Automatic Optimization of Hardware Accelerators for Image Processing

Oliver Reiche[1], Konrad Häublein[2], Marc Reichenbach[2], Frank Hannig[1],
Jürgen Teich[1], and Dietmar Fey[2]
[1]Hardware/Software Co-Design, Friedrich-Alexander University Erlangen-Nürnberg (FAU)
[2]Computer Architecture, Friedrich-Alexander University Erlangen-Nürnberg (FAU)
DATE Friday Workshop HIS, March 13, 2015, Grenoble

Introductory Example: Driver Assistance Systems

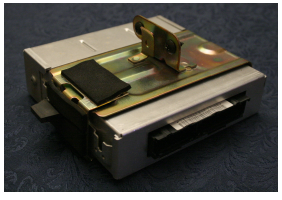
Mostly based on image feature detection:

(a) Corner detection

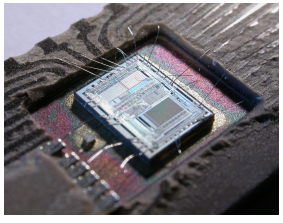
(b) Optical flow

(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

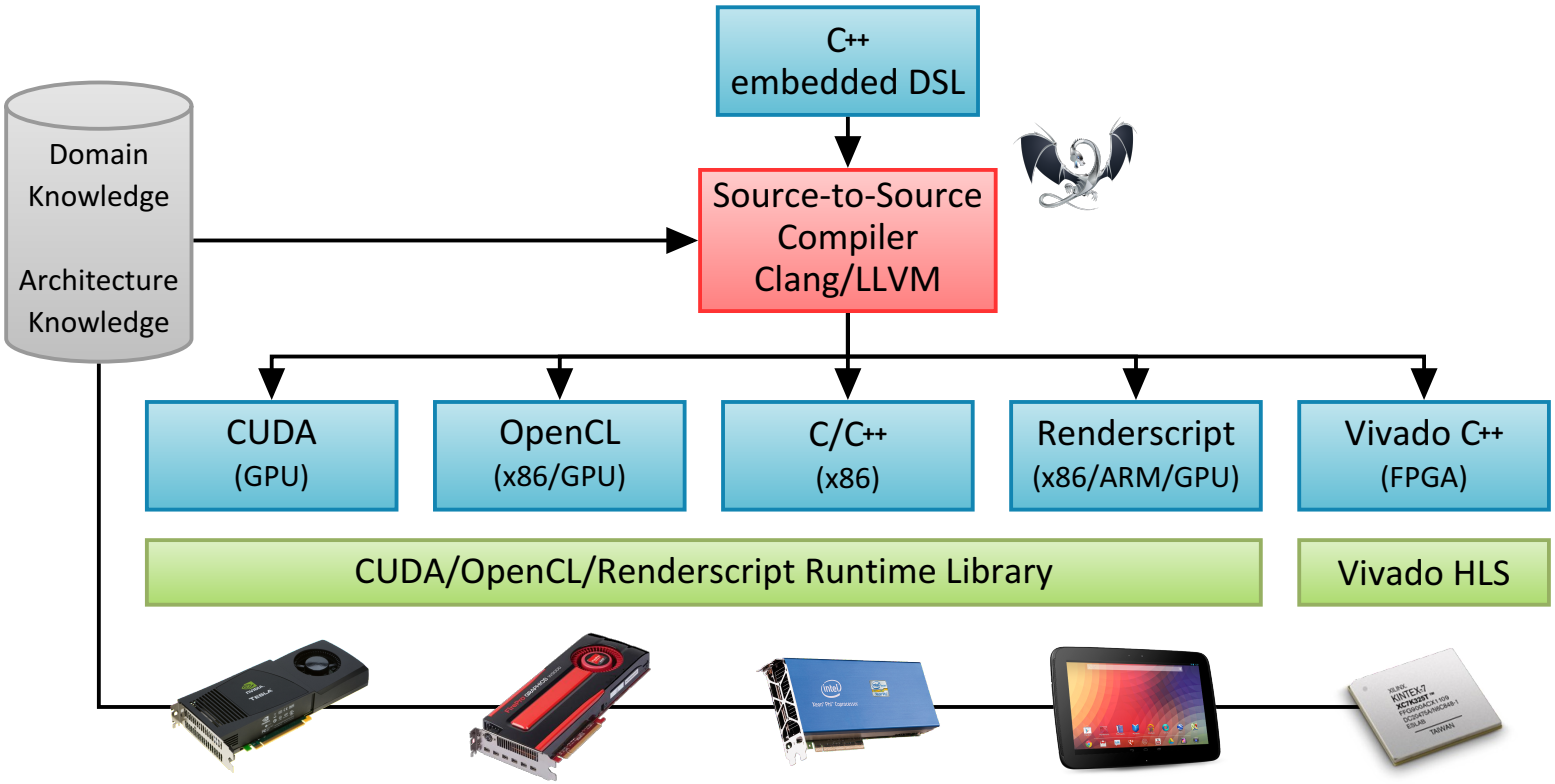
Case Study

Results

Conclusion

HIPA^{cc} Framework

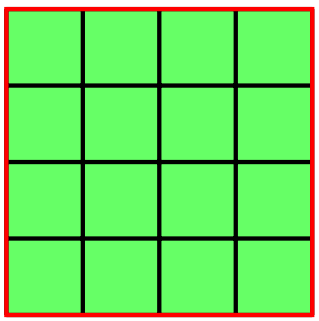
HIPA^{cc}: The Heterogeneous Image Processing Acceleration Framework



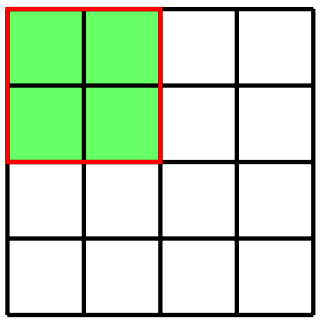
HIPA^{cc}: The Heterogeneous Image Processing Acceleration Framework

Domain-specific Extensions

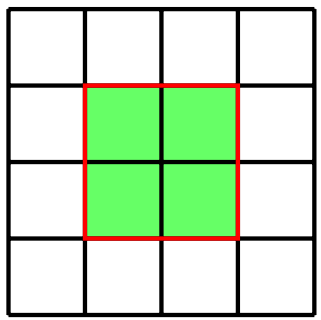
IterationSpace	**defines ROI of the output image**
Accessor	input ROI with filtering
BoundaryCondition	boundary handling modes
Mask	convolution mask



Output image



Crop of output image



Crop of output image with offset

HIPA^{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

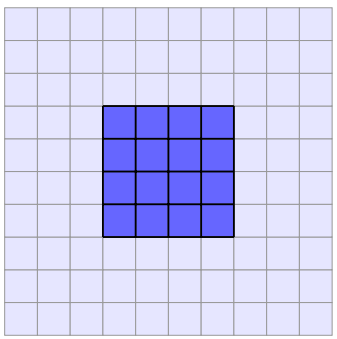


Image and boundary

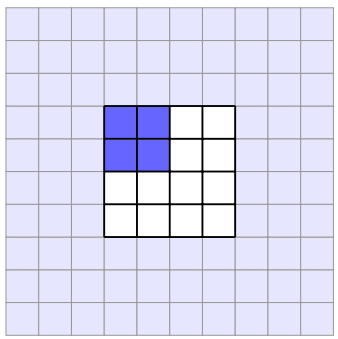


Image crop

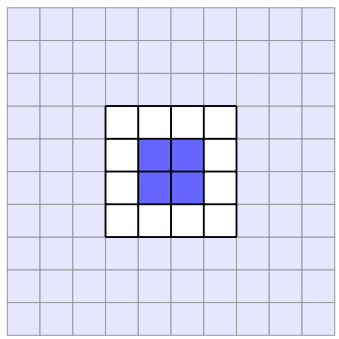


Image crop with offset

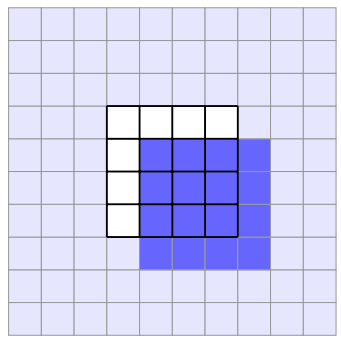


Image offset

HIPA^{cc}: The Heterogeneous Image Processing Acceleration Framework

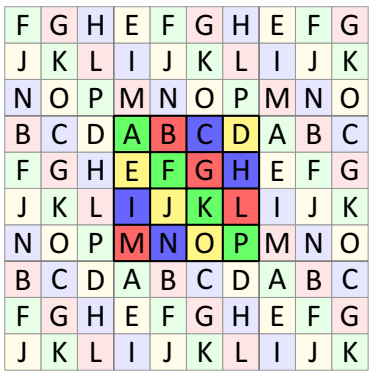
Domain-specific Extensions

IterationSpace defines ROI of the output image

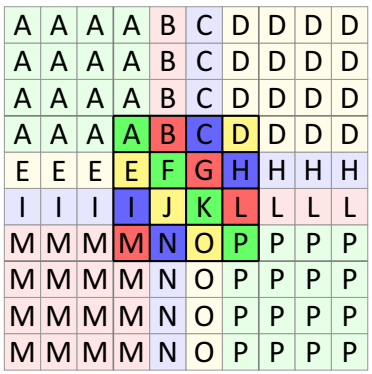
Accessor input ROI with filtering

BoundaryCondition **boundary handling modes**

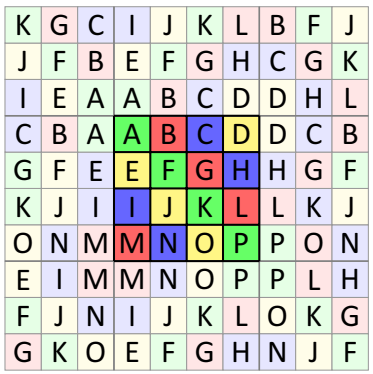
Mask convolution mask



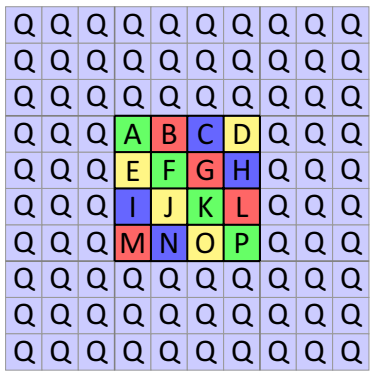
Repeat



Clamp



Mirror

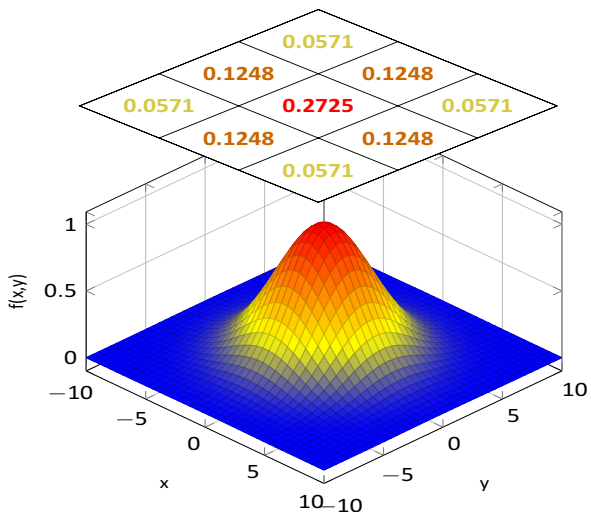


Constant

HIPA^{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 modes
Mask	**convolution mask**



Example: Laplacian Operator

```
1  // coefficients for Laplacian operator
2  const int coef[3][3] = { {  0,  1,  0 },
3                           {  1, -4,  1 },
4                           {  0,  1,  0 } };
5
6  Mask<int> mask(coef);
7  Image<uchar4> in(width, height);
8  Image<uchar4> out(width, height);
9
10 // load image data
11 in = image_bits;
12
13 // reading from in with mirroring as boundary condition
14 BoundaryCondition<uchar4> bound(in, mask, BOUNDARY_MIRROR);
15 Accessor<uchar4> acc(bound);
16
17 // output image
18 IterationSpace<uchar4> iter(out);
19
20 // define kernel
21 Laplacian filter(iter, acc, mask);
22
23 // execute kernel
24 filter.execute();
```

Example: Laplacian Operator Kernel

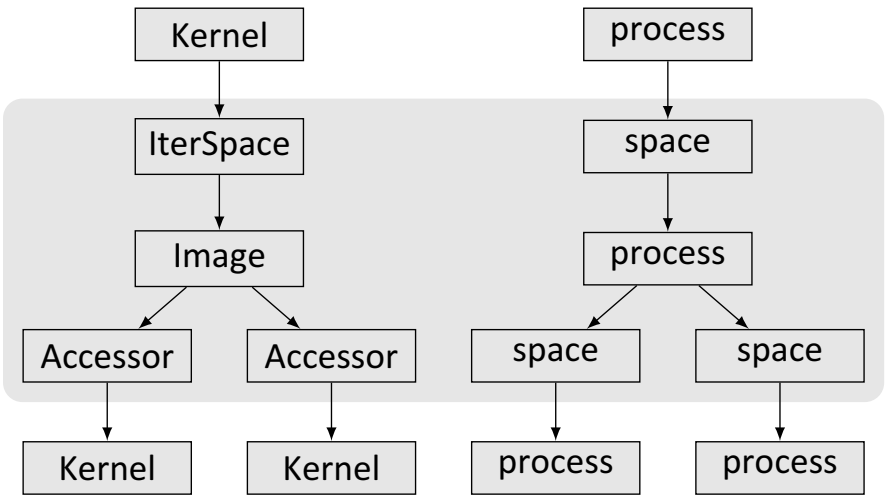
```
1  class Laplacian : public Kernel<uchar4> {
2    private:
3      Accessor<uchar4> &input;
4      Mask<int> &mask;
5
6    public:
7      Laplacian(IterationSpace<uchar4> &iter,
8                Accessor<uchar4> &input, Mask<int> &mask)
9          : Kernel(iter), input(input), mask(mask) {
10       addAccessor(&input);
11     }
12
13     void kernel() {
14       int4 sum = convolve(mask, HipaccSUM, [&] () -> int4 {
15                    return mask() * convert_int4(input(mask));
16                  });
17       sum = max(sum, 0);
18       sum = min(sum, 255);
19       output() = convert_uchar4(sum);
20     }
21 };
```

Convolution call

Generating the Streaming Pipeline

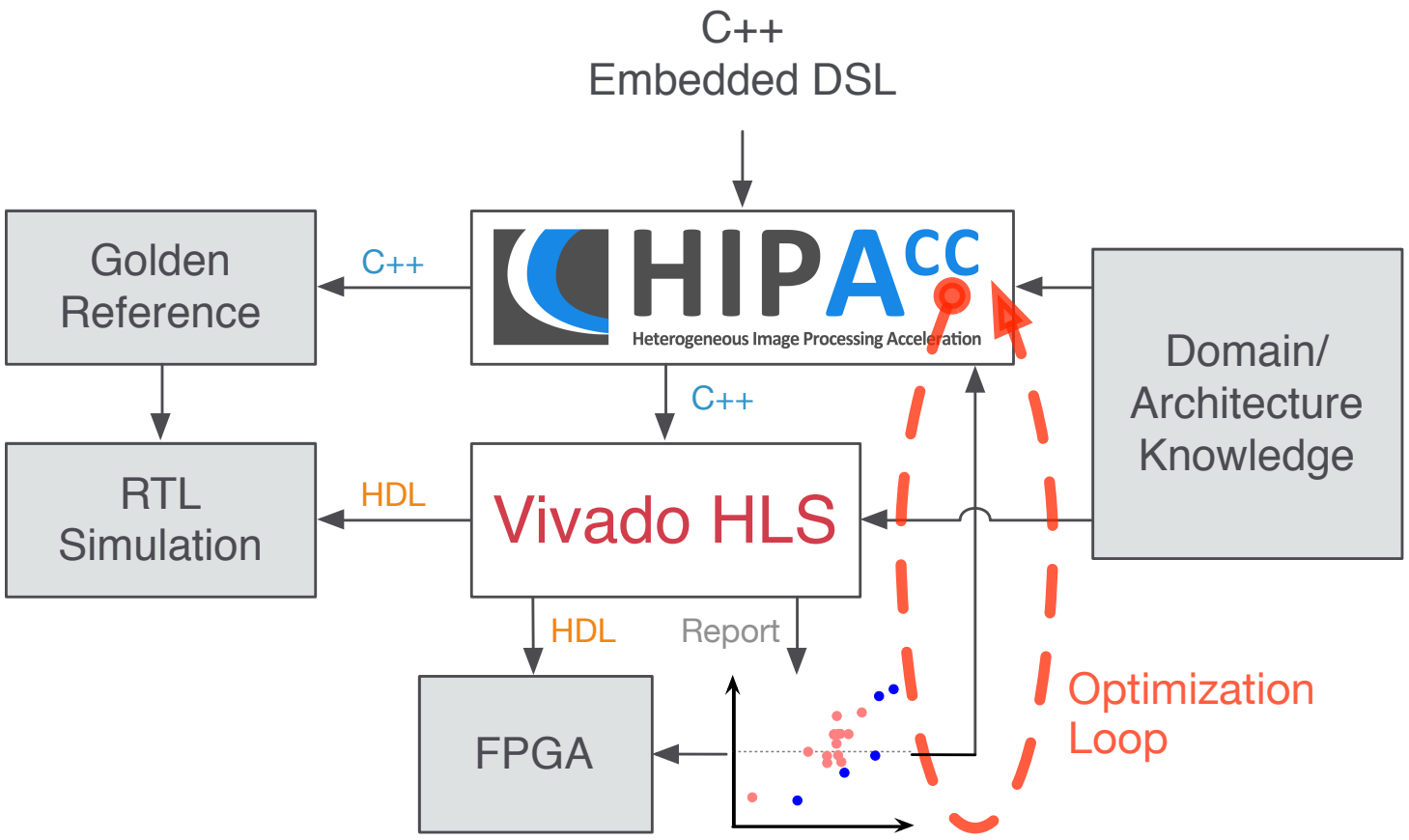
Trace host code and translate it to *internal representation*:

- 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



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.

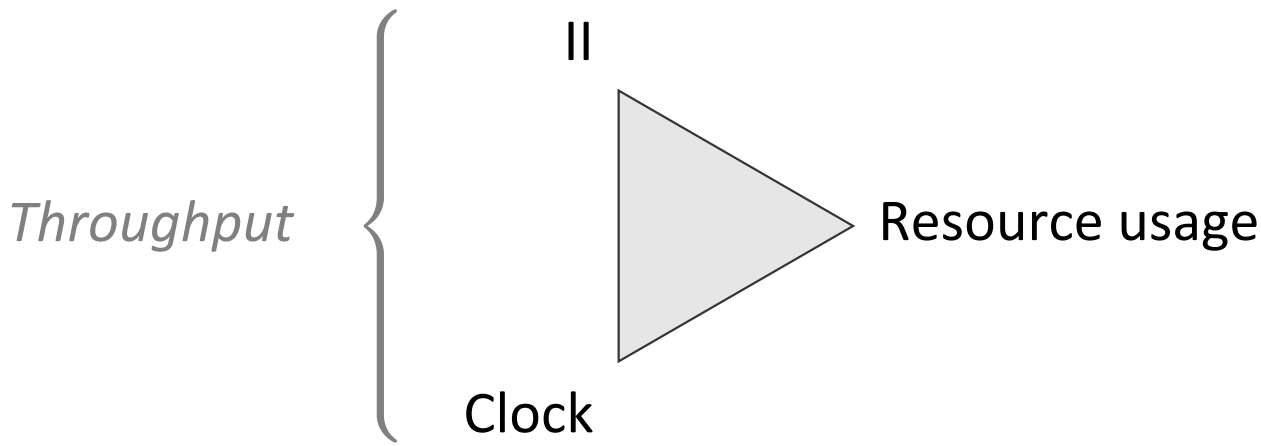


Figure: Three parameters *II*, *clock*, and *resource usage* with trade-off regarding *throughput*.

Bisectioning

```
 1: function Optimize(target, constraints)
 2:     low ← defaultLow(target)
 3:     high ← low
 4:     repeat                                   ▷ Phase 1: Find upper bound
 5:         high ← 2×high
 6:         generateCode(target, high, constraints)
 7:         runSynthesis( )
 8:     until constraintsMet(constraints)
 9:     while low ≠ high do                      ▷ Phase 2: Search optimum
10:         current ← (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
```

Case Study

Stereo Vision Algorithm

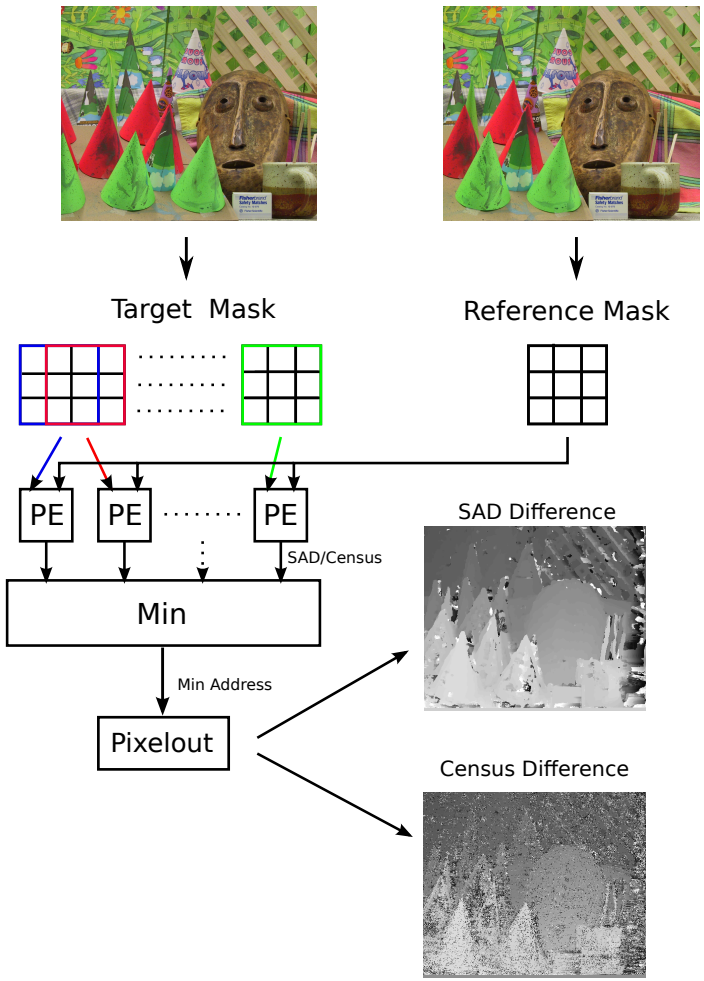
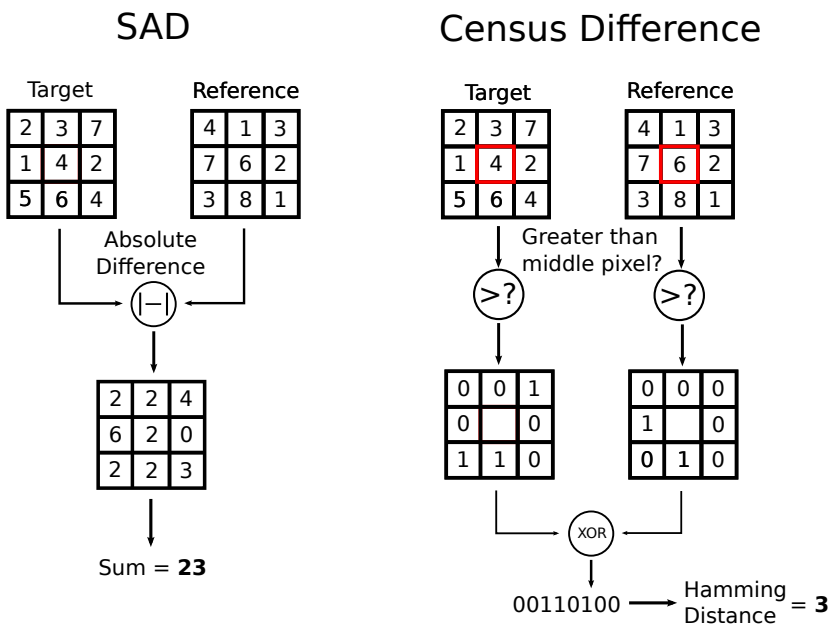


Figure: Images taken from Middlebury 2003 stereo datasets [6]

Block Matching Variants

SAD Sum of Absolute Difference

Census Census-transformed signatures



The Bit-Count Problem

Use Brain Kernighan algorithm to compute Hamming distance.

Naïve Approach

Use standard C++:

```
1 int count = 0;
2 while (val) {
3   val &= val - 1;
4   ++count;
5 }
```

DSL Approach

Use DSL constructs:

```
1 int count = 0;
2 iterate(sizeof(val)*8, [&] () {
3   if (!val) break_iterate();
4   val &= val - 1;
5   ++count;
6 });
```

Generated C++-code containing Vivado HLS pragmas:

```
1 int count = 0;
2 for (int i = 0; i < sizeof(val)*8; ++i) {
3 #pragma HLS unroll
4 #pragma HLS loop_tripcount min=0 max=sizeof(val)*8
5   if (!val) break;
6   val &= val - 1;
7   ++count;
8 }
```

Results

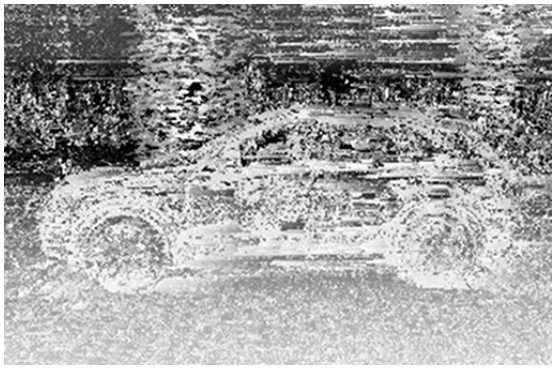
Experimental Setup

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 450×375 pixels

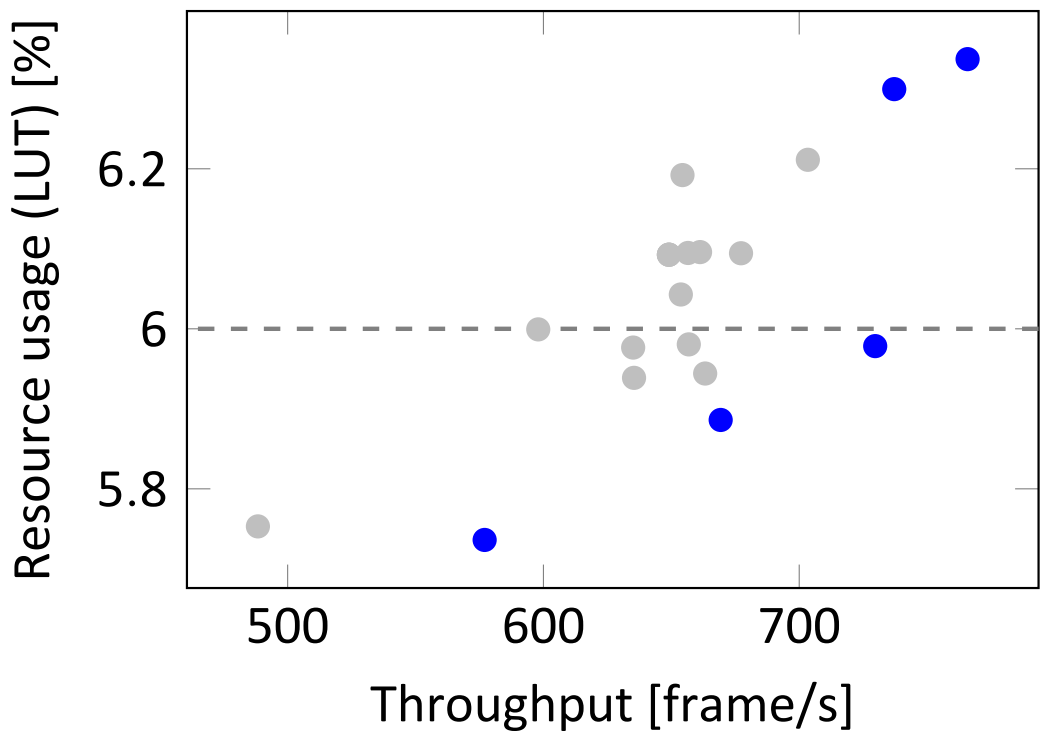
Results: Automatic Variant Exploration

Constraints

- fixed II $= 1$
- fixed resource usage $< 6\,\%$

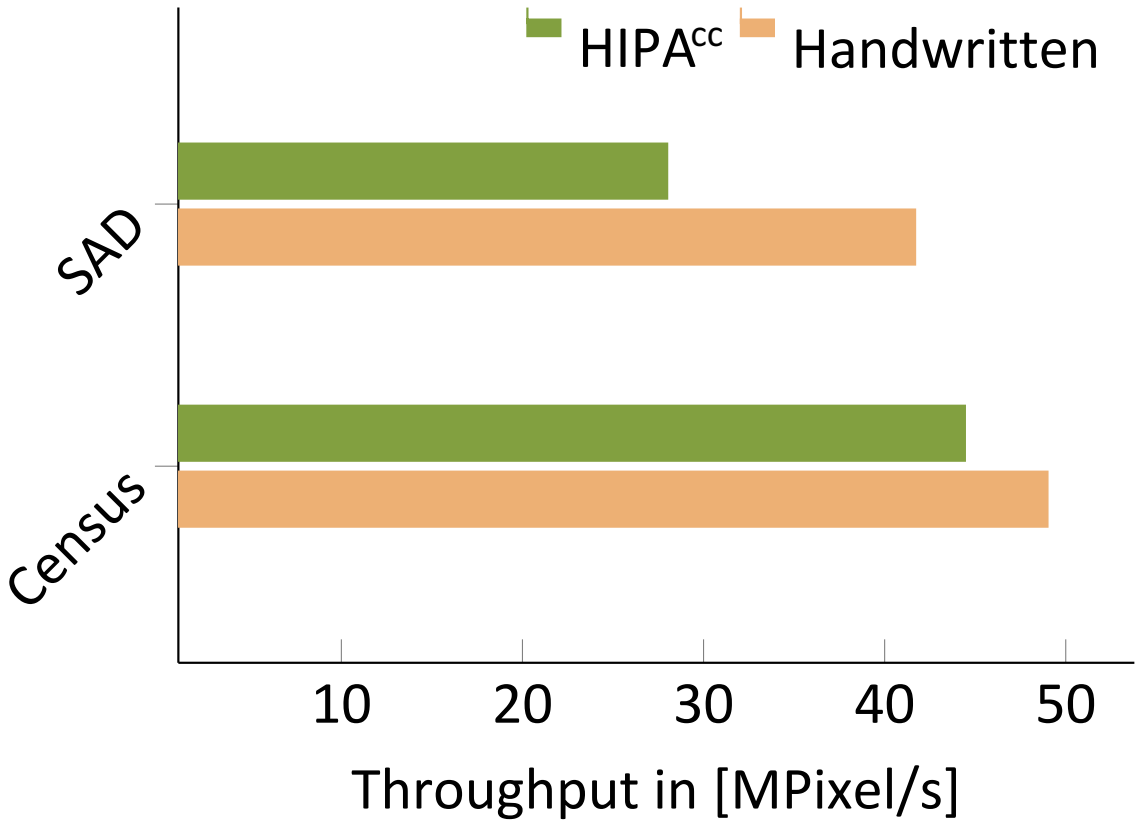
Parameter

- variable clock frequency
- directly affects achievable throughput



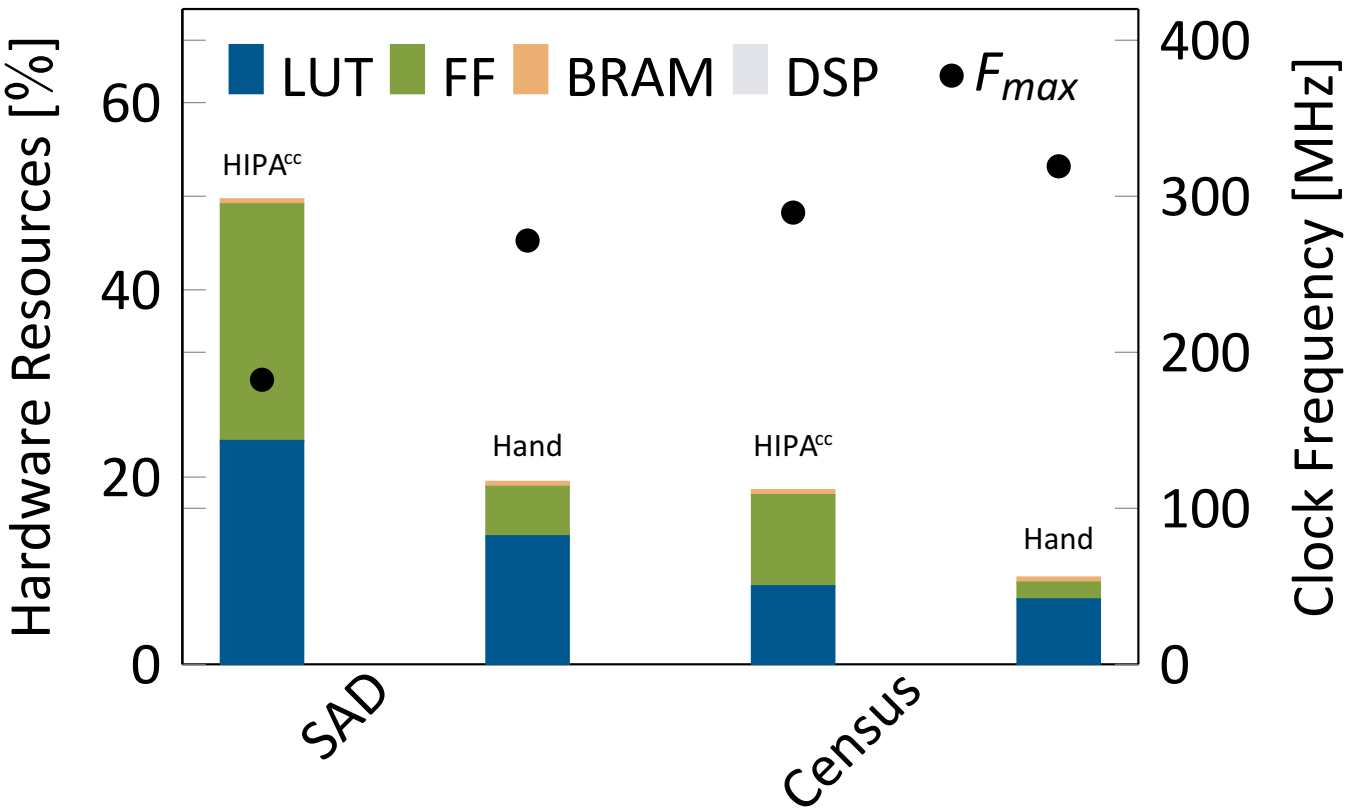
Results: Performance HIPA^{cc} vs. Handwritten

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*:



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?

http://github.com/hipacc

Title Automatic Optimization of Hardware Accelerators for Image Processing

Contact oliver.reiche@cs.fau.de

References

References I

[1] Ildar Sagdejev. *2008-04-17 ECU*. url: http://en.wikipedia.org/wiki/Electronic_control_unit#mediaviewer/File:2008-04-17_ECU.jpg.

[2] Ioan Sameli. *File:Intel 8742 153056995.jpg*. url: http://en.wikipedia.org/wiki/File:Intel_8742_153056995.jpg.

[3] Henriok. *LPC18A1-and-A7*. url: http://commons.wikimedia.org/wiki/Category:ARM_Cortex-M#mediaviewer/File:LPC18A1-and-A7.jpg.

[4] Poeggi. *NVIDIA T20 and T30 chips*. url: http://en.wikipedia.org/wiki/Tegra#mediaviewer/File:NVIDIA_T20_and_T30_chips.jpg.

[5] Dake. *Fpga xilinx spartan*. url: http://en.wikipedia.org/wiki/Field-programmable_gate_array#mediaviewer/File:Fpga_xilinx_spartan.jpg.

References II

[6] Daniel Scharstein and Richard Szeliski. “High-Accuracy Stereo Depth Maps Using Structured Light”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. (Madison, WI, USA). Vol. I. IEEE, 2003, pp. 195–202. doi: 10.1109/CVPR.2003.1211354.

Backup Slides

Detailed Results

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

	II	LAT	BRAM	DSP	FF	LUT	F[MHz]
SAD	1	181,797	4	2	140,228	66,185	182.38
Census	1	180,090	4	0	54,016	23,144	289.52

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.

	II	LAT	BRAM	DSP	FF	LUT	F[MHz]
SAD	1	170,565	4	0	29,288	37,940	271.59
Census	1	170,561	4	0	9,978	19,247	319.18