to-memory mapping models trade-off data storage for a less costly address generation hardware

- So far, the mapping models were evaluated only **relatively** (by comparing the storage requirements when different models are used)

This framework allows to better evaluate mapping models

- **by computing each array's minimum window** (the optimal memory sharing between each array's elements)

- **by computing the minimum data storage of the application** (the optimal memory sharing between all the scalars in the code)

Computation of the 1-D Window of a Lattice of Live Signals

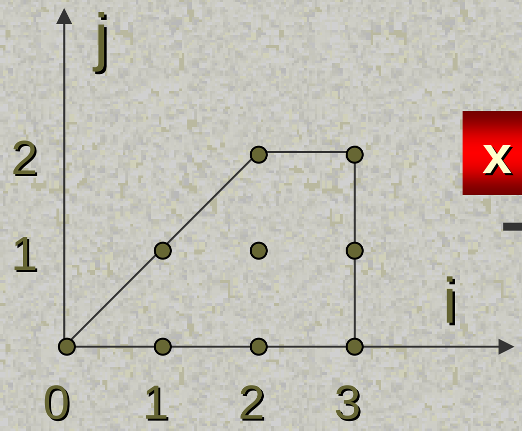
```
for ( i=0; i<=3; i++ )
    for ( j=0; j<= 2; j++ )
        if ( 3i >= 2j )  …   A [2i+3j] [5i+j]   …
```

Any array reference can be modeled as a lattice

{ x = T·i + u | A·i >= b }

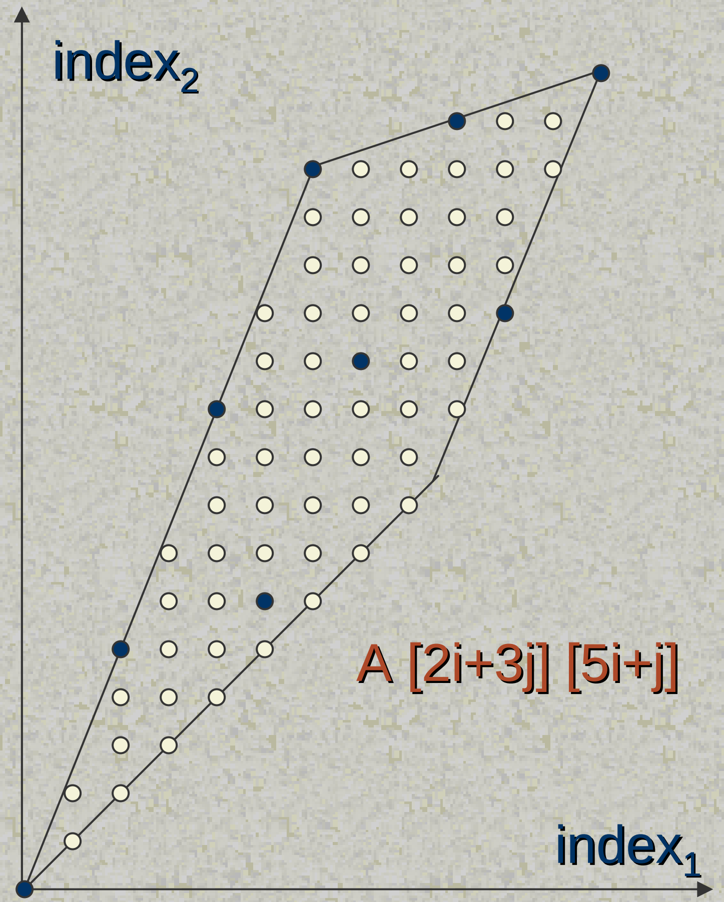
$$\left\{ \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \;\middle|\; \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 3 & -2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} >= \begin{pmatrix} -3 \\ 0 \\ -2 \\ 0 \end{pmatrix} \right\}$$

Computation of the 1-D Window of a Lattice of Live Signals



Iterator space

$x = T \cdot i + u$



Index space

$i \le 3, \quad 0 \le j \le 2, \quad 2j \le 3i$

Computation of the 1-D Window of a Lattice of Live Signals

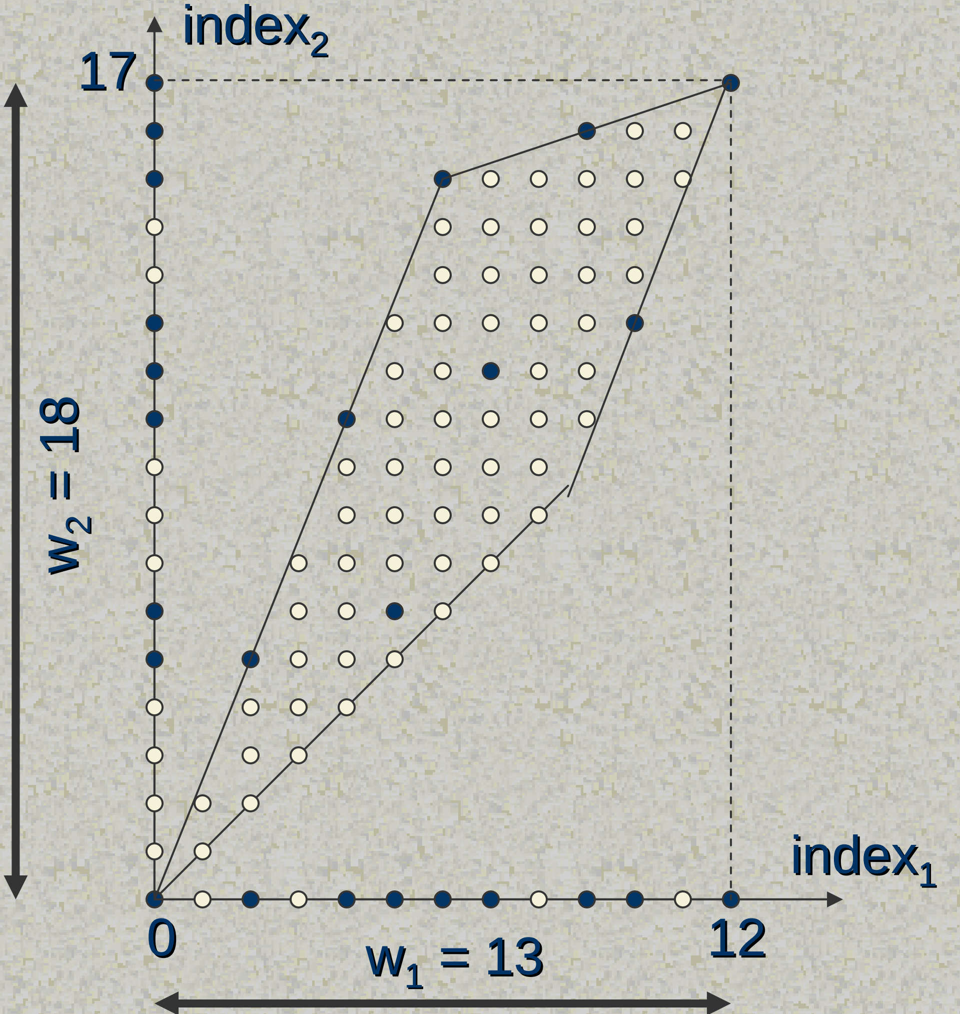
[Troncon 2002]
mapping model

A [2i+3j] [5i+j]

Two 1-D windows

($w_1 = 13$, $w_2 = 18$)

by integer projection
of the lattice on the axes



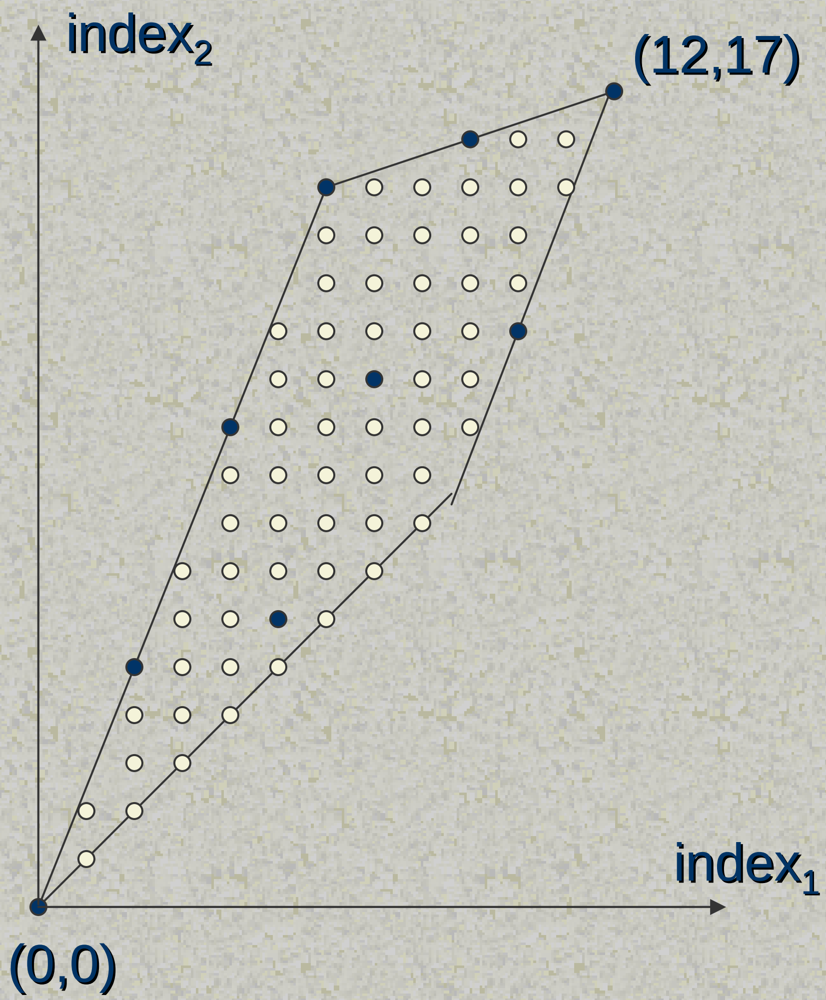
Computation of the 1-D Window of a Lattice of Live Signals

[De Greef 1997]
mapping model

A [2i+3j] [5i+j]

1-D window
(row concatenation)

$$\text{Dist}(\ \text{Min } A[\ index_1\ ][\ index_2\ ],\ \text{Max } A[\ index_1\ ][\ index_2\ ]\ ) + 1$$
$$= \text{Dist}(\ A(0,0)\ ,\ A(12,17)\ ) + 1$$
$$= 13 \times 18$$



Computation of the 1-D Window of a Lattice of Live Signals

[De Greef 1997] mapping model

A [2i+3j] [5i+j]

1-D window (column concatenation) → Dist(Min A [$index_2$] [$index_1$] , Max A [$index_2$] [$index_1$]) + 1 = Dist(A(0,0) , A(12,17)) + 1

For any linearization, the problem reduces to the computation of (lexicographically) Min / Max array elements in the lattice

Signal-to-Memory Mapping Algorithm

Decompose the array references into disjoint lattices

$L_1 \cap L_2 \rightarrow L$

$$\begin{cases} L_1 = \{\, x = T_1 \cdot i_1 + u_1 \mid A_1 \cdot i_1 >= b_1 \,\} \\ L_2 = \{\, x = T_2 \cdot i_2 + u_2 \mid A_2 \cdot i_2 >= b_2 \,\} \end{cases}$$

$T_1 \cdot i_1 + u_1 = T_2 \cdot i_2 + u_2$

Diophantine system of eqs.

$\{\, A_1 \cdot i_1 >= b_1 ,\ A_2 \cdot i_2 >= b_2 \,\}$

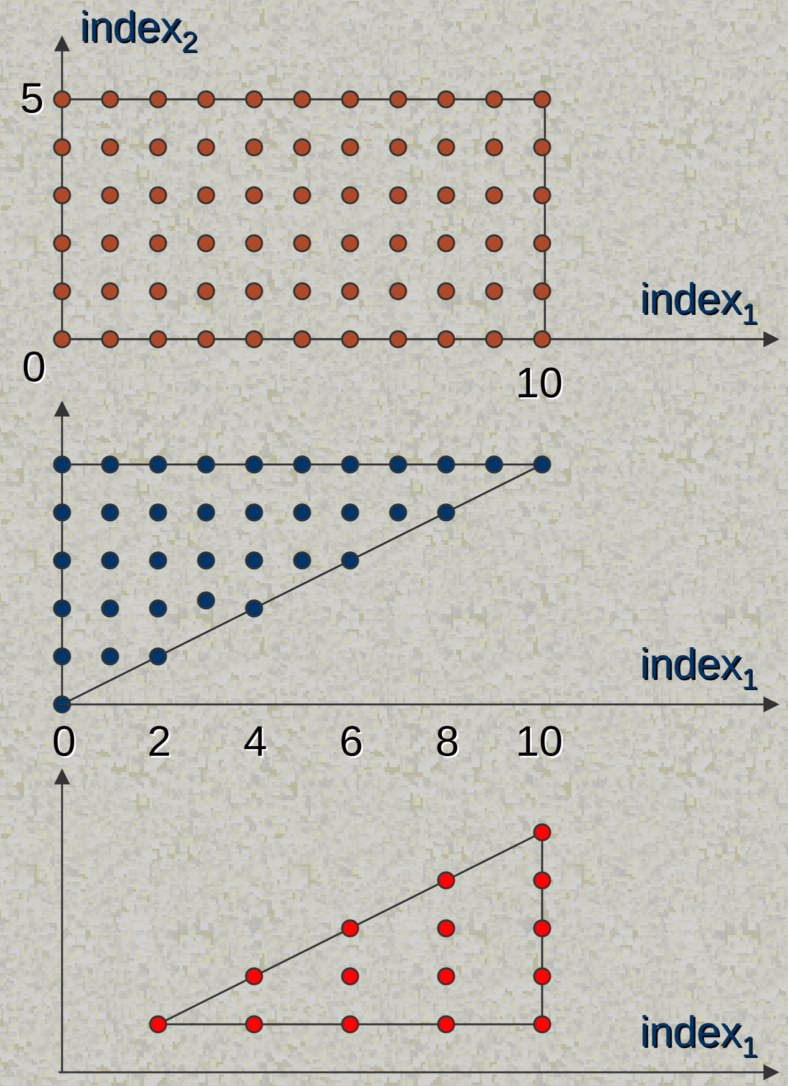
New polytope

Signal-to-Memory Mapping Algorithm

```
for ( k=0; k<=10; k++ )
    for ( l=0; l<= 5; l++ )
        A[k][l] = …

for ( j=0; j<=5; j++ )
    for ( i=0; i<= 2*j; i++ )
        … = A[i][j] ;

for ( i=1; i<=5; i++ )
    for ( j=0; j<= i-1; j++ )
        … = A[2*i][j+1] ;
```

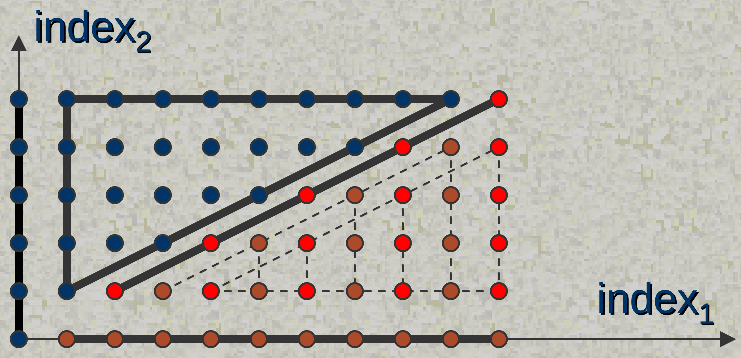


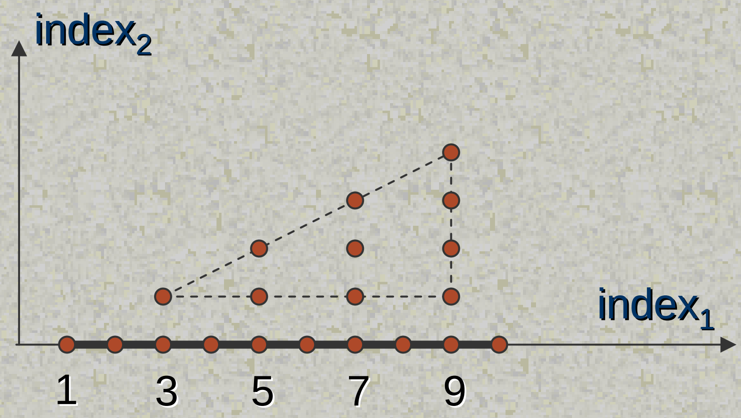
Signal-to-Memory Mapping Algorithm

```
for ( k=0; k<=10; k++ )
    for ( l=0; l<= 5; l++ )
        A[k][l] = …

for ( j=0; j<=5; j++ )
    for ( i=0; i<= 2*j; i++ )
        … = A[i][j] ;

for ( i=1; i<=5; i++ )
    for ( j=0; j<= i-1; j++ )
        … = A[2*i][j+1] ;
```





The live A-elements after the 3rd loop nest

Signal-to-Memory Mapping Algorithm

Step 1 For every indexed signal in the algorithmic specification, decompose the array references into disjoint lattices

Step 2 Perform a lifetime analysis for all the lattices relative to the blocks of code (e.g., loop nests)

Step 3 Using the computation of 1-D windows for lattices, compute the window sizes for every indexed signal at the borderline between the blocks of code

Step 4 Adjust the window sizes taken into account the lattices that are both produced and consumed in a same block

Experimental Results

Application	#Array Refs.	#Scalars	Mem. Size / CPU (Troncon model)	Mem. Size / CPU (De Greef model)
Motion detection	11	318,367	9,525 / 12 sec	9,636 / 20 sec
Regularity detection	19	4,752	4,353 / 3 sec	3,879 / 9 sec
Gaussian blur filter	20	177,167	48,646 / 34 sec	50,448 / 76 sec
SVD updating	87	386,472	17,554 / 18 sec	16,754 / 48 sec
Voice coder	232	33,619	13,104 / 14 sec	13,224 / 25 sec

Tests on a PC with a 1.85 GHz Athlon XP processor

Experimental Results

Application	Mem. Size (Troncon)	Mem. Size (De Greef)	Σ Min Array Windows	Min Memory Size
Motion detection	9,525	9,636	9,525	9,524
Regularity detection	4,353	3,879	2,449	2,304
Gaussian blur filter	48,646	50,448	48,646	16,515
SVD updating	17,554	16,754	10,204	8,725
Voice coder	13,104	13,224	12,963	11,890

Analysis of the mapping models effectiveness
The last columns computed using the technique [ASP DAC 2006]

Conclusions

- Signal-to-memory mapping is a central issue in the memory allocation design for multimedia signal processing systems
- This paper has presented an algebraic framework allowing to implement two classic mapping models several times faster
- This paper has illustrated a better evaluation strategy for different mapping models, by computing the minimum array windows and the minimum data memory of applications

The End