From Loop Transformation to Hardware Generation

Harald Devos
ELIS/PARIS
Ghent University
Introduction: 2D-IDWT

- **RESUME project** (Reconfigurable Embedded Systems for Use in scalable Multimedia Environments)

- Build real-time decoder for scalable video

- Software profiling: Hardware acceleration needed

- 2-Dimensional Inverse Discrete Wavelet Transform: 2D-IDWT
Introduction: 2D-IDWT

- Inverse Discrete Wavelet Transform
Introduction: Specifications

• Real-time:
  - 45 frames/s
  - CIF resolution (288x352 pixels)

• Target platform:
  - FPGA-board
  - Estimated clock speed: 50 MHz
Introduction

FPGA: Field Programmable Gate Array
e.g.: Altera Stratix
Introduction: 1st Design

- Simulation results:
  - 869530 cycles/frame
- Synthesis results
  - Clock speed: 68.91 MHz
- Expectation: 79 frames/s
- Measurements on hardware:
  - 29 frames/s: Memory bottle neck !!!
Introduction:
Memory bottle neck

• Memory hierarchy
  - large but slow external memory
  - fast but smaller (parallel) on-chip memory

• External memory often bottle neck
  - Minimize accesses to external memory
  - Increase reuse of data stored in on-chip buffers
Bandwidth Limited

Calculation Limited

Available Bandwidth to external Memory (MB/s)

Frame rate (frames/s)

Manual RC

Bandwidth Limited

Calculation Limited
Introduction: Problem

- Memory bottle neck
- Design automation needed
  - Manual design process = slow, error-prone, ...
  - Lots of designs to be made
    - Reconfigurable HW (QoS)
    - Different platforms
Introduction: Solution

• Loop Transformations
  - improve spatial and temporal locality of data accesses
  - polyhedral model
  - common practice for software

• Hardware Generation
  - from the polyhedral model
Overview

- Introduction: 2D-IDWT, 1\textsuperscript{st} try
- Overview
- Loop transformations
- Hardware Generation
- Experiment: 2D-IDWT, 2\textsuperscript{nd} try
- Conclusions and Future Work
Loop Transformations

Original algorithm in, e.g., C

Representation in the Polyhedral Model

Optimized algorithm in, e.g., C

Optimized algorithm in HW (VHDL)

Loop Transformations
The Polyhedral Model

Control flow

Iteration domains

Statement definitions

Memory accesses

for i = 0 .. N
  for j = 0 .. N-i
    if i==0
      B[j]=1;
      B[i+j]=B[i+j]*A[i][j];
    S1(i,j);
    S2(i,j);
The Polyhedral Model

Iteration domains:

\[
\begin{align*}
\text{for } & i = 0 \ldots N \\
\text{for } & j = 0 \ldots N-i \\
\text{if } & i == 0 \\
& S1(i,j); \\
& S2(i,j);
\end{align*}
\]

Memory accesses:

- \( S1(i,j): W B[j] \)
- \( S2(i,j): R/W B[i+j] \) \( R A[i][j] \)

Accesses to \( B[N-1] \)

Execution order
From Loop Transformation to Hardware Generation – Harald Devos – 2006-11-24
17th ProRISC Workshop, November 2006, Veldhoven, The Netherlands

The Polyhedral Model

\[ \begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} \]

\[ \begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} \]

Transformation

Accesses to B[N-1]

- S1(0,i'): W B[i']
- S2(i'-j',j'): R/W B[i'] R A[i-j][j]
The Polyhedral Model

Polyhedral representation
Iteration domains

Code generation
(CLooG)

Statement definitions

Optimized program

for \( i = 0 \ldots N \)
  \( S1(0,i); \)
  for \( j = 0 \ldots i \)
    \( S2(i-j,j); \)

for \( i = 0 \ldots N \)
  \( \text{B}[i]=1; \)
  for \( j = 0 \ldots i \)
    \( \\text{B}[i]=\text{B}[i]\times\text{A}[i-j][j]; \)

\( S1(0,i): \)
  \( \text{B}[i]=1; \)

\( S2(i-j,j): \)
  \( \text{B}[i]=\text{B}[i]\times\text{A}[i-j][j]; \)
The Polyhedral Model

• SCoP: Static Control Part
  – Part of program with data independent control flow
  – Typically set of nested loops (hot code)

• Loop bounds are linear expressions of parameters and other iterators
The Polyhedral Model

- More flexibility than syntactical representation
  - Long transformation sequences possible
- Used for SW-optimization
  - Tools: CLooG, WRaP-IT/URUK, ...
  - Less used for HW-generation
- CLooG extended: generation of HW-controller (sequential until now)
Hardware generation

for i = 0 .. N
  for j = 0 .. N-i
    if i==0
      B[j]=1;
      B[i+j]=B[i+j]*A[i][j];
    S1(i,j);
    S2(i,j);
Hardware generation

| Loop control | Depends on loop transformation |
| Statement Datapath+ctrl | Reusable for different transformations |
| Memories | 1 memory/array (1 buffer/array, refine to memory hierarchy) |
Hardware architecture

for $i$
  S1($i$)
  for $j$
    S2($i,j$)
    S3($i,j$)
  for $k$
    S4($k$)
    S5($k$)
    S6($k$)
  S7()
**Experiment: 2D-IDWT**

- Find good loop transformations
  - SLO: Suggestions for Locality Optimizations
- Apply transformations
  - URUK
- Generate Hardware
  - CLooGVHDL
Experiment: 2D-IDWT

```c
void fix_idwt2_comp(int** F, int rows, int cols, int K)
    /* performs the 2-D DWT in <K> levels and returns the result */
{
    int size;
    int i, j, l;
    /* Start IDWT */
    for (l=K-1; l>=0; l--) {
        /* Do the column filtering first (the opposite from the decomposition) */
        for (j=0; j<(cols>>l); j++) {
            vert_idwt_comp(l, F, j, rows);
        }
        /* Now, do the row-filtering as well using the results of the column-filtering */
        for (i=0; i<(rows>>l); i++) {
            hor_idwt_comp(l, F, i, cols);
        }
    }
    /* End IDWT */
}
```

SLO: Suggestions for Locality Optimizations
Experiment: 2D-IDWT

```c
void fix_idwt2_comp(int** F, int rows, int cols, int K)
    /* performs the 2-D DWT in <K> levels and returns the result */
{
    int size;
    int i, j, l;
    /* Start IDWT */
    for (l=K-1; l>=0; l--) {
        /* Do the column filtering first (the opposite from the decomposition) */
        for (j=0; j<(cols)>>l; j++) {
            vert_idwt_comp (l, F, j, rows);
        }
        /* Now, do the row-filtering as well using the results of the column-filtering */
        for (i=0; i<(rows)>>l; i++) {
            hor_idwt_comp (l, F, i, cols);
        }
    }
    /* End IDWT */
}
```

SLO: Suggestions for Locality Optimizations
Experiment: 2D-IDWT

• Transformation sequences in URUK-script: e.g., Fusion

addContext(enclose(LBLK2_NB),'rows8>=9')
addContext(enclose(LBLK2_NB),'cols8>=11')
...
stretch2(enclose(LBL_VERTS_NB_K0S5),2)
stretch2(enclose(LBL_HOR_NB_K0S5),2)
interchange(enclose(LBL_VERTT_NB_K0S5,2))
interchange(enclose(LBL_VERTS_NB_K0S5,2))
fusion(enclose(LBL_VERTT_NB_K0S5,3))
fusion(enclose(LBL_VERTT_NB_K0S5,2))
fusion(enclose(LBL_VERTT_NB_K0S5,2))
...

Fusion
Experiment: 2D-IDWT

• Generation of control block:
  - fully automatically: CLooGVHDL
  - sequential execution of statements

• Generation of statements:
  - small tools and ad-hoc scripts
  - manual optimizations possible
  - reusable for different transformations
## Experiment: 2D-IDWT

- Data flow to external memory

<table>
<thead>
<tr>
<th>Variant</th>
<th>Data flow</th>
<th>Burst Usage</th>
<th>Variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>5.25 RC</td>
<td>50%</td>
<td>RC-based</td>
</tr>
<tr>
<td>Fused</td>
<td>2.625 RC</td>
<td>100%</td>
<td>line-based</td>
</tr>
<tr>
<td>Fused+Tiled</td>
<td>2 RC</td>
<td>100%</td>
<td>stripe-based</td>
</tr>
</tbody>
</table>
Conclusions

- Memory bottle neck -> Loop transformations
  - SW-techniques can be reused for HW
  - Polyhedral model eases transformations

- Design automation
  - CLooGVHDL: hardware generation from the polyhedral model
Future work

- Methodology for finding transformations/schedule
- Introduce parallelism in techniques
- Methods for building memory hierarchy and scheduling memory transactions
Introduction

Scalable applications e.g. videocodec

Different designs for different HW platforms

Several configurations for a certain reconfigurable platform

Huge set of possible designs

Design automation needed
Introduction

Multimedia applications

- Flexibility of application. Characteristics (QoS) modified at run-time.
- High-computational (often real-time) requirements.

Conflict

- Software
- Hardware

FPGA: combine flexibility with high computational power
Memory bottle neck

• Use loop transformations to improve spatial and temporal locality of data accesses.

• Common practice for SW-optimization (e.g. cache behavior)

• Differences with HW-design?
SW  <->  HW

• Adapt application towards architecture
• Cache policy (e.g. LRU)
• Vertical memory hierarchy
• Sequential

• Adapt architecture towards application
• Data transfers are explicit
• Horizontal memory hierarchy
• Parallel
Experiment: 2D-IDWT

```c
void fix_idwt2_comp(int** F, int rows, int cols, int K)
    /* performs the 2-D DWT in <K> levels and returns the result */
{
    int size;
    int i, j, l;
    /* Start IDWT */
    for (l = K - 1; l >= 0; l--) {
        /* Do the column filtering first (the opposite from the decomposition) */
        for (j = 0; j < (cols >> l); j++) {
            vert_idwt_comp(l, F, j, rows);
        }
        /* Now, do the row–filtering as well using the results of the column–filtering */
        for (i = 0; i < (rows >> l); i++) {
            hor_idwt_comp(l, F, i, cols);
        }
    }
    /* End IDWT */
}
```
Experiment: 2D-IDWT

![Graph showing the relationship between log2(Reuse Distance) and # Accesses for different optimizations.](image-url)