Automatic Optimization of Hardware Accelerators for Image Processing

Oliver Reiche\textsuperscript{1}, Konrad Häublein\textsuperscript{2}, Marc Reichenbach\textsuperscript{2}, Frank Hannig\textsuperscript{1}, Jürgen Teich\textsuperscript{1}, and Dietmar Fey\textsuperscript{2}

\textsuperscript{1}Hardware/Software Co-Design, Friedrich-Alexander University Erlangen-Nürnberg (FAU)
\textsuperscript{2}Computer Architecture, Friedrich-Alexander University Erlangen-Nürnberg (FAU)

DATE Friday Workshop HIS, March 13, 2015, Grenoble
Introductory Example: Driver Assistance Systems

Mostly based on image feature detection:

(a) Corner detection  
(b) Optical flow  
(c) Stereo vision

Where to compute features?

ECU [1]  
μC [2]  
CPU [3]  
GPU [4]  
FPGA [5]

Question: Where to run and who implements the algorithm?
Outline

HIPA$^{cc}$ Framework

Optimization Feedback Loop

Case Study

Results

Conclusion
HIPA\textsuperscript{cc} Framework
HIPA\textsuperscript{CC}: The Heterogeneous Image Processing Acceleration Framework

- Domain Knowledge
- Architecture Knowledge
- C++ embedded DSL
- Source-to-Source Compiler Clang/LLVM
- CUDA (GPU)
- OpenCL (x86/GPU)
- C/C++ (x86)
- Renderscript (x86/ARM/GPU)
- Vivado C++ (FPGA)

CUDA/OpenCL/Renderscript Runtime Library

Vivado HLS
HIPA\textsuperscript{cc}: The Heterogeneous Image Processing Acceleration Framework

Domain-specific Extensions

\textbf{IterationSpace} defines ROI of the output image
\textbf{Accessor} input ROI with filtering
\textbf{BoundaryCondition} boundary handling modes
\textbf{Mask} convolution mask

Output image

Crop of output image

Crop of output image with offset
**HIPA\textsuperscript{cc}: The Heterogeneous Image Processing Acceleration Framework**

**Domain-specific Extensions**

- **IterationSpace**: defines ROI of the output image
- **Accessor**: input ROI with filtering
- **BoundaryCondition**: boundary handling mode
- **Mask**: convolution mask

![Image and boundary](image1.png)
![Image crop](image2.png)
![Image crop with offset](image3.png)
![Image offset](image4.png)
HIPAcc: The Heterogeneous Image Processing Acceleration Framework

Domain-specific Extensions

**IterationSpace** defines ROI of the output image

**Accessor** input ROI with filtering

**BoundaryCondition** boundary handling modes

**Mask** convolution mask

Repeat

Clamp

Mirror

Constant
HIPA$^c^c$: The Heterogeneous Image Processing Acceleration Framework

Domain-specific Extensions

**IterationSpace** defines ROI of the output image

**Accessor** input ROI with filtering

**BoundaryCondition** boundary handling modes

**Mask** convolution mask
Example: Laplacian Operator

```cpp
// coefficients for Laplacian operator
const int coef[3][3] = { { 0, 1, 0 },
                         { 1, -4, 1 },
                         { 0, 1, 0 } };

Mask<int> mask(coef);
Image<uchar4> in(width, height);
Image<uchar4> out(width, height);

// load image data
in = image_bits;

// reading from in with mirroring as boundary condition
BoundaryCondition<uchar4> bound(in, mask, BOUNDARY_MIRROR);
Accessor<uchar4> acc(bound);

// output image
IterationSpace<uchar4> iter(out);

// define kernel
Laplacian filter(iter, acc, mask);

// execute kernel
filter.execute();
```
Example: Laplacian Operator Kernel

```cpp
class Laplacian : public Kernel<uchar4> {
    private:
        Accessor<uchar4> &input;
        Mask<int> &mask;

    public:
        Laplacian(IterationSpace<uchar4> &iter,
            Accessor<uchar4> &input, Mask<int> &mask)
            : Kernel(iter), input(input), mask(mask) {
            addAccessor(&input);
        }

        void kernel() {
            int4 sum = convolve(mask, HipaccSUM, [&] () -> int4 {
                return mask() * convert_int4(input(mask));
            });
            sum = max(sum, 0);
            sum = min(sum, 255);
            output() = convert_uchar4(sum);
        }
};
```

Convolution call
Generating the Streaming Pipeline

Trace host code and translate it to *internal representation*:

- model as combination of *processes* and *spaces*
- create unique stream objects for each *space*
- identify memory reuse
- insert *copy processes*
- build dependency graph
- traverse in depth-first search starting from output *spaces*
Optimization Feedback Loop
Workflow for the Optimization Feedback Loop

- Golden Reference
- RTL Simulation
- Vivado HLS
- FPGA

C++
Embedded DSL

Domain/Architecture Knowledge

Optimization Loop

HDL
Report
Optimization Targets

Three Parameters:

- **Initiation Interval (II)**: # of clock cycles before next pixel is processed
- **Clock Frequency**: # of clock cycles per second in MHz
- **Resource usage**: 4-tuple for LUTs, FFs, BRAMs, and DSPs in %

Two are set as constraints, the third is variable.

![Diagram](https://via.placeholder.com/150)

*Figure: Three parameters II, clock, and resource usage with trade-off regarding throughput.*
Bisectioning

1: function Optimize(target, constraints)
2:     low ← defaultLow(target)
3:     high ← low
4: repeat
5:     high ← 2 × high
6:     generateCode(target, high, constraints)
7:     runSynthesis()
8:     until constraintsMet(constraints)
9: while low ≠ high do
10:    current ← \frac{low + high}{2}
11:    generateCode(target, current, constraints)
12:    runSynthesis()
13:    if constraintsMet(constraints) then
14:        high ← current
15:    else
16:        high ← low
17:    end if
18: end while
19: end function

▷ Phase 1: Find upper bound
▷ Phase 2: Search optimum
Case Study
**Stereo Vision Algorithm**

**Block Matching Variants**

- **SAD** Sum of Absolute Difference
- **Census** Census-transformed signatures

**Figure:** Images taken from Middlebury 2003 stereo datasets [6]
The Bit-Count Problem

Use Brain Kernighan algorithm to compute Hamming distance.

Naïve Approach

Use standard C++:

```c++
int count = 0;
while (val) {
    val &= val - 1;
    ++count;
}
```

DSL Approach

Use DSL constructs:

```c++
int count = 0;
iterate(sizeof(val)*8, [&] () {
    if (!val) break_iterate();
    val &= val - 1;
    ++count;
});
```

Generated C++-code containing Vivado HLS pragmas:

```c++
int count = 0;
for (int i = 0; i < sizeof(val)*8; ++i) {
    #pragma HLS unroll
    #pragma HLS loop_tripcount min=0 max=sizeof(val)*8
    if (!val) break;
    val &= val - 1;
    ++count;
}
```
Results
Experimental Setup

Algorithm
Two block matching variants for stereo vision:

(a) Input image  (b) SAD result  (c) Census result

Evaluation Environment

Zynq 7100  Kintex-7 FPGA
Image size  $450 \times 375$ pixels
Results: Automatic Variant Exploration

Constraints

- fixed $ll = 1$
- fixed resource usage < 6%

Parameter

- variable clock frequency
- directly affects achievable throughput
Results: Performance HIPA$^{cc}$ vs. Handwritten

Performance comparison of the block matching variants $SAD$ and $Census$:
Results: Resource Usage HIPA$^{cc}$ vs. Handwritten

Resource usage comparison of the block matching variants SAD and Census:

![Graph showing resource usage comparison between SAD, Census, Handwritten HIPA$^{cc}$ with LUT, FF, BRAM, DSP resources and clock frequency.]
Conclusion
Conclusion

Advantages

Design decisions can be postponed thanks to DSL-based approach:

- 3P: Productivity, Performance, Portability
- thorough variant exploration near given constraints

Case Study

Results from stereo vision case study show:

(-) DSL may introduce restrictions to overcome
(-) HLS leaves room for improvement
(+ ) fast prototyping on other architectures
(+ ) less error prone, stick to DSL constructs

Non-architecture experts can achieve good results.
Thanks for listening. Any questions?

Title  Automatic Optimization of Hardware Accelerators for Image Processing
Contact  oliver.reiche@cs.fau.de

http://github.com/hipacc
References I


Backup Slides
Detailed Results

Table: Synthesis results of the HIPA\textsuperscript{cc}-generated block matching algorithms SAD and Census difference for an image of size 450 \times 375 on a Zynq 7100.

<table>
<thead>
<tr>
<th></th>
<th>II</th>
<th>LAT</th>
<th>BRAM</th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
<th>F[MHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAD</td>
<td>1</td>
<td>181,797</td>
<td>4</td>
<td>2</td>
<td>140,228</td>
<td>66,185</td>
<td>182.38</td>
</tr>
<tr>
<td>Census</td>
<td>1</td>
<td>180,090</td>
<td>4</td>
<td>0</td>
<td>54,016</td>
<td>23,144</td>
<td>289.52</td>
</tr>
</tbody>
</table>

Table: Synthesis results of the handwritten block matching algorithms SAD and Census difference for an image of size 450 \times 375 on a Zynq 7100.

<table>
<thead>
<tr>
<th></th>
<th>II</th>
<th>LAT</th>
<th>BRAM</th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
<th>F[MHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAD</td>
<td>1</td>
<td>170,565</td>
<td>4</td>
<td>0</td>
<td>29,288</td>
<td>37,940</td>
<td>271.59</td>
</tr>
<tr>
<td>Census</td>
<td>1</td>
<td>170,561</td>
<td>4</td>
<td>0</td>
<td>9,978</td>
<td>19,247</td>
<td>319.18</td>
</tr>
</tbody>
</table>