Control Flow Analysis for Recursion Removal

Stefaan Himpe, Geert Deconinck
K.U.Leuven ESAT/ELECTA
Francky Catthoor
IMEC

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Introduction

- Recursion removal
  - Traditionally done to reduce resource consumption (time and memory)
  - Now meant as enabling step for other transformations, typically on imperative or OO code (C/C++/JAVA/...)
    - Usually dependency removal transformations, and parallelising transformations
    - Hence, main goals:
      - Try to introduce as little new dependencies in iterative resulting algorithm implementation as possible
      - Guarantee correctness in presence of side-effects
    - However, it is still useful to evaluate impact on execution time, memory usage

© ESAT/ELECTA KULeuven'2003

imec

KU LEUVEN

ESAT

electa

© ESAT/electa 2003

Slide by Tanja Vanachteren, used with permission, copyright IMEC

Visual texture coding

High quality texture

Low quality texture

AC Reset

... TM ... TM ... TM ...

2D texture

3D object

AC = Arithmetic Coder
TM = Texture marker

- Part of the MPEG-4 standard
- Transmission of still images
- Scalable
  - Quality: successive quantisation
  - Resolution: Wavelet based
  - Region of Interest (ROI) selection

© ESAT/ELECTA KULeuven'2003

KU LEUVEN

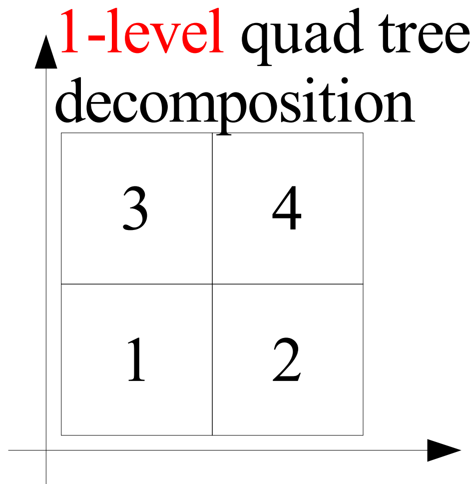
ESAT

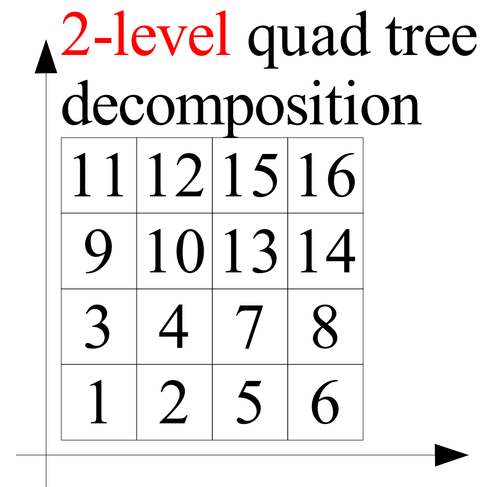
electa

© ESAT/electa 2003

Recursive part of the VTC algorithm

- MPEG 4 VTC





VTC algorithm (partial)

```
Decode (d, x, y) {
  if (d= =0) {
    DecodePixel(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d= =4) Check();
    Decode(d,x+k,y);
    if (d= =4) Check();
    Decode(d,x,y+k);
    if (d= =4) Check();
    Decode(d,x+k,y+k);
    if (d= =4) Check();
}
```

© ESAT/ELECTA KULeuven'2003

Recursive part of the VTC algorithm

- Time spent in this part of VTC is 50% of overall time
- DecodePixel is expensive compared to argument trafo and recursive function calls
- Between recursive calls extra functionality is present
  - It has side-effects
  - It must be preserved

VTC algorithm (partial)

```
Decode (d, x, y) {
  if (d= =0) {
    DecodePixel(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d= =4) Check();
    Decode(d,x+k,y);
    if (d= =4) Check();
    Decode(d,x,y+k);
    if (d= =4) Check();
    Decode(d,x+k,y+k);
    if (d= =4) Check();
}
```

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Recursive part of the VTC algorithm

- First step:
  - DecodePixel is expensive
  - DecodePixel is activated with argument values that are transformed through recursion

  Hence
  - First collect argument values, store in memory
  - Then use in iteration with DecodePixel calls

VTC algorithm (partial)

```
Decode (d, x, y) {
  if (d= =0) {
    DecodePixel(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d= =4) Check();
    Decode(d,x+k,y);
    if (d= =4) Check();
    Decode(d,x,y+k);
    if (d= =4) Check();
    Decode(d,x+k,y+k);
    if (d= =4) Check();
}
```

© ESAT/ELECTA KULeuven'2003

imec

KU LEUVEN ESAT electa

© ESAT/electa 2003

Recursive part of the VTC algorithm

- Second step:
  - Storing collected values in memory is expensive

  Hence

  - Replace recursive information collection with a formula producing these same argument values

VTC algorithm (partial)

```
Decode (d, x, y) {
  if (d= =0) {
    DecodePixel(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d= =4) Check();
    Decode(d,x+k,y);
    if (d= =4) Check();
    Decode(d,x,y+k);
    if (d= =4) Check();
    Decode(d,x+k,y+k);
    if (d= =4) Check();
}
```

© ESAT/ELECTA KULeuven'2003

imec

KU LEUVEN

ESAT

electa

© ESAT/electa 2003

Step 1: separate base case calculation from recursion

- Record all information that is needed in base case, in the original recursive algorithm without base case calculation
  - Applied to VTC:
    - Leave out call to DecodePixel
    - Add recording of d, x, y values in algorithm
- Loop over all collected information
  - Each time calling DecodePixel with correct argument values retrieved from memory
  - This loop is fully parallelizable, if the side-effects inside DecodePixel do not interfere

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Pseudo-code result

VTC algorithm (info collection)

```
Decode (d, x, y) {
  if (d= =0) {
    info[++cnt]=(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d==4) deq4[cnt]=1;
    Decode(d,x+k,y);
    if (d==4) deq4[cnt]=1;
    Decode(d,x,y+k);
    if (d==4) deq4[cnt]=1;
    Decode(d,x+k,y+k);
    if (d==4) deq4[cnt]=1;
}
```

VTC algorithm (iterative part)

```
ItDecode(d,x,y) {
  Decode(d,0,0);
  for (i=0;i<cnt;i++) {
    DecodePixel(info[i]);
    if (deq4[i])

      Check();
  }
}
```

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

What has happened up to now?

- Recursive part of algorithm
  - Collects information
  - No longer dominates execution time
- Iterative part of algorithm
  - Uses collected information
  - Method itself does not impose inter-iteration dependencies
    - Hence: opportunity for parallelization
- Drawback
  - Storing all information needs a lot of memory, certainly a lot more than was needed in the recursion; bad for energy consumption

© ESAT/ELECTA KULeuven'2003

Step 2: replace recursive info collection with iterative info generation

- Determine iteration bound
  - Call to Decode(d,_,_) results in 4 calls to Decode(d-1,_,_)
  - Number of iterations easily found by solving recurrence equation

$$I(d) = 4\,I(d-1) \text{ with } I(0) = 1$$

  - With solution

$$I(d) = 4^d$$

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Step 2: replace recursive info collection with iterative info generation

- Modeling of argument transformations
  - Call to Decode(d,x,y) results in 4 calls:
    Decode(d-1, x , y);
    Decode(d-1, x+k, y);
    Decode(d-1, x , y+k);
    Decode(d-1, x+k, y+k);

    (with k = 2^(d-1))
  - Before base case is reached, arguments have undergone a large amount of additions with different k-values = argument transformations

© ESAT/ELECTA KULeuven'2003

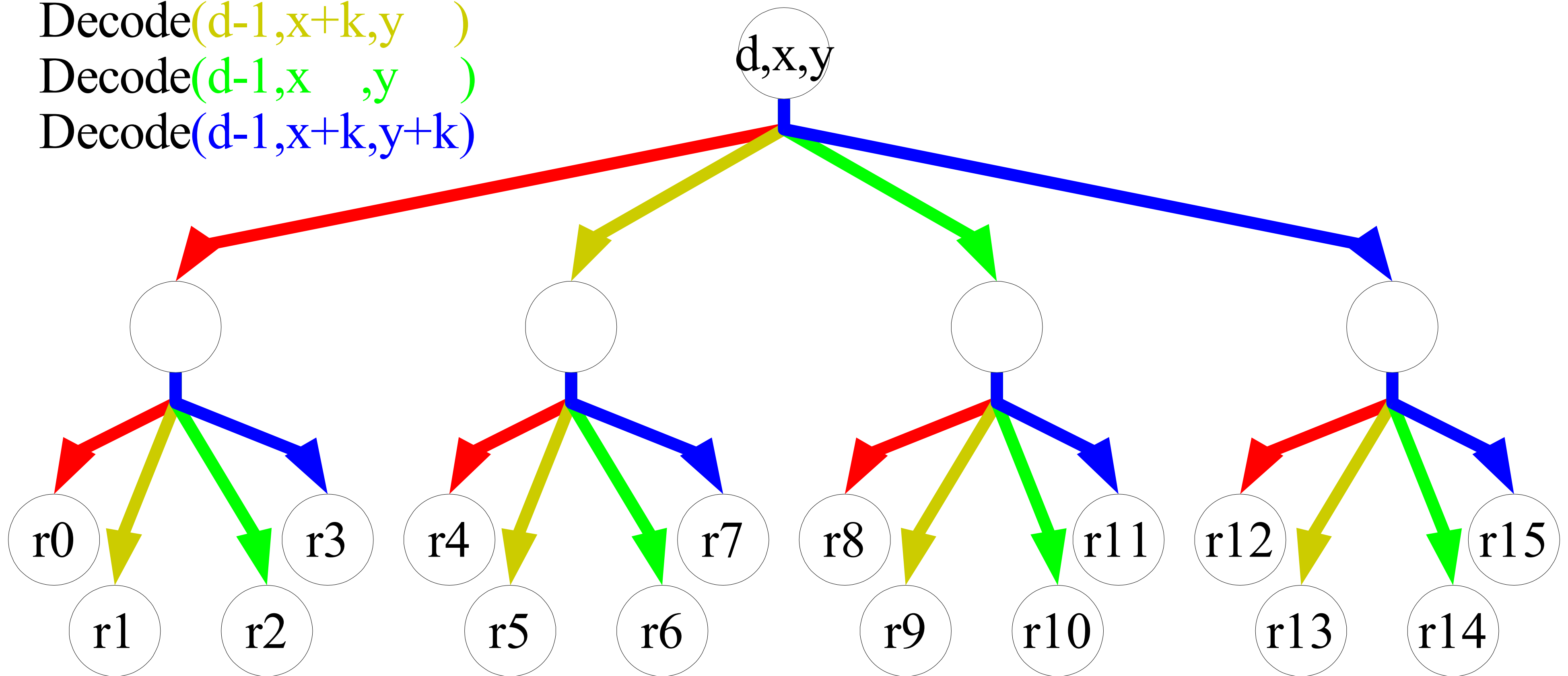
© ESAT/electa 2003

Call tree

Copyright
ESAT/ELECTA

Example: call to Decode(2,x,y) results in 4 calls:

Decode(d-1,x ,y)
Decode(d-1,x+k,y)
Decode(d-1,x ,y)
Decode(d-1,x+k,y+k)



Problem:

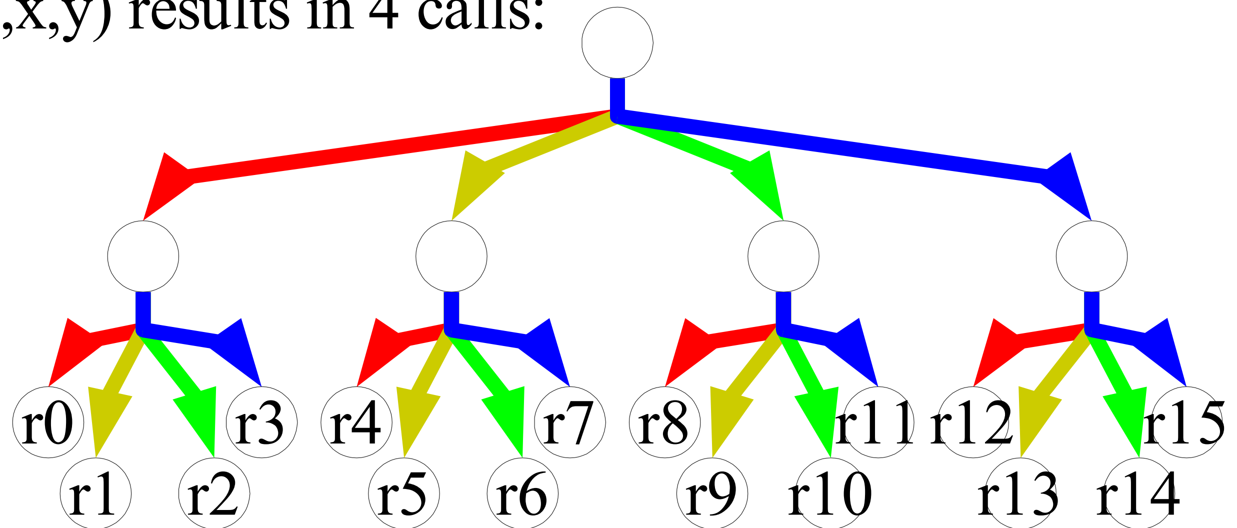
How is (d,x,y) transformed through the recursion when reaching the base case ?

Copyright
ESAT/ELECTA

Call tree

Example: call to Decode(2,x,y) results in 4 calls:

Decode(d-1,x ,y)
Decode(d-1,x+k,y)
Decode(d-1,x ,y)
Decode(d-1,x+k,y+k)



Answer:
(for mathematical derivation: see paper)

$$r_j = \begin{bmatrix} 0 \\ \sum_{i=0}^{n-1} 2^i \left(\left(j \operatorname{div} 2^{2i} \right) \operatorname{mod} 2 \right) \\ \sum_{i=0}^{n-1} 2^i \left(\left(j \operatorname{div} 2^{2i+1} \right) \operatorname{mod} 2 \right) \end{bmatrix}$$

What has happened up to now?

VTC algorithm (info collection)

```
Decode (d, x, y) {
  if (d= =0) {
    info[++cnt]=(x,y);
  } else {
    --d; k=1<<d;
    Decode(d,x,y);
    if (d==4) deq4[cnt]=1;
    Decode(d,x+k,y);
    if (d==4) deq4[cnt]=1;
    Decode(d,x,y+k);
    if (d==4) deq4[cnt]=1;
    Decode(d,x+k,y+k);
    if (d==4) deq4[cnt]=1;
}
```

- Formula for rj can be used to replace non-grayed out part of VTC with iteration
  - Synthesize a loop that calculates rj values for j = 0 .. 4^d
- We have ignored the extra functionality between the recursive Decode calls
- Let's put it back in !

© ESAT/ELECTA KULeuven'2003

imec

LEUVEN

ESAT

electa

© ESAT/electa 2003

Handling intermediate functionality

- Problem: when must activation of

  "if (d==4) Check()"

  be scheduled, to preserve correctness in presence of side-effects inside Check() function ?

- Strategy:
  - We look for a relation R(j) between iteration counter j (which calculates successive rj's) and the result of condition (d == 4)
  - Then iterative algorithm calculates rj, followed by checking condition R(j) and possible activation of intermediate functionality

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Handling intermediate functionality

- Consider call to Decode(4,_,_)
  - This results in 4 calls to Decode(3,_,_)
  - Each of these 4 calls results in 4 calls to Decode (2,_,_) => 4*4 = 16 calls
  - ...
  - Results in 4*4*4*4 = 256 calls to Decode(0,_,_)
    (call to Decode(0,_,_) is what activates base case calculation!)
- Conclusion:
  - Between two calls to Decode(4,_,_) there are 4^4 = 256 calls to Decode(d,_,_) with d < 4
  - Hence (d == 4) is true if j is multiple of 256 !
  - Similarly (d > 4) is true if j is multiple of 1024

© ESAT/ELECTA KULeuven'2003

imec KU LEUVEN ESAT electa

© ESAT/electa 2003

Handling intermediate functionality

- Hence the following relation has been found

$$d = 4$$
$$\Leftrightarrow$$
$$j \, mod \, 256 = 0 \wedge j \, mod \, 1024 \neq 0$$

- Because 256 and 1024 are powers of two, this is cheaply implemented by bit masking

© ESAT/ELECTA KULeuven'2003

imec

KU LEUVEN

ESAT

electa

© ESAT/electa 2003

Resulting iterative VTC core

- Iterative version

for j = 0 to $4^n - 1$ {

// calculate argument values

$$x_bc = \sum_{i=0}^{n-1} 2^i \left(\left(j \operatorname{div} 2^{2i} \right) \bmod 2 \right);$$

$$y_bc = \sum_{i=0}^{n-1} 2^i \left(\left(j \operatorname{div} 2^{2i+1} \right) \bmod 2 \right);$$

// activate base case calculation

DecodePixel(x_bc,y_bc);

// activation intermediate functionality

if $\left(\left(j \bmod 256 = 0 \right) \&\& \left(j \bmod 1024 \neq 0 \right) \right)$

Check();

}

© ESAT/ELECTA KULeuven'2003

imec

KU LEUVEN

ESAT

electa

© ESAT/electa 2003

Intermediate conclusions

- What have we done up to now?
  - Separate base case calculation from recursion
    - To isolate dominant part from recursion
  - Replace argument recording with argument calculation
    - To remove argument recording memory usage
    - Solution to three sub-problems was required
      - Determination of iteration bound
      - Modeling argument transformations through the recursion
      - Activation of intermediate functionality at correct moments in time

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

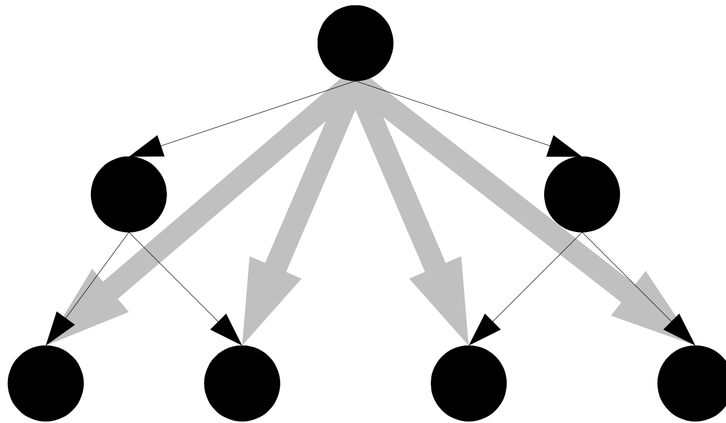
Intermediate conclusions

- How general is this approach ?
  - Has been generalised and extended to handle recursive algorithms with any combination of
    - Multiple base cases
    - Multiple recursive cases
    - Intermediate functionality which depends on any combination of recursive function arguments (or other arguments, e.g. global variables)
    - Any kind of argument transformation functions
    - Functions with return values and functions to combine return values to new return values
  - In the most general case all functions may have side-effects
  - So, it is quite general!

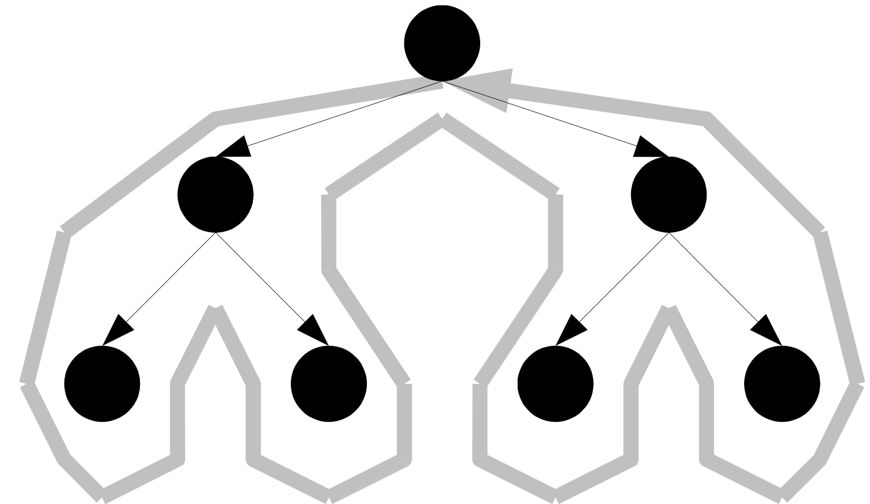
© ESAT/ELECTA KULeuven'2003

Comparison iterative - recursive

- Iterative algorithm



- Recursive algorithm



- Traverses each path from root node to leaf
- Arrows closer to root are evaluated more often than arrows nearer to leaf

- Performs depth-first traversal of call tree
- Each arrow is evaluated exactly once

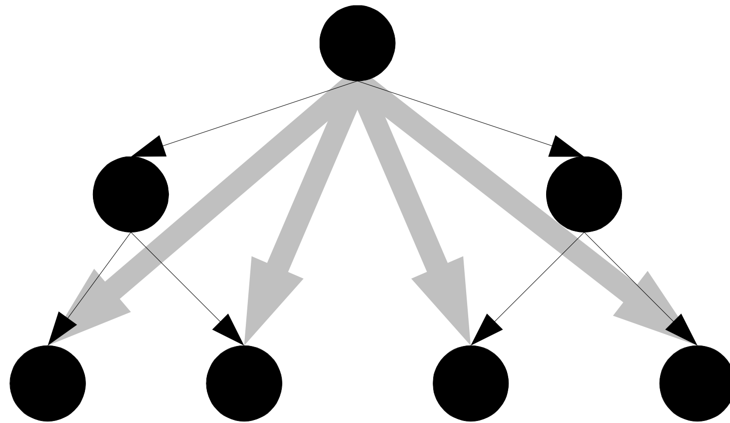
imec

© ESAT/ELECTA KULeuven'2003

KU LEUVEN ESAT electa

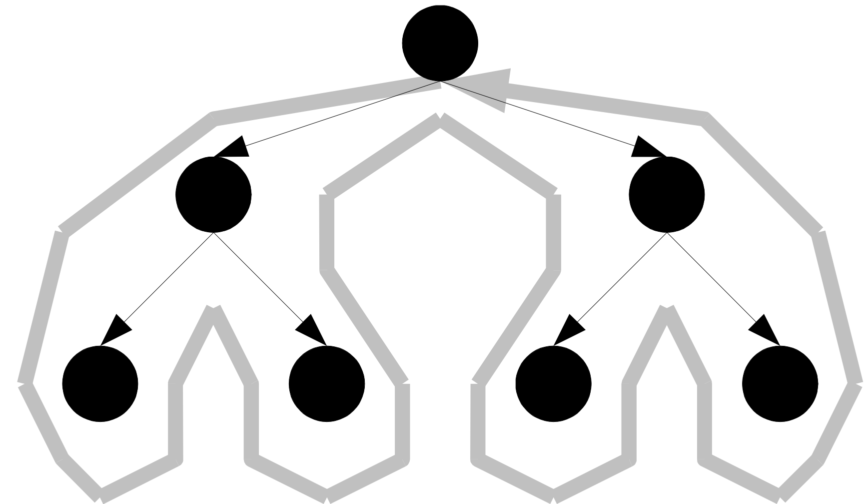
Comparison calculation requirements

- Iterative algorithm



- Each arrow represents
  - Function call
  - Argument trafo step
- There are (at most) $\frac{B^{D_{MAX}+1}-1}{B-1}-1$ arrows

- Recursive algorithm



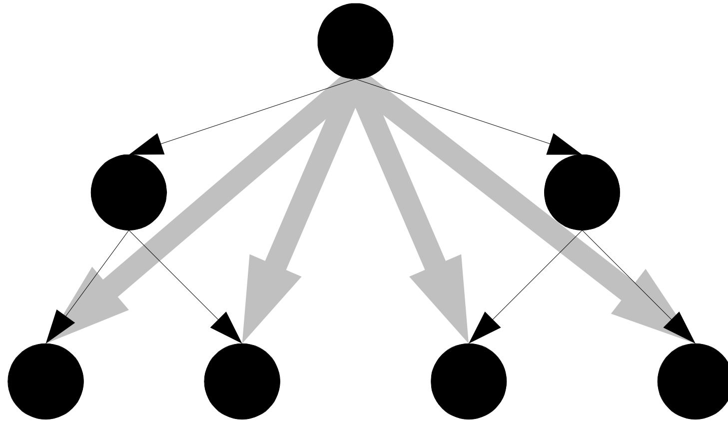
- Each arrow represents
  - Argument trafo step
- Arrow from depth d to depth d+1 is redone (at most) $B^{D_{MAX}-1}$ times
- From depth d to d+1 there are B^{d+1} arrows
- Total #calcs = $D_{MAX} B^{D_{MAX}}$

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

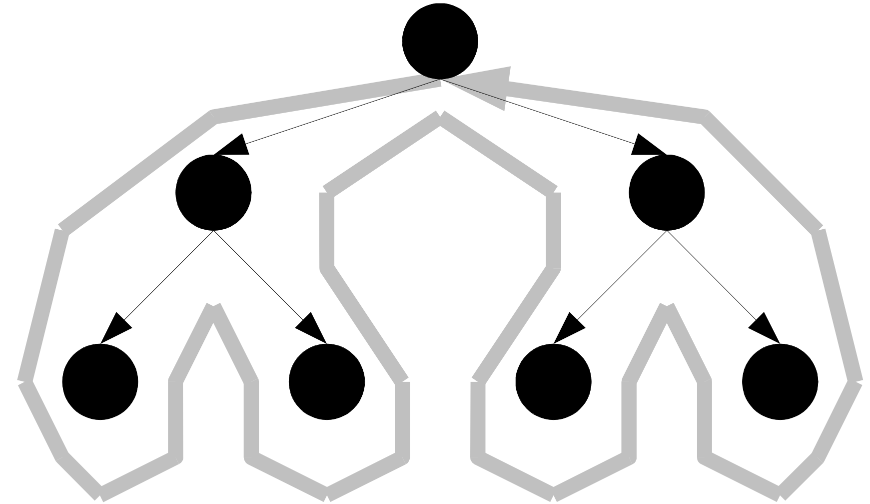
Comparison memory requirements

- Iterative algorithm



- Some fixed amount of book keeping memory required
- No extra memory depending on recursion depth

- Recursive algorithm



- Memory required is at most

  = stack frame size X recursion depth

© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Systematic trade-offs

imec
© ESAT/ELECTA KULeuven'2003
KU LEUVEN
ESAT
electa

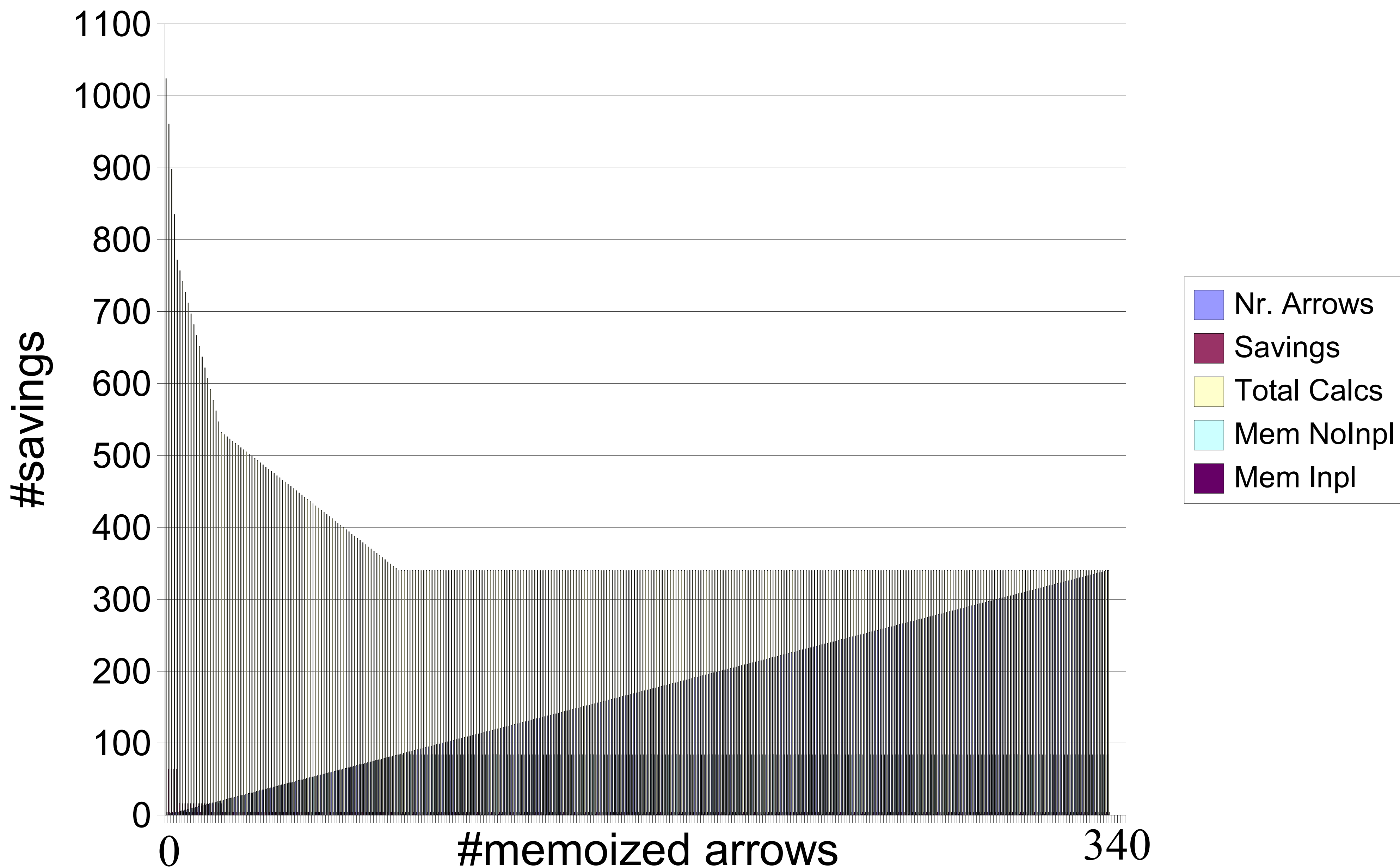
- First trade-off:
  - Trade-off argument transformation calculations verus memory used
  - By applying partial memoization
- Second trade-off
  - Trade-off memoization memory for parallelizability
  - By applying in-place optimization

First trade-off: calculations vs. memory

- Storing results from argument transformation step going from depth d to d+1 reduces number of argument transformation steps with an amount of $B^{D_{MAX}-d}-1$
- Memoization of leaf does not have advantage (leaf is needed only once, $B^{D_{MAX}-D_{MAX}}-1=0$)
- Storing each result in separate memory location
  - Reduces amount of calculations to amount in recursive algorithm
  - But uses up to $\frac{B^{D_{MAX}+1}-1}{B-1}-1$ memory locations
  - Recursive version needs only D_{max} locations
- This version has potentially full parallelisability, but this is usually not needed

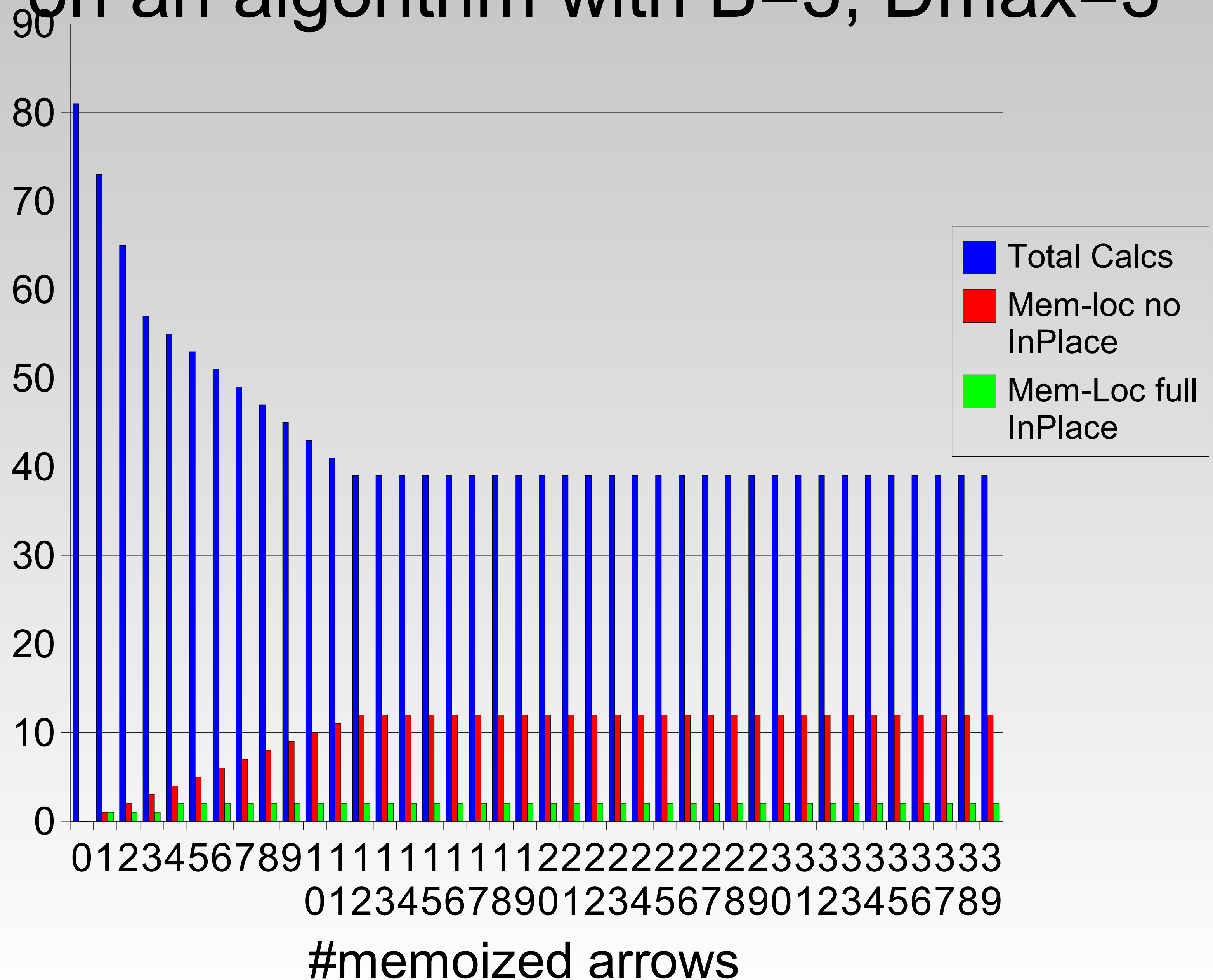
© ESAT/ELECTA KULeuven'2003

Effect of memoization on #calcs, #mem



Copyright
ESAT/ELECTA

Effect of memoization on #calcs, #mem on an algorithm with B=3, Dmax=3



© ESAT/ELECTA KULeuven'2003

© ESAT/electa 2003

Second trade-off: memory vs. parallelisability

- Not all argument transformation results are needed at the same time
  (unless fully parallelised version is desired)
- Hence memory locations can be reused
- By full memoization and fully exploiting limited life-times
  - Can achieve same time and memory complexity as recursive algorithm or better
  - This depends on such factors as
    - Known lack of side-effects in certain parts of the algorithm
    - Possibilities for symbolically simplifying r_j - equation

© ESAT/ELECTA KULeuven’2003

© ESAT/electa 2003

Summary of trade-offs

	Recursive calls	Argument trafo calcs	Memory locations	Parallel. oportunities
Recursive version	$\frac{B^{D_{MAX}+1}-1}{B-1}-1$	$\frac{B^{D_{MAX}+1}-1}{B-1}-1$	$S.D_{MAX}$	none
Iterative version	0	$D_{MAX}B^{D_{MAX}}$	0	full
Iterative + partial memoization	0	Between $D_{MAX}B^{D_{MAX}}$ and $\frac{B^{D_{MAX}+1}-1}{B-1}-1$	Between 0 and $\frac{B^{D_{MAX}+1}-1}{B-1}-1$	full
Iterative + memoization + in-place	0	Between $D_{MAX}B^{D_{MAX}}$ and $\frac{B^{D_{MAX}+1}-1}{B-1}-1$	Full memo: between $\frac{B^{D_{MAX}+1}-1}{B-1}-1$ and $S'.D_{MAX}$	Between full and none

© ESAT/ELECTA KULeuven'2003

Summary and future work

- Method for removing and analysing recursion in quite general cases
  - Here applied on VTC algorithm
  - Many other examples have been transformed as well
- Recursion removal on VTC enabled further transformation to get energy reduction of up to factor 2 (without reduction of execution time). For details, see:

Zhe Ma, Chun Wong, Stefaan Himpe, Eric Delfosse, Francky Catthoor and Geert Deconinck, "Task concurrency analysis and exploration of Visual Texture Coder on a Heterogeneous Platform", in: Proceedings of the 2003 IEEE workshop on signal processing systems (SIPS03)

imec

© ESAT/ELECTA KULeuven'2003

KU LEUVEN

ESAT

electa

© ESAT/electa 2003

The End...

...thank you!

© ESAT/ELECTA
KULeuven'2003

© ESAT/electa 2003