Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study (ADRES, DATE ’04)

B. Mei, S. Vernalde, D. Verkest, R. Lauwereins
Katholieke Universiteit Leuven, Vrije Universiteit
1. Introduction
2. ADRES Architecture Overview
3. C-Based Design Flow
4. Mapping an MPEG-2 Decoder Application
5. Conclusion and Future Work
1. Introduction

- CGRA (Coarse-Grained Reconfigurable Architecture)
  - Consist of tens to hundreds of FUs
  - vs FPGA
    - Reduce delay, area, power, configuration time
  - Target applications
    - Telecommunications and multimedia

- System consists of RISC and CGRA
  - CGRA
    - Execute time-critical code segments
    - Exploit parallelism
  - RISC
    - Control intensive segments
    - Complement CGRA

SoC Optimizations and Restructuring
1. Introduction

- Lacks of design methodology and tools for CGRA
  - Could not exploit high parallelism and deliver a software-like design experience

- Previous work
  - A novel modulo scheduling algorithm
    - Mapping a kernel to a family of reconfigurable architecture
  - ADRES
    - Tightly coupled VLIW-CGRA architecture resulting in many advantages over common reconfigurable systems with loosely coupled RISC/reconfigurable matrix
1. Introduction

- In this paper
  - C-based design flow taking full advantage of the scheduling algorithm
  - ADRES features using an MPEG-2 decoder as an example
- The methodology can design an application with efforts comparable with software development while still achieving the high performance expected from reconfigurable architectures.
2. ADRES Architecture Overview

- VLIW + CGRA Architecture
  - Two functional view
    - CGRA
      - Kernel code
      - Exploit high parallelism
    - VLIW
      - Non-kernel code
      - ILP (Instruction Level Parallelism)
  - Shared register file
  - Shared memory access
  - Shared FU
    - Connected to shared register file
2. ADRES Architecture Overview

- Reconfigurable cell
  - FU (Functional Unit) + RF (Register File)
  - Configuration RAM
    - Provide configuration for a RC every cycle
  - Predicate support
  - Connected to other RCs
    - According to topology
    - Able to read data from other RCs

Figure 3. Reconfigurable Cell
2. ADRES Architecture Overview

- ADRES template
  - Many design options
    - Overall topology, supported operation set, resource allocation, timing and even the internal organization of each RC
    - Using XML for configuration of architecture

- Advantage of tightly coupled integration of VLIW and CGRA
  - VLIW instead of RISC
    - Can accelerate non-kernel parts with ILP
  - Shared RF and memory access
    - Reduce both communication overhead and programming complexity
  - Shared resources
    - Reduce costs
3. C-Based Design Flow

- Starts from C description
- Profiling/Partitioning
  - Identifies the candidate loops for mapping on the reconfigurable matrix
  - Based on the execution time and possible speed-up

Figure 4. Design flow for ADRES
3. C-Based Design Flow

- **Source-level transformation**
  - Rewrite the kernel in order to make it pipelineable and to maximize the performance

- **IMPACT frontend**
  - A compiler framework mainly for VLIW
  - Parse the C code
  - Do some analysis and optimizations
  - Emit Lcode IR (Intermediate Representation)

**Figure 4. Design flow for ADRES**
3. C-Based Design Flow

- VLIW code
  - ILP scheduling
  - Register allocation
- CGRA code
  - Data flow analysis and optimization
  - Modulo Scheduling
    - XML-Architecture description and program as input

Figure 4. Design flow for ADRES

SoC Optimizations and Restructuring
3. C-Based Design Flow

- **Code generation**
  - Integration of VLIW code and CGRA code
  - Could be simulated by co-simulator

- **Kernel scheduling**
  - When configuration RAM is not sufficient to contain all kernel codes
  - Divide kernel codes and schedule

*Figure 4. Design flow for ADRES*
3. C-Based Design Flow

- Some key steps need efforts of designer
  - Partitioning and Source-level transformation
  - Most design time are spent
- Partitioning
  - Made in the early phase
  - Highly dependent on designer’s experience and knowledge
  - Profiler only provide some useful information
- Source level parallelism
  - In order to map more loops to the RA
    - In nature C code, we can map only few loops to RA
  - Construct pipelineable loops
    - Using many techniques
    - Function inlining, loop unrolling, ...
3. C-Based Design Flow

- Compilation of loops
  - Focus on one loop at a time
  - Source level transformation for mapping on RA
  - Transformed code is verified on VLIW
  - Compile the code for RA and evaluate II
  - When II is low, design parameters are annotated in setting file and the loop is mapped to RA
  - Otherwise, mapped to VLIW
3. C-Based Design Flow

- Communication between kernel and non-kernel code
  - Handled by compiler automatically with low overhead
  - Analyze variable life and assign them to shared register file
  - Advantage of tightly coupled architecture

Fig. 5. Interfacing between the VLIW processor and the Reconfigurable matrix
4. Mapping an MPEG-2 Decoder Application

- MPEG-2 Decoder
  - Representative multimedia application
  - Requires very high computation power
  - Most execution time is spent on several kernels
  - Good candidate for reconfigurable architectures application
4.1 Mapping to the ADRES Architecture

- 14 loops from the original applications as candidate for pipelining on the RA by profiling the application
  - \textit{form\_comp\_pred1} ~ \textit{form\_comp\_pred8}
  - \textit{idct1} and \textit{idct2}
  - \textit{add\_block1} and \textit{add\_block2}
  - \textit{clear\_block} and \textit{saturate}

- 2 loops from VLD(Variable Length Decoding)
  - Using source-level transformation
  - \textit{non\_intra\_dequant} and \textit{intra\_dequant}

- 16 loops on RA
  - 84.6\% of the total execution time
  - 3.3\% of the total code size
4.1 Mapping to the ADRES Architecture

- Source-level transformation

```c
for(i=0; i++){
    /* VLD, highly ctrl intensive */
    if (code>=16384)
    {
        if (i==0) ...
        else ...
    }
    else if (code>=1024) ...
    else if (code>=512) ...
    ...
    /* dequantize */
    j = scan[id1->alternate_scan][i];
    val = (val * ld1->quantizer_scale * qmat[j]) >> 4;
    bp[j] = sign ? -val : val;
}
```

```c
for(i=0; i++){
    /* VLD */
    /* dequantize replaced */
    run_val[nc] = val;
    run_pos[nc] = i;
}

for(i = 0; i < nc; i++)
    /* dequantize */
    val = run_val[i];
    pos = run_pos[i];
    j = scan[id1->alternate_scan][pos];
    tmp = (val * ld1->quantizer_scale * qmat[j]) >> 4;
}
```

Figure 6. Extract intra_dequant loop

SoC Optimizations and Restructuring
4.1 Mapping to the ADRES Architecture

- Source-level transformation

When the condition is met, the function is ended but it incurs irregular loop.

Function inlining is applied and shortcusts are terminated.

Figure 7. Transformation for idct1 loop

SoC Optimizations and Restructuring
4.1 Mapping to the ADRES Architecture

- Reducing programming complexity and communication overhead
  - Many scalar values are transferred from VLIW to RA using shared register file

```
if ((macroblock_type & MACROBLOCK_MOTION_FORWARD) || (picture_coding_type==P_TYPE))
{
    if (picture_structure==FRAME_PICTURE)
    {
        if ((motion_type==MC_FRAME) || !(macroblock_type & MACROBLOCK_MOTION_FORWARD))
        {
            if (stwtop<2)
                form_prediction(forward_reference_frame,0,current_frame,0,
                Coded_Picture_Width,Coded_Picture_Width<<1,16,8,bx,by,
                PMV[0][0][0],PMV[0][0][1],stwtop);
            ...
        }
    }
    ...
```

Figure 8. A piece of form_predictions
4.2 Mapping Results

- Mapping to an architecture resembling the topology of MorphoSys
  - 64 FUs divided into four tiles
- Entire design took less than one person-week to finish starting from the software implementation
  - Most of the time is spent on partitioning and source-level transformation

<table>
<thead>
<tr>
<th>kernel</th>
<th>no. of ops</th>
<th>II</th>
<th>IPC</th>
<th>stages</th>
<th>sched. time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear_block</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>0.05</td>
</tr>
<tr>
<td>form_comp_pred1</td>
<td>41</td>
<td>2</td>
<td>20.5</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td>form_comp_pred2</td>
<td>13</td>
<td>1</td>
<td>13</td>
<td>6</td>
<td>4.6</td>
</tr>
<tr>
<td>form_comp_pred3</td>
<td>57</td>
<td>2</td>
<td>28.5</td>
<td>10</td>
<td>586</td>
</tr>
<tr>
<td>form_comp_pred4</td>
<td>33</td>
<td>2</td>
<td>16.5</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>form_comp_pred5</td>
<td>54</td>
<td>2</td>
<td>27</td>
<td>8</td>
<td>245</td>
</tr>
<tr>
<td>form_comp_pred6</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>form_comp_pred7</td>
<td>67</td>
<td>3</td>
<td>22.3</td>
<td>6</td>
<td>167</td>
</tr>
<tr>
<td>form_comp_pred8</td>
<td>43</td>
<td>2</td>
<td>21.5</td>
<td>6</td>
<td>132</td>
</tr>
<tr>
<td>saturate</td>
<td>78</td>
<td>3</td>
<td>26</td>
<td>10</td>
<td>1720</td>
</tr>
<tr>
<td>idct1</td>
<td>83</td>
<td>3</td>
<td>27.7</td>
<td>7</td>
<td>363</td>
</tr>
<tr>
<td>idct2</td>
<td>132</td>
<td>4</td>
<td>33</td>
<td>7</td>
<td>459</td>
</tr>
<tr>
<td>add_block1</td>
<td>48</td>
<td>2</td>
<td>24</td>
<td>7</td>
<td>73</td>
</tr>
<tr>
<td>add_block2</td>
<td>44</td>
<td>2</td>
<td>22</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>non_intra_dequant</td>
<td>20</td>
<td>1</td>
<td>20</td>
<td>12</td>
<td>53</td>
</tr>
<tr>
<td>intra_dequant</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>12</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 1. Scheduling results for kernels
4.3 Comparison with VLIW Architecture

<table>
<thead>
<tr>
<th></th>
<th>VLIW (IMPACT)</th>
<th>ADRES</th>
</tr>
</thead>
<tbody>
<tr>
<td>total ops</td>
<td>$2.92 \times 10^9$</td>
<td>$5.31 \times 10^9$</td>
</tr>
<tr>
<td>total cycles</td>
<td>$1.28 \times 10^9$</td>
<td>$4.20 \times 10^8$</td>
</tr>
<tr>
<td>frames/sec</td>
<td>35.2</td>
<td>107.1</td>
</tr>
<tr>
<td>speed-up/kernels</td>
<td></td>
<td>4.84</td>
</tr>
<tr>
<td>speed-up/overall</td>
<td></td>
<td>3.05</td>
</tr>
<tr>
<td>IPC (excl. kernels)</td>
<td></td>
<td>2.71</td>
</tr>
</tbody>
</table>

Table 2. Comparison with VLIW architecture
4.3 Comparison with VLIW Architecture

There is some ILP for the non-kernel code
5. Conclusion and Future Work

- CGRA
  - Have advantages over traditional FPGAs
  - How to map not only computation-intensive kernels but also an entire application
  - Needs for powerful design tool to deliver both high performance and SW-like design experience

- ADRES
  - VLIW+CGRA
  - C based design flow and automotive tools

- Future work
  - Source-level transformation
    - Provide some criteria
  - Kernel scheduling
    - On-going