Memory Size Computation for Multimedia Processing Applications

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

University of Illinois at Chicago

Outline

- Introduction: research context
- The memory size computation problem
- Storage requirements of an array reference
- Exact memory size computation
  using data-dependence analysis
- Experimental results
- Conclusions

Introduction: Research Context

Real-time multidimensional processing systems

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

- A large part of power dissipation is due to

  data transfer and data storage

Fetching operands from an off-chip memory for addition consumes 33 times more power than the computation

[Catthoor 98]

- Area cost often largely dominated by memories

Introduction: Research Context

In the early years of high-level synthesis

memory management tasks tackled at **scalar level**

Algebraic techniques -- similar to those used in modern compilers -- allow to handle memory management at non-scalar level

Requirement: addressing the entire class of affine specifications

- multi-dimensional signals with (complex) affine indexes
- loop nests having as boundaries affine iterator functions
- conditions – relational and / or logical operators of affine fct.

The Memory Size Computation Problem

```
optDelta[0] = 0                 //  Code derived from a motion
                                //       detection algorithm [ Chan 93 ]
for (i=8; i<=120; i++)
   for (j=6; j<=120; j++)
   {  Delta[i][j][0] = 0
      for (k=i-8; k<=i+8; k++)
         for (l=j-8; l<=j+8; l++)
            Delta[i][j][17*(k-i) + l – j + 145] = A[i][j] – A[k][l]@1
                                 + Delta[i][j][17*(k-i) + l – j + 144]
      optDelta[113*I + j - 911] = Delta[[i][j][289]
                                          + optDelta[113*I + j - 912]
   }
Opt = optDelta[12769]
```

The Memory Size Computation Problem

How many memory locations are necessary
to store the signals of a multimedia algorithm

Any scalar signal must be stored only during its lifetime

Signals having disjoint lifetimes can share the same location

The number of scalars in the illustrative algorithm:	3,749,063
The amount of necessary storage:	33,284

This paper presents an algorithm computing **exactly**
the amount of storage locations

Computation of Array Reference Size

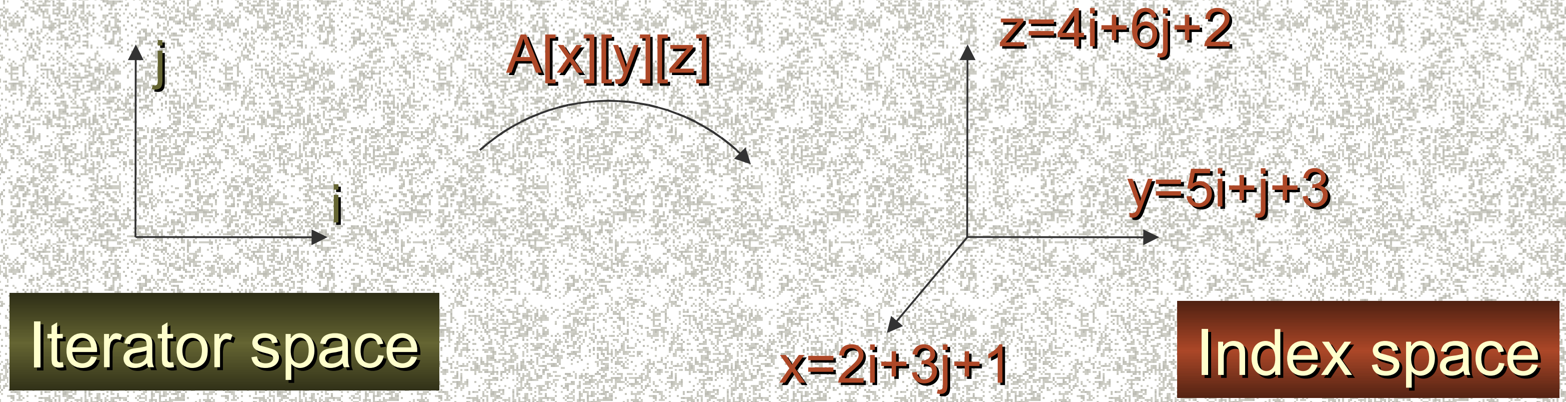
```
for (i=0; i<=4; i++)
    for (j=0; j <= 2i  &&  j <= -i+6; j++)
        …   A [2i+3j+1] [5i+j+3] [4i+6j+2]   …
```

How many memory locations are necessary to store the array reference

A [2i+3j+1] [5i+j+3] [4i+6j+2]

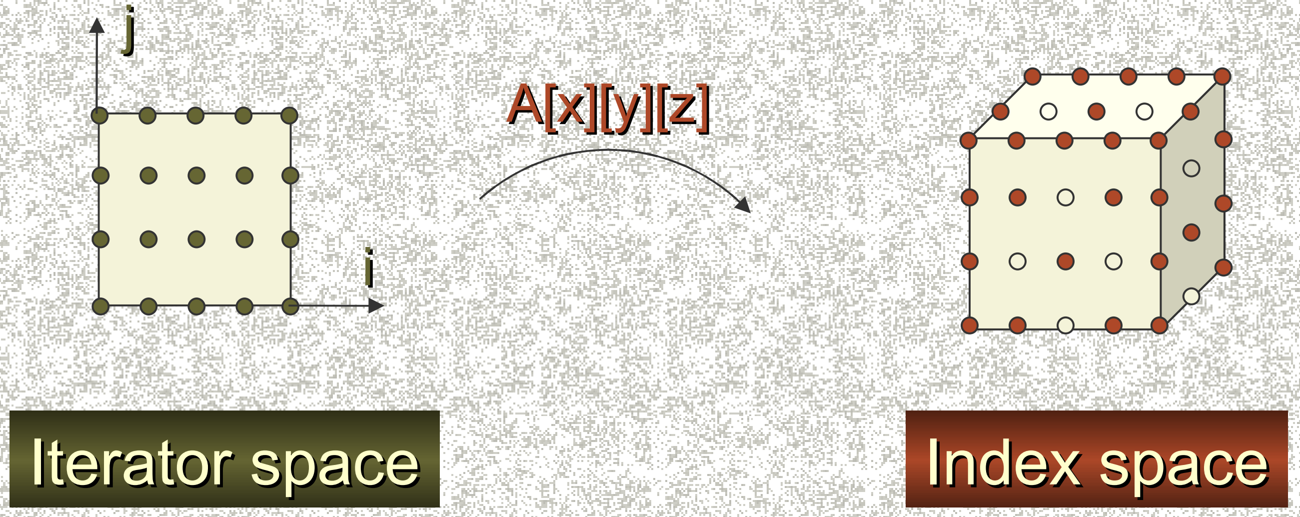
Computation of Array Reference Size

```
for (i=0; i<=4; i++)
    for (j=0; j <= 2i  &&  j <= -i+6; j++)
        …   A [2i+3j+1] [5i+j+3] [4i+6j+2]   …
```



Computation of Array Reference Size

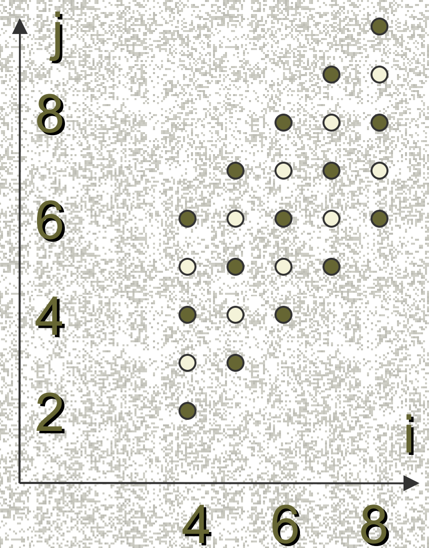
```
for (i=0; i<=511; i++)
    for (j=0; j<=511; j++)
        …   A [2i+3j+1] [5i+j+2] [4i+6j+3]   …
```



Computation of Array Reference Size

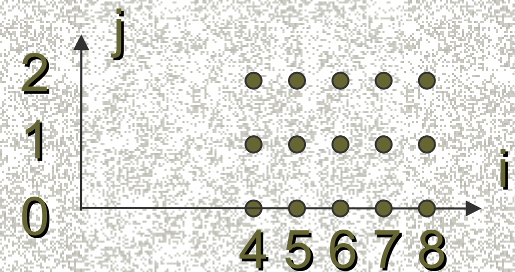
Remark   The iterator space may have ``holes'', too

```
for (i=4; i<=8; i++)
    for (j=i-2; j<=i+2; j+=2)    …  C[i+j]  …
```




```
for (i=4; i<=8; i++)
    for (j=0; j<=2; j++) … C[2i+2j-2] …
```

normalization



Computation of Array Reference Size

```
for (i=0; i<=4; i++)
    for (j=0; j <= 2i  &&  j <= -i+6; j++)
        …   A [2i+3j+1] [5i+j+3] [4i+6j+2]   …
```

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 1 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

Index space ← affine mapping — Iterator space

$0 \le i \le 4$, $0 \le j \le 2i$, $j \le -i+6$

Computation of Array Reference Size

Any array reference can be modeled as a linearly bounded lattice (LBL)

$$\text{LBL} = \{\, \mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u} \;\mid\; \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b} \,\}$$

Affine mapping

Iterator space

- scope of nested loops, and
- iterator-dependent conditions

LBL ← affine mapping ← Polytope

Computation of Array Reference Size

The size of the array reference
is the size of its index space – an LBL !!

LBL = { **x = T·i + u** | **A·i >= b** }

$f : Z^n \longrightarrow Z^m$ f(i) = T·i + u

Is function f a one-to-one mapping ??

↓ If YES

Size(index space) = Size(iterator space)

Computation of Array Reference Size

$f : Z^n \longrightarrow Z^m$ $\qquad f(i) = T \cdot i + u$

$$P \cdot T \cdot S = \begin{bmatrix} H & 0 \\ G & 0 \end{bmatrix}$$ Reduced Hermite Form

H - nonsingular lower-triangular matrix

S - unimodular matrix

P - row permutation

Computation of Array Reference Size

Case 1 rank(**H**)=n **function f is a one-to-one mapping**

```
for (i=0; i<=4; i++)
    for (j=0; j<=2i  &&  j<=-i+6; j++)
                                 … A [2i+3j+1] [5i+j+3] [4i+6j+2] …
```

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 1 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

$$P \cdot T \cdot S = I_3 \begin{pmatrix} 2 & 3 \\ 5 & 1 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} -1 & 3 \\ 1 & -2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -4 & 13 \\ \hdashline 2 & 0 \end{pmatrix} \begin{matrix} \mathbf{H} \\ \\ \mathbf{G} \end{matrix}$$

locations A[ ][ ][ ] = size (0 <= i <= 4 , 0 <=j <= 2i, -i+6) = 21

Computation of Array Reference Size

Case 2 rank(**H**)<n

function f is not a one-to-one mapping

```
for (i=0; i<=511; i++)
    for (j=0; j<=511; j++)   …
        for (k=0; k<=511; k++)   …   B [i+k] [j+k]   …
```

$$P \cdot T \cdot S = I_2 \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right)$$

$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{H} \qquad \mathbf{0}$$

{ 0 <= i , j , k <= 511 } ⇒ { 0 <= I-K , J-K , K <= 511 }

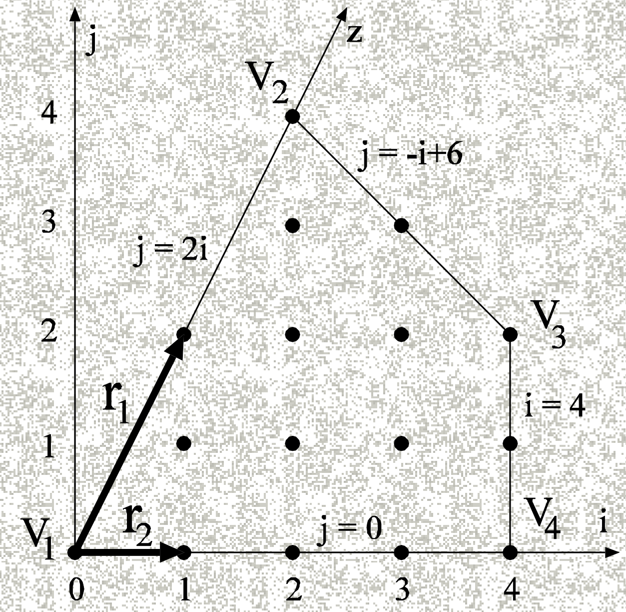
\# locations B = size (0<=I,J<=1022 , I-511<=J<=I+511) = 784,897

(whereas the size of the iterator space is 512^3 = 134,217,728)

Computation of Array Reference Size

Computation of the size of an integer polytope

```
for (i=0; i<=4; i++)
    for (j=0; j <= 2i  &&  j <= -i+6; j++)
      …   A [2i+3j+1] [5i+j+3] [4i+6j+2]
```



Step 1

Find the vertices of the iterator space and their supporting polyhedral cones

$$C(V_1) = \{ r_1 , r_2 \} = \left\{ \begin{pmatrix} 1 \\ 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$$

Computation of Array Reference Size

Computation of the size of an integer polytope (cont'd)

Step 2 Decompose the supporting cones into unimodular cones (Barvinok's decomposition algorithm)

$$C(V_1) = + \left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right\} + \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$

Step 3 Find the generating function of each supporting cone

$$F(V_1) = + \frac{1}{(1-xy^2)\,(1-y^{-1})} + \frac{1}{(1-y)\,(1-x)}$$

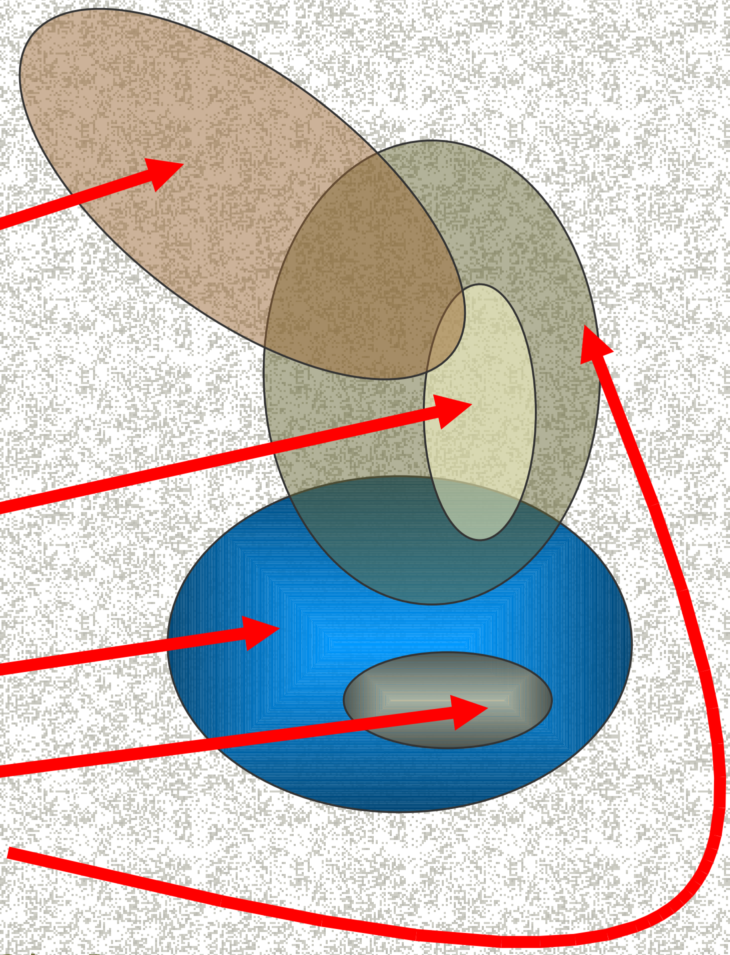
Step 4 Find the number of monomials in the generating function of the whole polytope $F = F(V_1) + F(V_2) + \ldots$

Memory Size Computation

```
# define n 6

for ( j=0; j<n ; j++ )
{   A [ j ] [ 0 ] = in0;
    for ( i=0; i<n ; i++ )
        A [ j ] [ i+1 ] = A [ j ] [ i ] + 1;
}

for ( i=0; i<n ; i++ )
{   alpha [ i ] = A [ i ] [ n+i ] ;
    for ( j=0; j<n ; j++ )
        A [ j ] [ n+i+1 ] =
            j < i    ?    A [ j ] [ n+i ] :
            alpha [ i ] + A [ j ] [ n+i ] ;
}
for ( j=0; j<n ; j++ ) B [ j ] = A [ j ] [ 2*n ];
```



Memory Size Computation

Decompose the LBL's of the array refs. into disjoint LBL's !!

$LBL_1 \cap LBL_2 \Rightarrow LBL$

$$\begin{cases} LBL_1 = \{ x = T_1 \cdot i_1 + u_1 \mid A_1 \cdot i_1 >= b_1 \} \\ LBL_2 = \{ x = T_2 \cdot i_2 + u_2 \mid A_2 \cdot i_2 >= b_2 \} \end{cases}$$

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

Diophantine system of eqs.

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

New polytope

Memory Size Computation

A0 = { x=j , y=0 | n-1>=j>=0 }

A1 = { x=j , y=i | n-1>=j>=0 , n-1>=i>=1 }

A2 = { x=0 , y=n }

A3 = { x=i , y=i+n | n-1>=i>=1 }

A4 = { x=i , y=i+n+1 | n-2>=i>=0 }

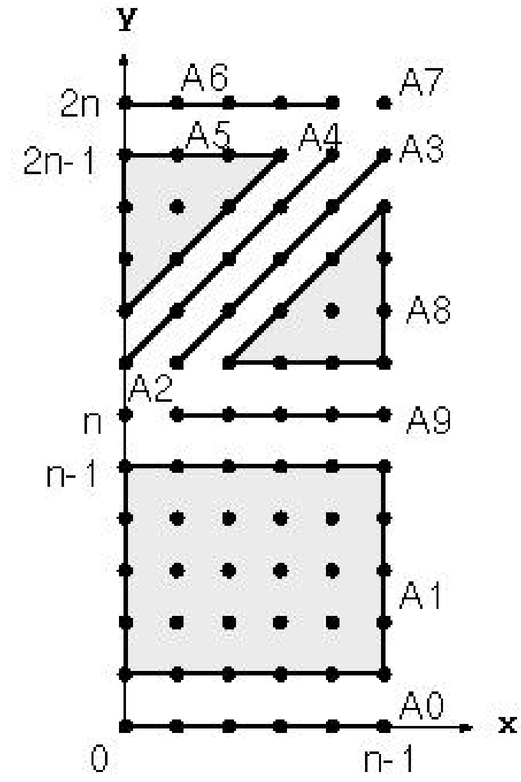
A5 = { x=j , y=i+n+1 | n-2>=i , j>=0 , i-j>=1 }

A6 = { x=i , y=2n | n-2>=i>=0 }

A7 = { x=n-1 , y=2n }

A8 = { x=j , y=i+n | n-1>=j , i>=1 , j-i>=1 }

A9 = { x=j , y=n | n-1>=j>=1 }



Decomposition of the array references of signal A (illustrative example)

Memory Size Computation Algorithm

Step 1 For every indexed signal in the algorithmic specification, decompose the array references in disjoint LBL's

Step 2 Based on the LBL lifetime analysis, find the exact memory size at the boundaries between the blocks of code

Step 3 Analyzing the amounts of signals produced and consumed In each block, prune the blocks of code where the maximum storage cannot happen

Step 4 For each of the remaining blocks, compute the maximum memory size

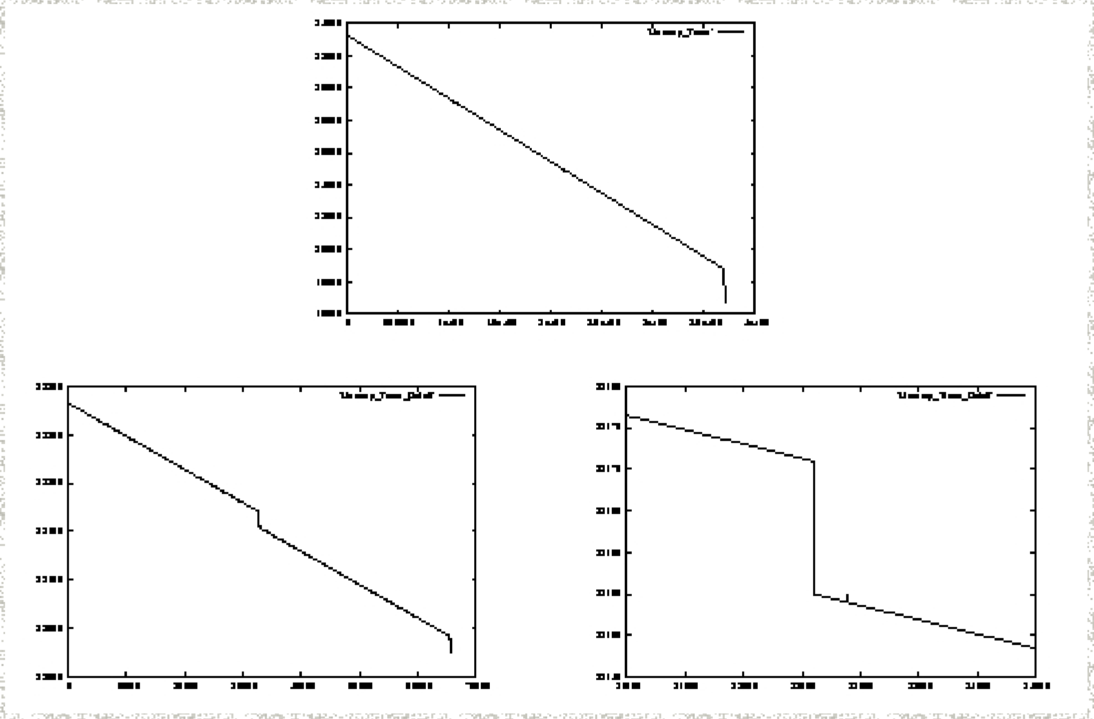
- exploiting the one-to-one mapping property of array references
- computing the maximum iterator vectors of the scalars

Experimental Results

Application (parameters)	# LBL's	# Scalar signals	# Storage locations	CPU sec
Regularity detection (MaxGrid=8, L=64)	28	4,752	2,304	< 1
Vocoder kernel	247	33,619	11,890	2
Motion detection (M=N=32, m=n=4)	583	72,543	2,740	2
(M=N=64, m=n=4)	1,255	318,367	9,524	11
Durbin alg. (N=500)	6,986	252,499	1,249	18
Dynamic prog. (N=500)	6,474	21,082,751	124,751	78
2D Gauss. blur filter (M=N=500)	7,479	2,735,027	250,005	103

Tests on a PC with a 1.85 GHz Athlon XP processor

Experimental Results



Memory size variation in the illustrative example

- abscissas: the number of data-path instructions in the code
- ordinates: the number of storage locations

The global maximum: x=2 , y=33,284

Conclusions

- The storage of multi-dimensional signals has a significant influence on both the area and power consumption in multimedia processing systems
- Algebraic techniques are powerful non-scalar instruments in the memory management of multimedia signal processing
- This paper has presented a non-scalar approach for the exact computation of storage requirements in real-time multimedia algorithms

The End