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

2011-06-01

Presenter : Ingoo Heo

Contents

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

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

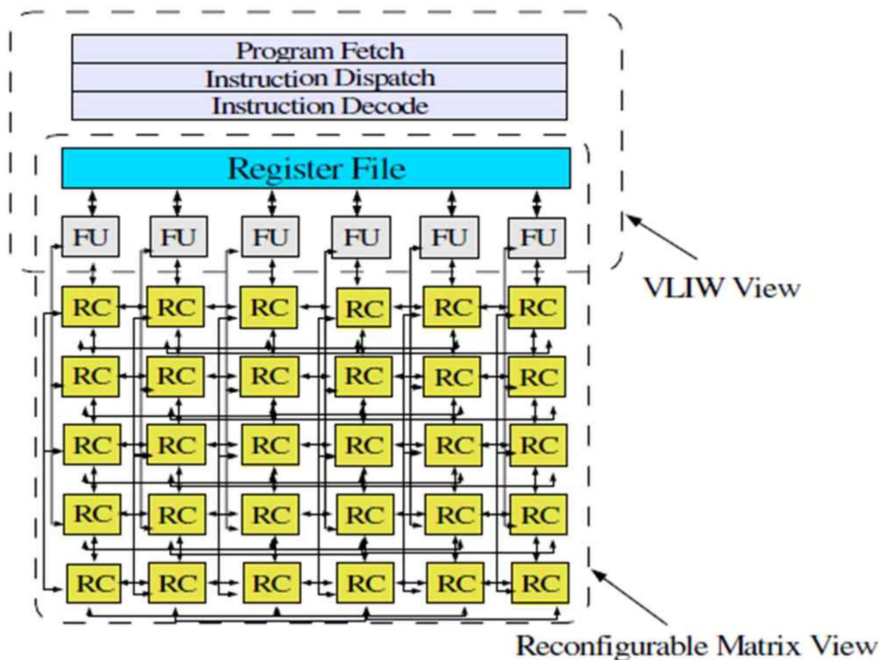

Figure 1. ADRES core

- 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

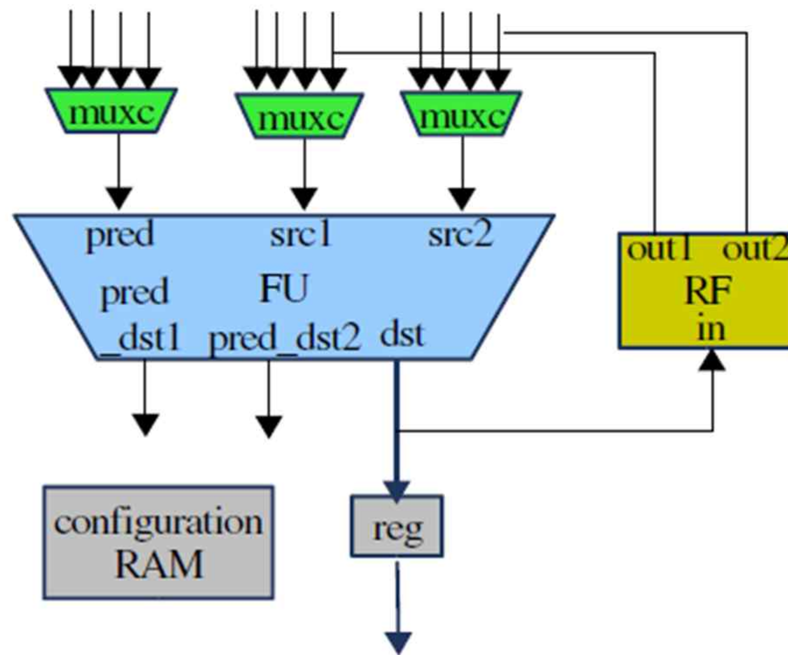

Figure 3. Reconfigurable Cell

- 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

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

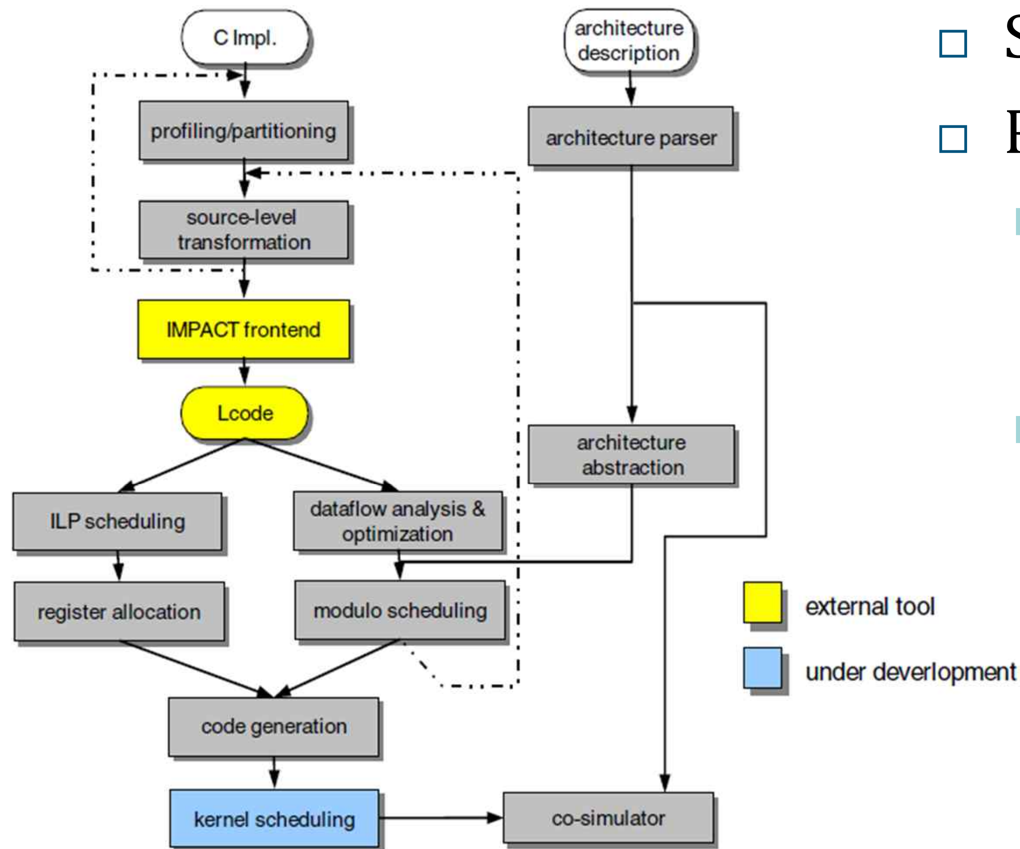

Figure 4. Design flow for ADRES

- 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

3. C-Based Design Flow

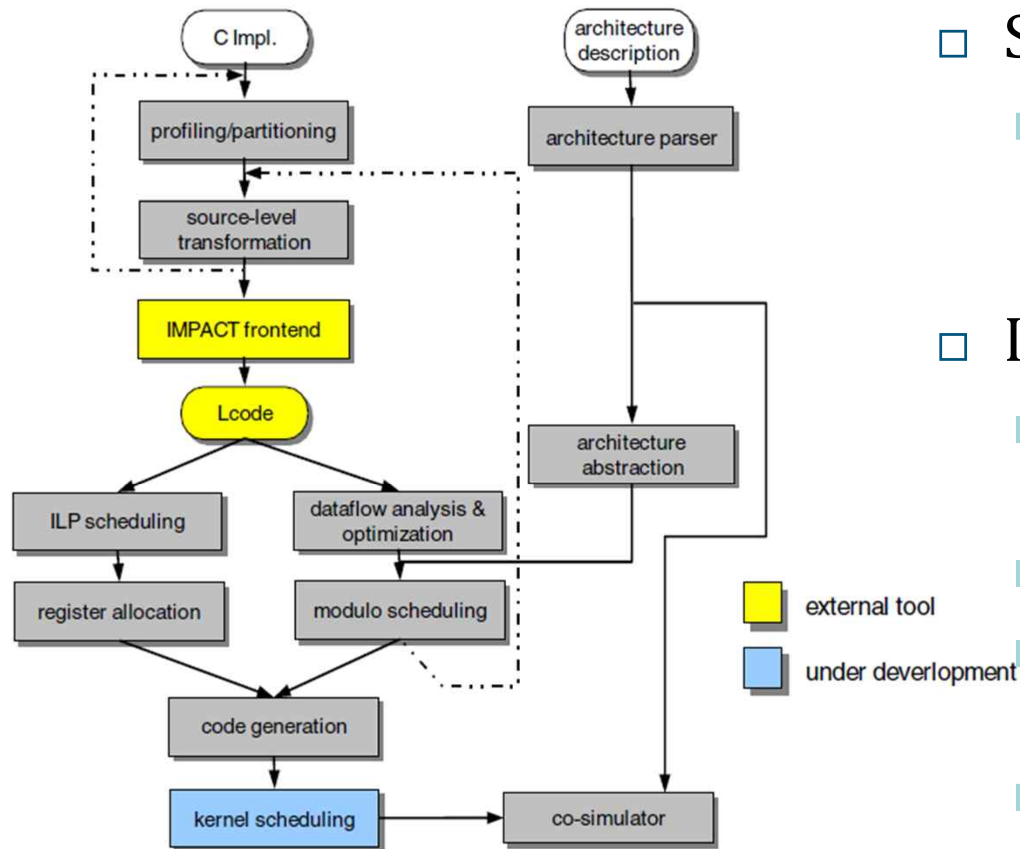

Figure 4. Design flow for ADRES

- 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)

3. C-Based Design Flow

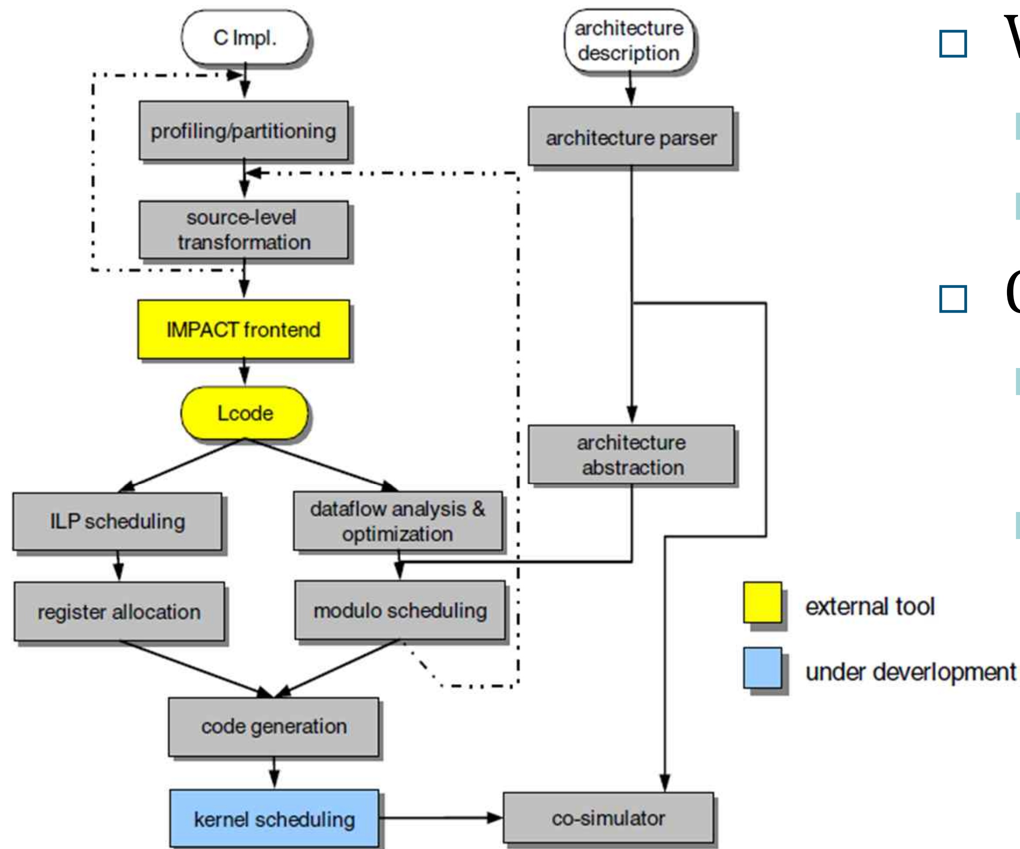

Figure 4. Design flow for ADRES

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

3. C-Based Design Flow

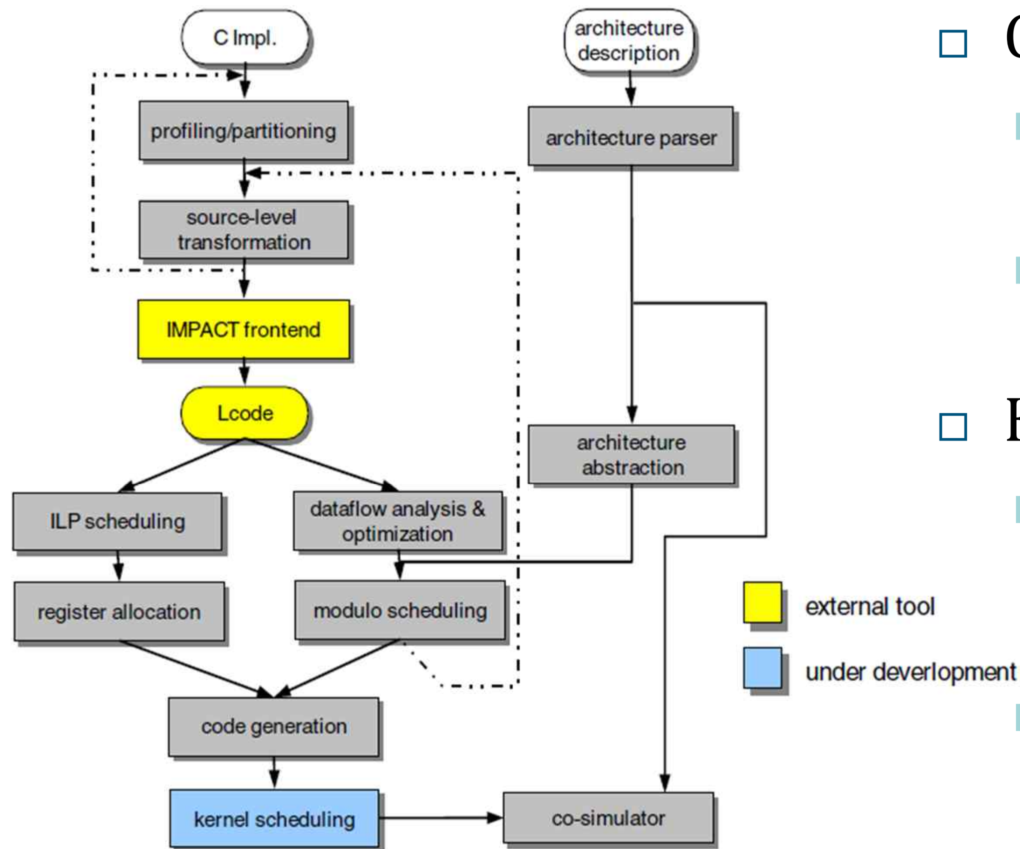

Figure 4. Design flow for ADRES

- 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

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

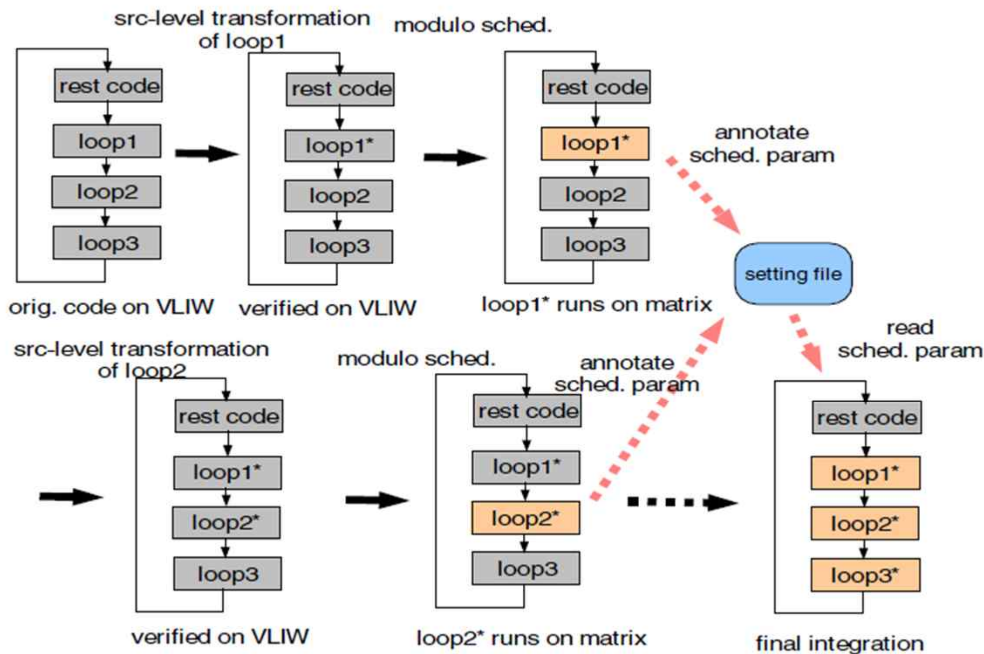

Figure 5. Focus on one loop at a time

- 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

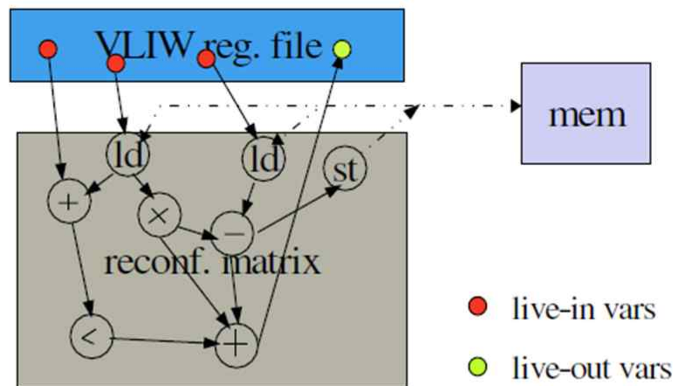

Fig. 5. Interfacing between the VLIW processor and the Reconfigurable matrix

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
  - *form_comp_pred1 ~ form_comp_pred8*
  - *idct1 and idct2*
  - *add_block1 and add_block2*
  - *clear_block and saturate*
- 2 loops from VLD(Variable Length Decoding)
  - Using source-level transformation
  - *non_intra_dequant and 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

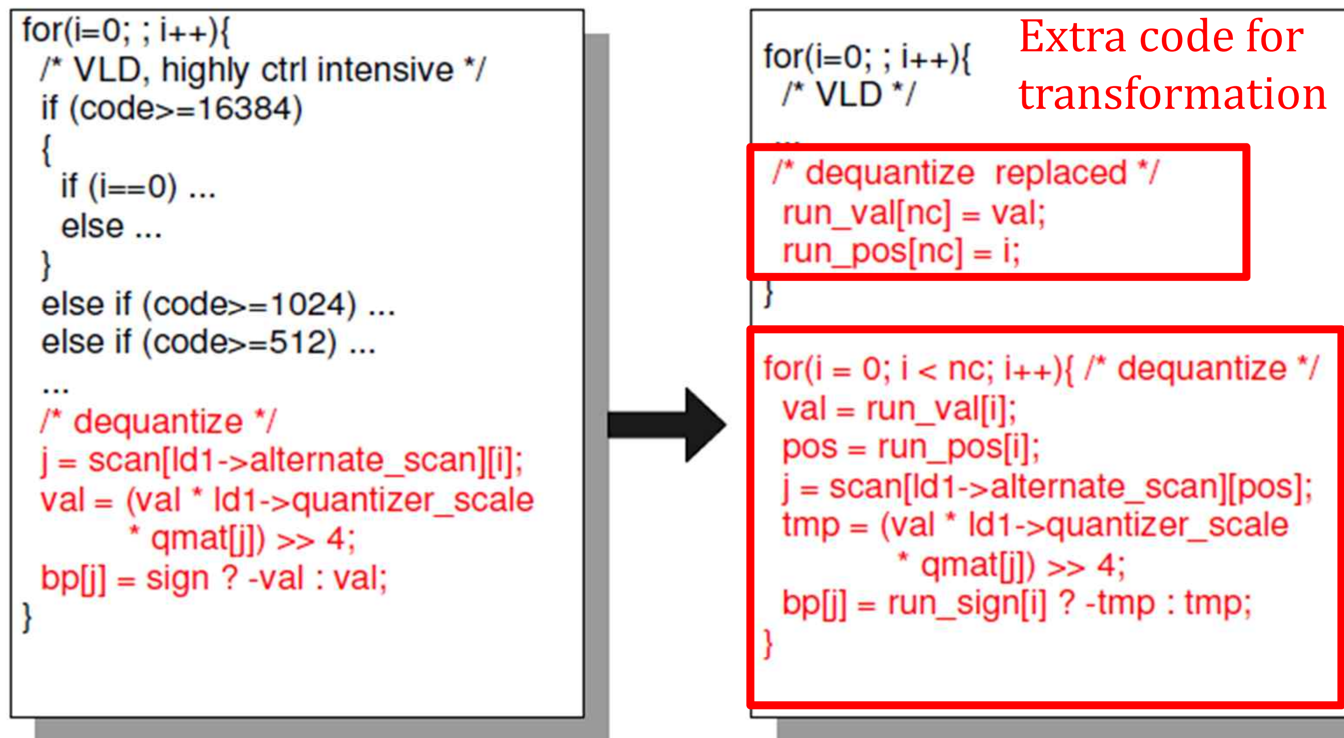

Figure 6. Extract intra_dequant loop

4.1 Mapping to the ADRES Architecture

- Source-level transformation

Function inlining is applied and shorcuts are terminated.

```
for (i=0; i< 8; i++)
   idctrow(block + 8*i);
...
void idctrow(short *blk)
{
if (!((x1 = blk[4]<<11) | (x2 = blk[6]) |...)
      ...)
  {   /*shorcut */  }

  x0 = (blk[0]<<11) + 128
  x8 = W7*(x4+x5);)>>8;
  ...
  blk[6] = (x3-x2)>>8;
  blk[7] = (x7-x1)>>8;
}
```

```
short block[12][64];
...
for (i=0; i<8 * block_count; i++){
  n = i / 8;  /* n is block no. */
  m = i % 8;  /* m is row no. */

  blk = block[n] + 8 * m;

  x0 = (blk[0] << 11) + 128;
  x1 = blk[4] << 11;
  ...
  blk[6] = (x3-x2)>>8;
  blk[7] = (x7-x1)>>8;
}
```

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

Figure 7. Transformation for idct1 loop

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

SOAR RESEARCH GROUP

4.2 Mapping Results

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

Table 1. Scheduling results for kernels

- 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

4.3 Comparison with VLIW Architecture

교육과학기술부
국가지정연구실
National Research Lab

	VLIW(IMPACT)	ADRES
total ops	2.92×10^9	5.31×10^9
total cycles	1.28×10^9	4.20×10^8
frames/sec	35.2	107.1
speed-up/kernels		4.84
speed-up/overall		3.05
IPC(excl. kernels)		2.71

Table 2. Comparison with VLIW architecture

4.3 Comparison with VLIW Architecture

	VLIW(IMPACT)	ADRES
total ops	2.92×10^9	5.31×10^9
total cycles	1.28×10^9	4.20×10^8
frames/sec	35.2	107.1
speed-up/kernels		4.84
speed-up/overall		3.05
IPC(excl. kernels)		2.71

Table 2. 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