UNIVERSITÄT DES SAARLANDES

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

A Journey into DSL Design using Generative Programming:

FPGA Mapping of Image Border Handling through Refinement

M. Akif Özkan [1],
Arsene Perard-Gayot [2] Richard Membarth [2,3], Philipp Slusallek [2,3], Frank Hannig [1], and Jürgen Teich [1]

[1] Friedrich-Alexander University Erlangen-Nürnberg,
[2] Saarland University (UdS), Germany
[3] German Research Center for Artificial Intelligence (DFKI), Germany

FPGAs for Software Programmers (co-located with FPL), August 31, 2018, Dublin

Outline

Motivation

Design of a DSL and AnyDSL

Border Handling

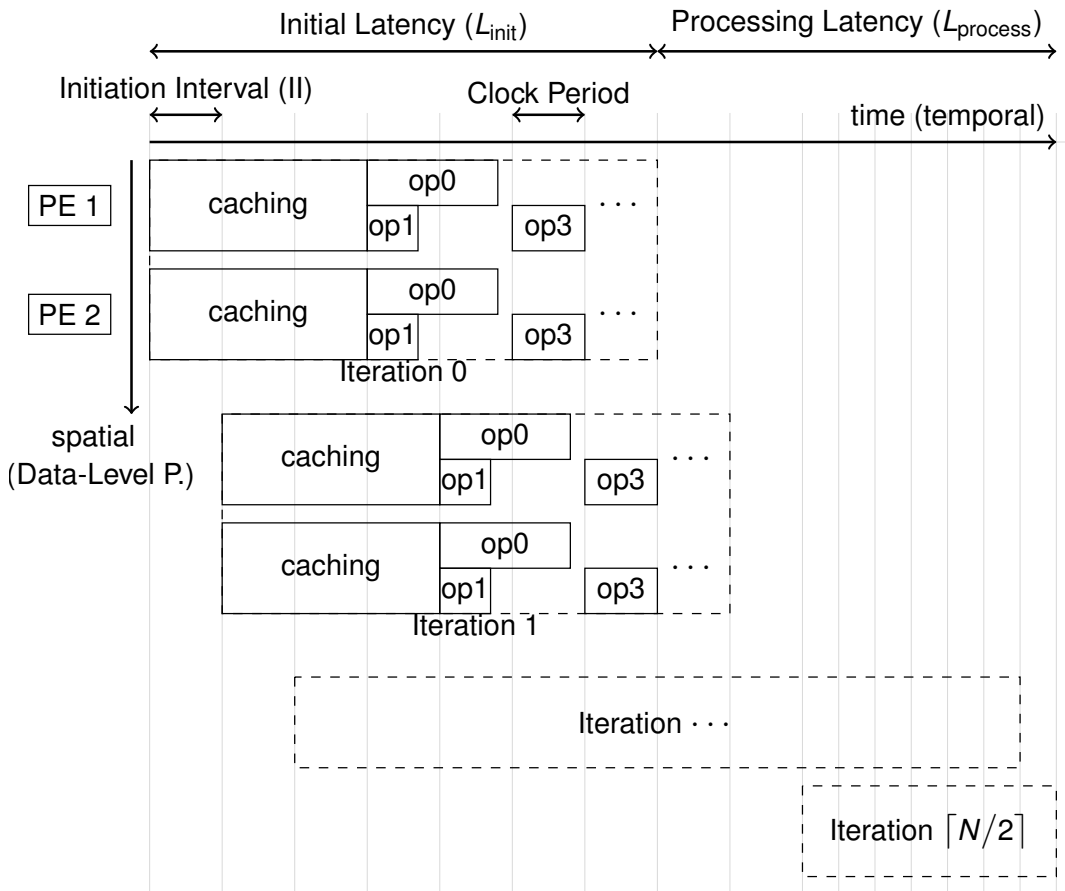
Evaluation and Results

Motivation

FPGAs . . .

. . . can improve the throughput and reduce the energy consumption by tailoring the implementation to the application

- Exploit data locality
  - Spatial locality: Data-Level Parallelism
  - Temporal locality: Pipelining (structural, loop (functional))
- Optimize data-path
  - Constant propagation
  - Constant Multipliers
  - ...
- Device-specific mapping
  - Map directly to the LUTs
  - ...
- ...

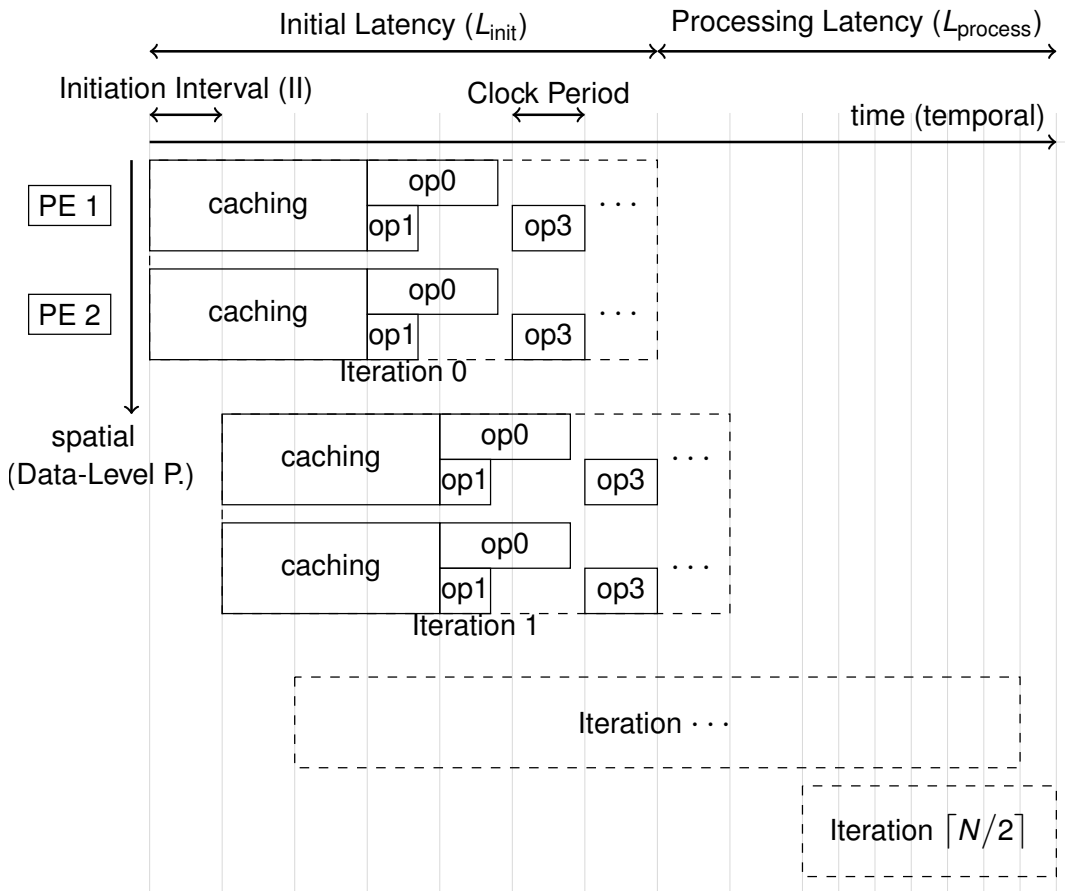


FPGAs . . .

. . . can improve the throughput and reduce the energy consumption by tailoring the implementation to the application

- Exploit data locality
  - Spatial locality: Data-Level Parallelism
  - Temporal locality: Pipelining (structural) loop (functional))
- Optimize data-path
  - Constant propagation
  - Constant Multipliers
  - ...
- Device-specific mapping
  - Map directly to the LUTs
  - ...
- ...

Code restructure is needed!



This is time-consuming and error-prone, even for experts

High-Level Synthesis . . .

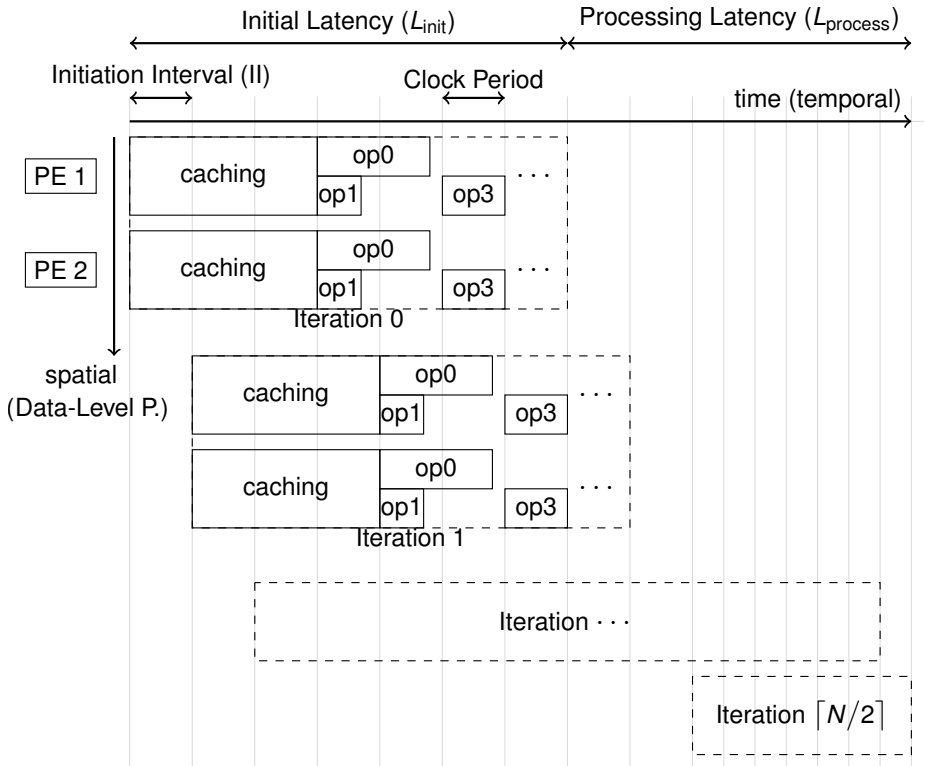
. . . increases the abstraction level, thus eliminates the low-level issues:

- clock-level timing
- register allocation
- structural pipelining
- ...

. . . still requires FPGA expertise for performance, where

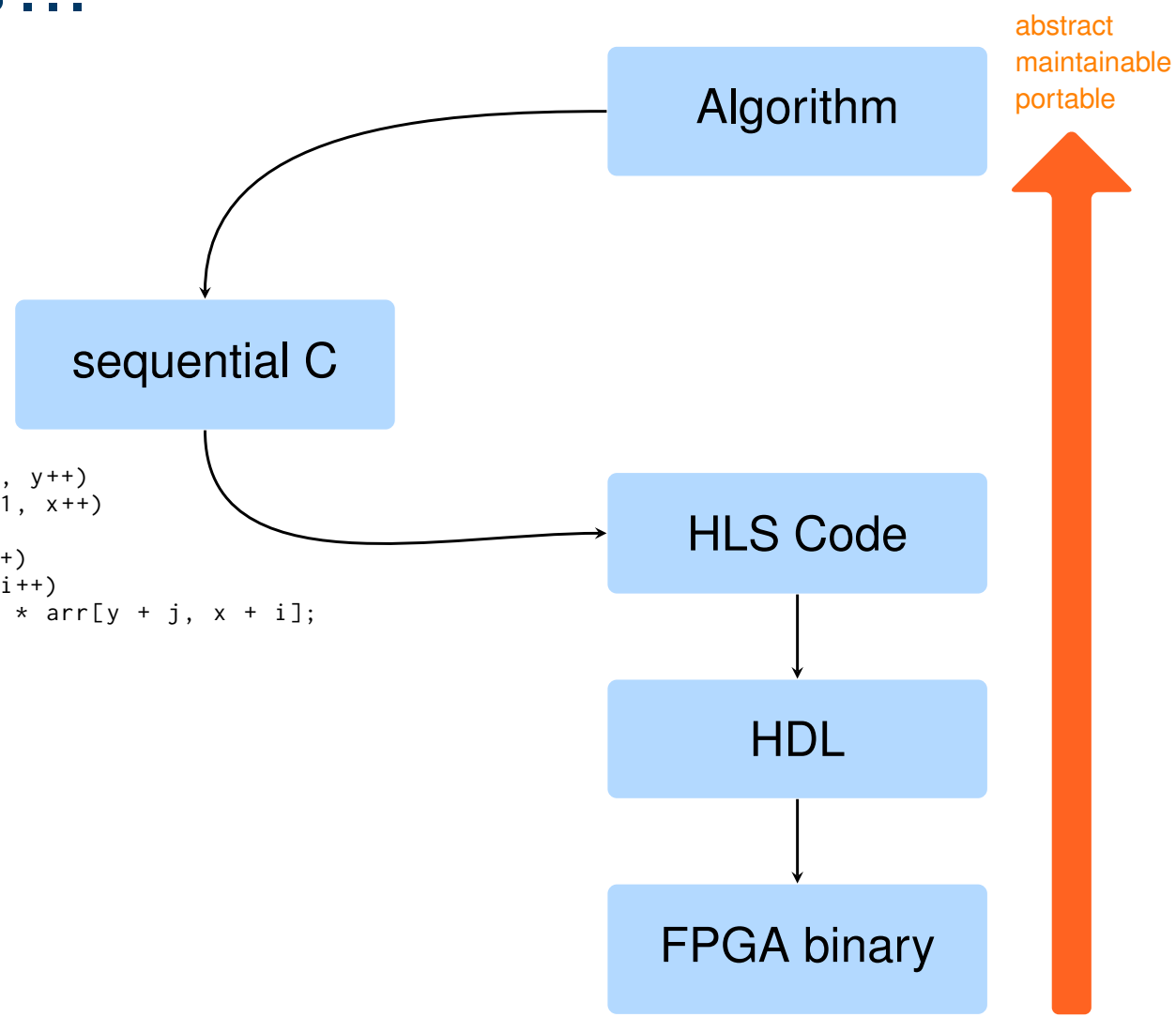
- application-specific caching
- functional (software) pipelining
- ...

should be described explicitly

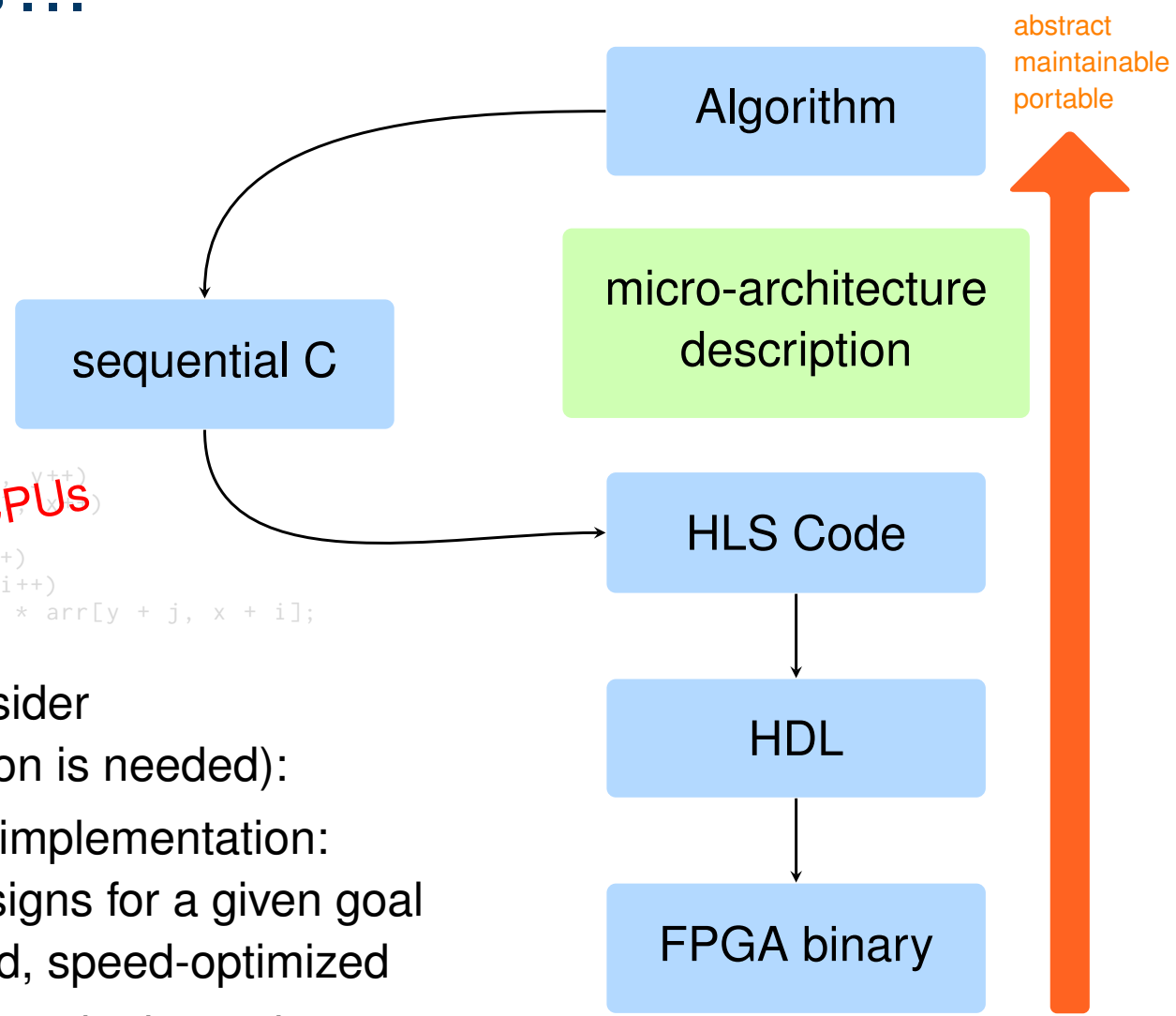


Abstraction Layers . . .

```
// input domain scan
for (int y = 1; y < height - 1, y++)
  for (int x = 1; x < width - 1, x++)
    // stencil function
    for (int j = -1; j < 2, j++)
      for (int i = -1; i < 2, i++)
        out(j, i) = mask[j, i] * arr[y + j, x + i];
```



Abstraction Layers . . .



Sequential C
is not the algorithm

```
// input domain scan
for (int y = 1; y < height - 1; y++)
  for (int x = 1; x < width - 1; x++)
    // stencil f
    for (int j = -1; j < 2, j++)
      for (int i = -1; i < 2, i++)
        out(j, i) = mask[j, i] * arr[y + j, x + i];
```

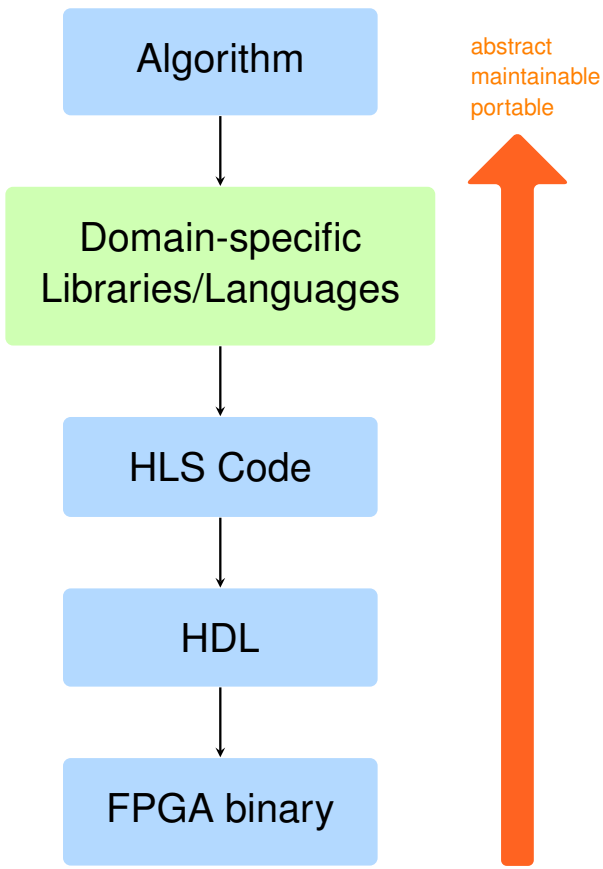
not even optimized for CPUs

More challenges to consider
(Design space exploration is needed):

- There is no "best" implementation:
  Pareto-optimal designs for a given goal
  i. e., area-optimized, speed-optimized
- Pareto-optimality mostly depends on
  the input parameters; i. e., image size, kernel size

Why not express algorithm more high level?

- identify abstractions that are relevant to the performance
- write code generic to these abstractions
- provide platform-specific impelementations to these abstractions



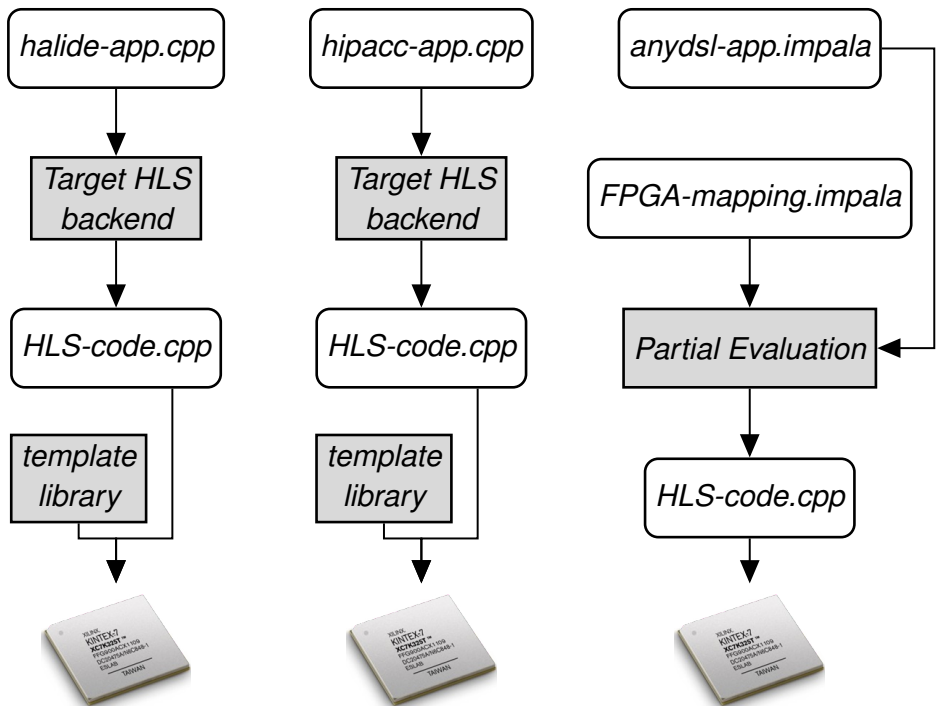
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

FACULTY OF ENGINEERING

Design of a DSL and AnyDSL

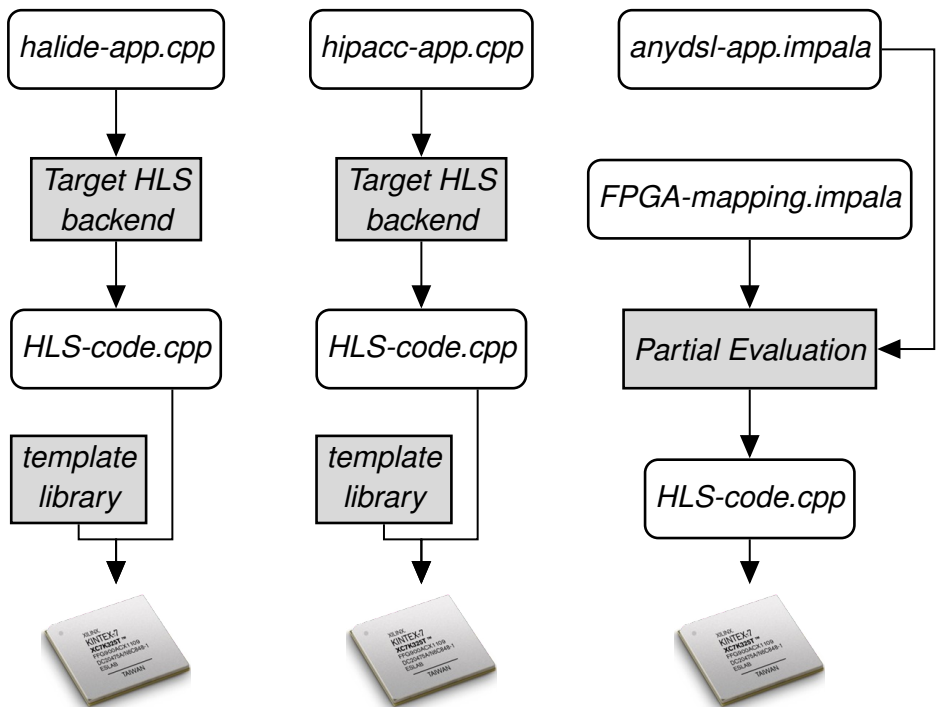
Embedded DSL Design Techniques

- **Deeply embedding:** the abstract syntax of the embedded program is represented as a data structure in the host program, i. e., Halide
- **Shallow embedding:** DSL is implemented within the host language itself i. e., Hipacc, Anydsl



AnyDSL

- gets metaprogramming out of the way
  (no difference between the host and the embedded program)
- exposes black boxes of the typical DSL compilers to the user
- provides an infrastructure for DSL design



AnyDSL

- Partial evaluation
- Higher order functions
- Target-specific code generation

specialize statically known variables at compile

- for the functions annotated via @
- unless the variables annotated via (?n) is not known

```
fn @(?n) pow(x: int, n: int) -> int
      {
  if n == 0 {
    1
  } else {
    if n % 2 == 0 {
      let y = pow(x, n / 2);2
      y * y2
    } else {
      x * pow(x, n - 1)2
    }
  }
}
```

thus the call

```
let z = pow(x, 5);      let z = pow(2, 5);
```

will result :

```
let y = x * x;
let z = x * y * y;        let z = 32
```

AnyDSL

- Partial evaluation
- Higher order functions
- Target-specific code generation

for protocol is a syntactic sugar to call a higher-order function.

Ex: loop unrolling

usage:

```
for i in unroll(0, range) {
    result += new[i];
}
```

description:

```
fn @unroll(lower: i32, upper: i32, body: fn(i32) -> ()) -> () {
  if lower < upper {
      @@body(lower);
      unroll(lower + 1, upper, body)
  }
}
```

get rid of the vendor-specific compiler directives (“#pragma HLS unroll”)

AnyDSL

- Partial evaluation
- Higher order functions
- Target-specific code generation

schedule:

```
for x in platform_loop(arr, mask) {
  out(x) = body(arr, mask, x);
}
```

CUDA Mapping:

```
fn @platform_loop(arr: &[f32], body: fn(&[f32], Mask, i32)->()) -> (){
  let dim = (arr.size , 1, 1);
  let block = (128, 1, 1);
  nvvm(dim , block , || {
    let x = nvvm_read_tid_x() + nvvm_read_ntid_x() * nvvm_read_ctid_x();
    body(arr, mask, x);
  });
}
```

CPU Mapping:

```
fn @platform_loop(arr: &[f32], body: fn(&[f32], Mask, i32)->()) -> (){
  vectorize(arr.size, 8, || -> () {
    let x = get_thread_id();
    body(arr, mask, x);
  });
}
```

Example: Stencil Functions

Decouple the **schedule** from the **algorithm**

ex: Stencil Functions

algorithm:

```
let mask = Mask { data : [ 0.2f, 0.5f, 0.2f ], ...  };
```

abstraction:

```
fn (?x) @stencil(arr: &[f32], mask: Mask, x: i32) -> f32 {
  let mut res = 0.0f;

  for i in unroll(-mask.lower, mask.upper) {
    let coeff = mask(i + mask.size / 2);
    if coeff != 0.0f {
      res += arr(x + i) * coeff;
    }
  }

  res
}
```

schedule:

```
fn @iteration(arr: &[f32], mask: Mask, body: fn(&[f32], Mask, i32) -> ()) {
  for x in platform_loop(arr) {
    out(x) = body(arr, mask, x);
  }
}
```

Example: Stencil Functions

Decouple the **schedule** from the **algorithm** how?

ex: Stencil Functions

algorithm:

```
// application
let mask = Mask { data : [ 0.2f, 0.5f, 0.2f ], ...  };
```

abstraction:

```
// domain or user : algorithmic abstractions
fn (?x) @stencil(arr: &[f32], mask: Mask, x: i32) -> f32 {
  let mut res = 0.0f;

  for i in unroll(-mask.lower, mask.upper) {
    let coeff = mask(i + mask.size / 2);
    if coeff != 0.0f {
      res += arr(x + i) * coeff;
    }
  }

  res
}
```

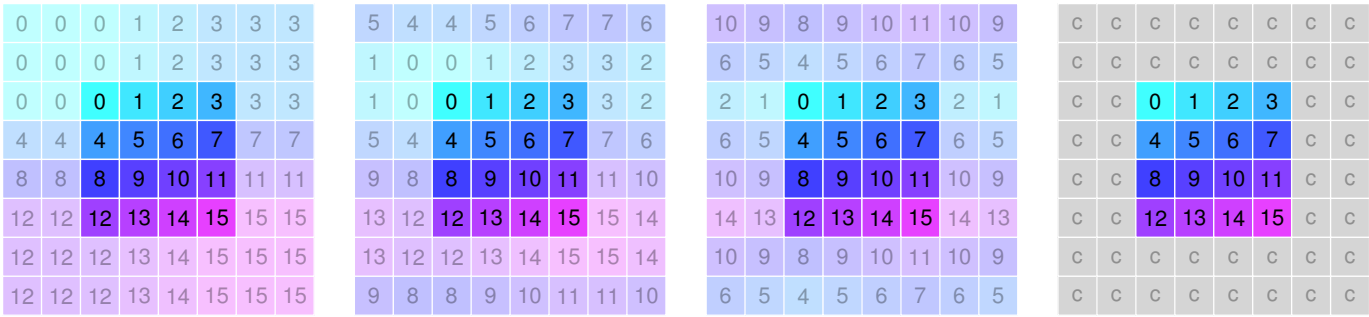
schedule:

```
// domain : platform mapping abstractions
fn @iteration(arr: &[f32], mask: Mask, body: fn(&[f32], Mask, i32) -> ()) {
  for x in platform_loop(arr) {
    out(x) = body(arr, mask, x);
  }
}
```

Border Handling

Border Handling

Virtually enlarging input space according to well-known border patterns for the out-of-bounds dependencies



(a) clamp (b) mirror (c) mirror-101 (d) constant

Border handling as a case study to present

- how to higher the abstraction level for HLS
- how to design a DSL abstraction via Impala's *partial evaluation* and *functional language* features

How to express an *algorithm* without a platform-specific *schedule*?

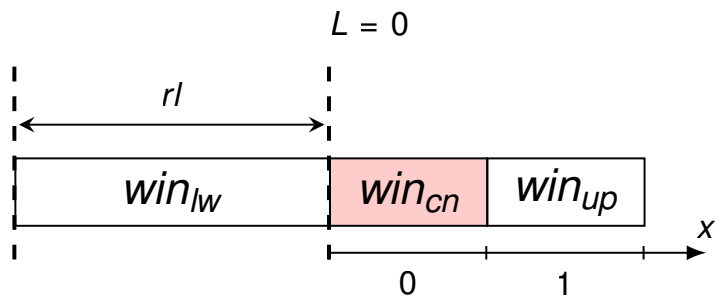
Border Handling: Stencil Window

Lets derive mathematical expressions:

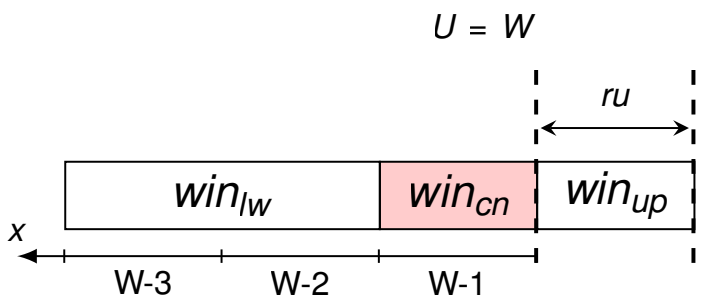
1D stencil function:

$$y[x_{cl}, x_{cu-1}] = f(\{x_{wl}, \cdots, x_{wcl}, \cdots, x_{wcu}, \cdots, x_{wu-1}\})$$

stencil window at the edges:



(a) Window is at the *left* border (x= 0). Border handling is necessary for the win_{lower}.



(b) Window is at the *right* border (x= $W - 1$). Border handling is necessary for the win_{upper}.

Border Handling: Decoupling the iteration schedule

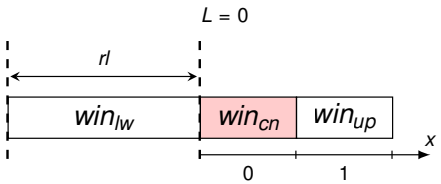
Border Handling Conditions:

$$b_{clamp}(x,[L,U]) = \begin{cases} L & L > x \geq L - rl \\ U-1 & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

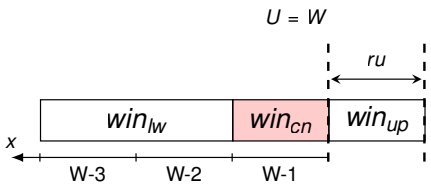
$$b_{mirror}(x,[L,U]) = \begin{cases} L + (L - x - 1) & L > x \geq L - rl \\ U + (U - x - 1) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$b_{mirror101}(x,[L,U]) = \begin{cases} L + (L - x) & L > x \geq L - rl \\ U + (U - x - 2) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$fb_{constant}(x,in,cval,[L,U]) = \begin{cases} cval & L > x \geq L - rl \\ cval & U + ru > x \geq U \\ in[x] & \text{else} \end{cases}$$



(a) x= 0



(b) x= $W - 1$

Border Handling: Decoupling the iteration schedule

$$f_{bh}(x, in, \cdots) = \begin{cases} f_{bh\_lower}(x, in, L, \cdots) & L > x \geq L - rl \\ f_{bh\_upper}(x, in, U, \cdots) & U + ru > x \geq U \\ f_{bh\_center}(x, in, \cdots) & U > x \geq L \end{cases}$$

$$b_{clamp}(x, [L, U]) = \begin{cases} L & L > x \geq L - rl \\ U - 1 & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

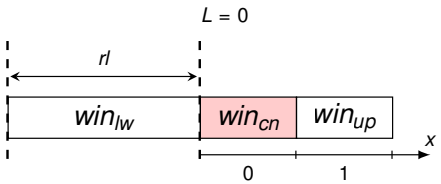
$$b_{mirror}(x, [L, U]) = \begin{cases} L + (L - x - 1) & L > x \geq L - rl \\ U + (U - x - 1) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$b_{mirror101}(x, [L, U]) = \begin{cases} L + (L - x) & L > x \geq L - rl \\ U + (U - x - 2) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

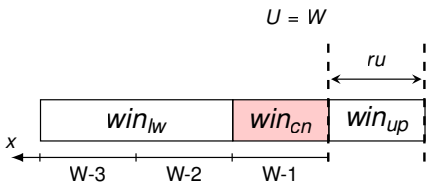
$$fb_{constant}(x, in, cval, [L, U]) = \begin{cases} cval & L > x \geq L - rl \\ cval & U + ru > x \geq U \\ in[x] & \text{else} \end{cases}$$

Data assignments: Algorithm

Coordinate checks: Iteration schedule



(a) x= 0



(b) x= $W - 1$

Border Handling: Decoupling the iteration schedule

$$f_{bh}(x, in, \cdots) = \begin{cases} f_{bh\_lower}(x, in, L, \cdots) & L > x \geq L - rl \\ f_{bh\_upper}(x, in, U, \cdots) & U + ru > x \geq U \\ f_{bh\_center}(x, in, \cdots) & U > x \geq L \end{cases}$$

$$f_{bh}(x, in, \cdots) = \begin{cases} fb_{bh}(x, in, \cdots) & bh = constant \\ in\,[\, b_{bh}(x)\,] & bh = others \end{cases}$$

$$b_{clamp}(x, [L, U]) = \begin{cases} L & L > x \geq L - rl \\ U - 1 & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$b_{mirror}(x, [L, U]) = \begin{cases} L + (L - x - 1) & L > x \geq L - rl \\ U + (U - x - 1) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$b_{mirror101}(x, [L, U]) = \begin{cases} L + (L - x) & L > x \geq L - rl \\ U + (U - x - 2) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$fb_{constant}(x, in, cval, [L, U]) = \begin{cases} cval & L > x \geq L - rl \\ cval & U + ru > x \geq U \\ in[x] & \text{else} \end{cases}$$

```
enum BoundaryMode {
    Index(i32),
    Const(pixel_t)
}
```

Data assignments: Algorithm

Coordinate checks: Iteration schedule

Border Handling: Decoupling the iteration schedule

$$f_{bh}(x, in, \cdots) = \begin{cases} f_{bh\_lower}(x, in, L, \cdots) & L > x \geq L - rl \\ f_{bh\_upper}(x, in, U, \cdots) & U + ru > x \geq U \\ f_{bh\_center}(x, in, \cdots) & U > x \geq L \end{cases}$$

$$f_{bh}(x, in, \cdots) = \begin{cases} fb_{bh}(x, in, \cdots) & bh = constant \\ in\,[\, b_{bh}(x)\,] & bh = others \end{cases}$$

$$b_{mirror}(x, [L, U]) = \begin{cases} L + (L - x - 1) & L > x \geq L - rl \\ U + (U - x - 1) & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

$$fb'_{constant}(x, in, cval, [L, U]) = \begin{cases} cval & L > x \geq L - rl \\ cval & U + ru > x \geq U \\ x & \text{else} \end{cases}$$

Data assignments: Algorithm

Coordinate checks: Iteration schedule

```
enum BoundaryMode {
    Index(i32),
    Const(pixel_t)
}
```

Algorithm:

Data Assignments

```
fn @mirror_lower(idx: int, lower: int, upper: int) -> BoundaryMode {
  BoundaryMode::Index(if idx  < lower { lower + (lower - idx-1) } else { idx })
}
fn @mirror_upper(idx: int, lower: int, upper: int) -> BoundaryMode {
  BoundaryMode::Index(if idx >= upper { upper - (idx+1 - upper) } else { idx })
}

fn @const_lower(idx: i32, lower: i32, upper: i32, cval: pixel_t) -> BoundaryMode {
    if idx < lower { BoundaryMode::Const(cval) } else { BoundaryMode::Index(idx) }
}
fn @const_upper(idx: i32, lower: i32, upper: i32, cval: pixel_t) -> BoundaryMode {
    if idx >= upper { BoundaryMode::Const(cval) } else { BoundaryMode::Index(idx) }
}
```

Border Handling: Decoupling the iteration schedule

$$f_{bh}(x, in, \cdots) = \begin{cases} f_{bh\_lower}(x, in, L, \cdots) & L > x \geq L - rl \\ f_{bh\_upper}(x, in, U, \cdots) & U + ru > x \geq U \\ f_{bh\_center}(x, in, \cdots) & U > x \geq L \end{cases}$$

Schedule:

Iteration Coordinate Checks

```
fn @get_data(x: i32, read: fn(i32) -> pixel_t, boundary: Boundary, L: i32, U: i32,
             bh_lower: BoundaryFn, bh_upper: BoundaryFn) -> pixel_t {
  let mode = match boundary {
    Boundary::Lower  => bh_lower(x, L),
    Boundary::Center => BoundaryMode::Index(x),
    Boundary::Upper  => bh_upper(x, U)
  };

  match mode {
    BoundaryMode::Index(idx) => read(idx),
    BoundaryMode::Const(c)   => c
  }
}
```

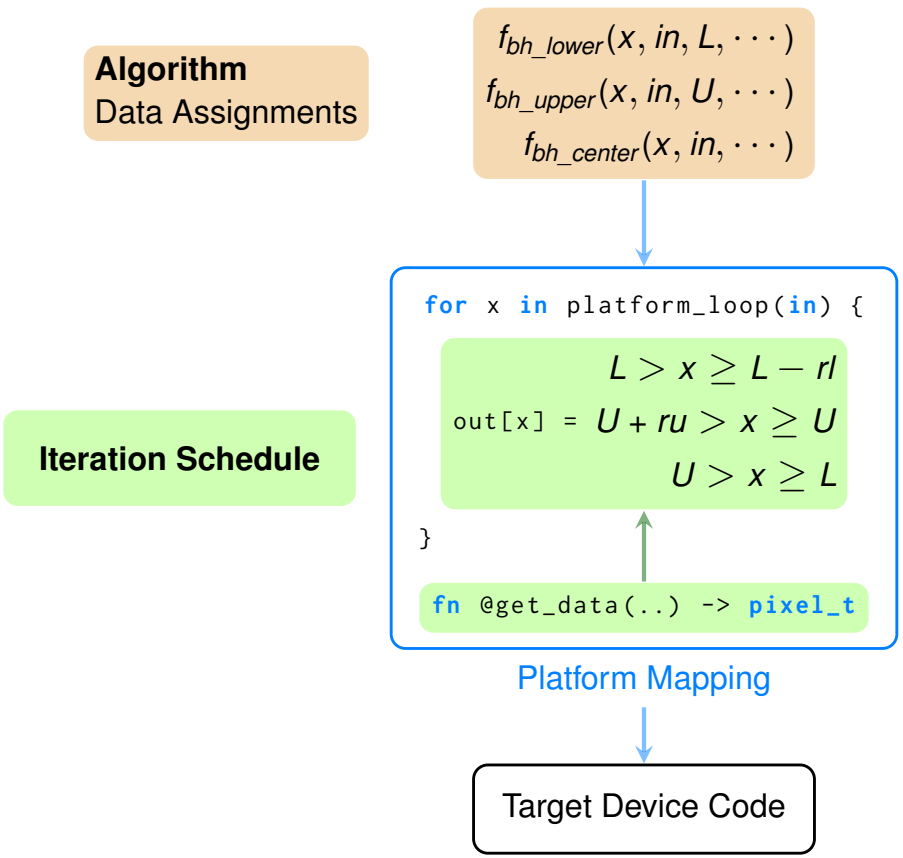
Algorithm:

Data Assignments

```
fn @mirror_lower(idx: int, lower: int, upper: int) -> BoundaryMode {
  BoundaryMode::Index(if idx  < lower { lower + (lower - idx-1) } else { idx })
}
fn @mirror_upper(idx: int, lower: int, upper: int) -> BoundaryMode {
  BoundaryMode::Index(if idx >= upper { upper - (idx+1 - upper) } else { idx })
}

fn @const_lower(idx: i32, lower: i32, upper: i32, cval: pixel_t) -> BoundaryMode {
    if idx < lower { BoundaryMode::Const(cval) } else { BoundaryMode::Index(idx) }
}
fn @const_upper(idx: i32, lower: i32, upper: i32, cval: pixel_t) -> BoundaryMode {
    if idx >= upper { BoundaryMode::Const(cval) } else { BoundaryMode::Index(idx) }
}
```

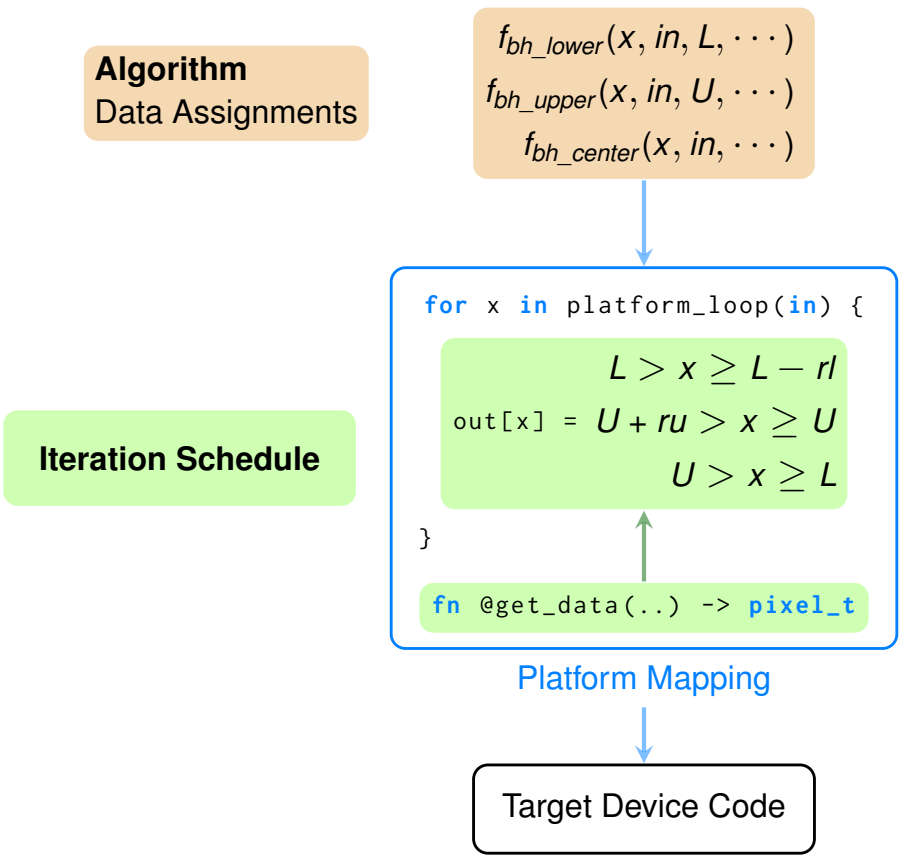
Border Handling: Platform Mapping (I/III)

Platform mapping is a function that takes as input parameter an algorithm and generates a target-specific structural code



Border Handling: Platform Mapping (I/III)

Platform mapping is a function that takes as input parameter an algorithm and generates a target-specific structural code



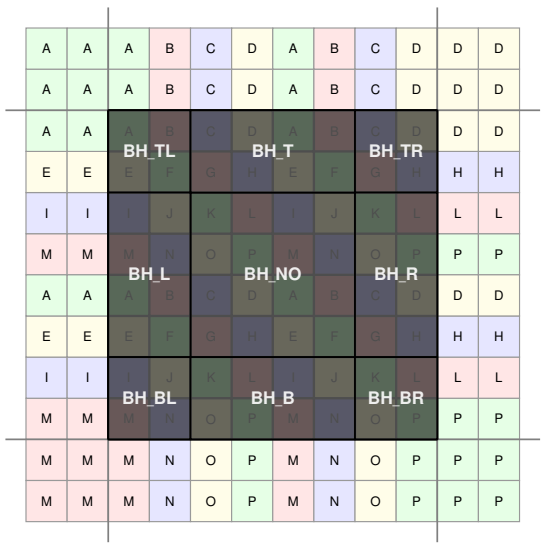


Figure: An efficient GPU mapping divides a kernel to 9 different regions and specializes the pixel reads accordingly, thus avoids thread divergence.

Border Handling: Data read from a window

Execution can mostly be accelerated when the dependency pixels are transferred to a faster memory for a faster calculation

- caching to a shared memory, on-chip memory, registers

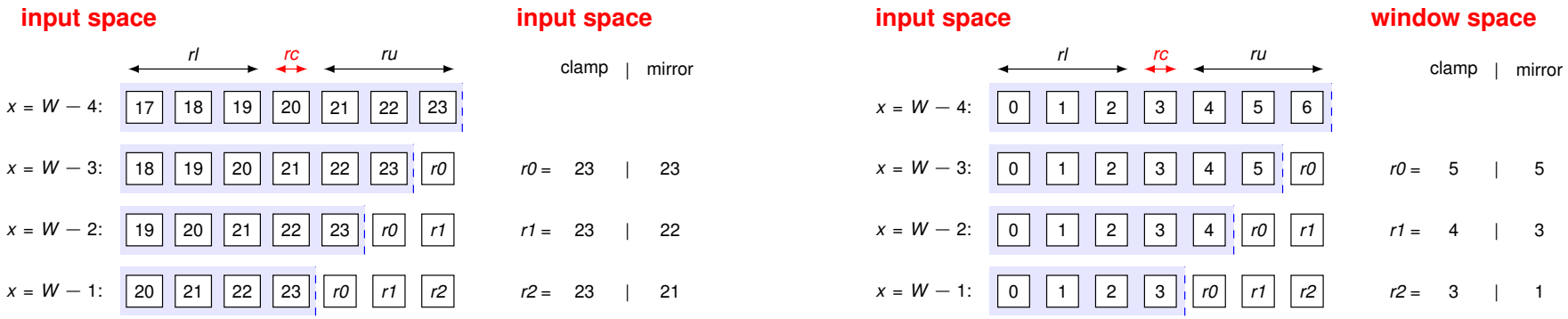
complicates border handling: the data assignments depend on the coordinates of both the input and the fast memory spaces

Border Handling: Data read from a window

Execution can mostly be accelerated when the dependency pixels are transferred to a faster memory for a faster calculation

- caching to a shared memory, on-chip memory, registers

complicates border handling: the data assignments depend on the coordinates of both the input and the fast memory spaces

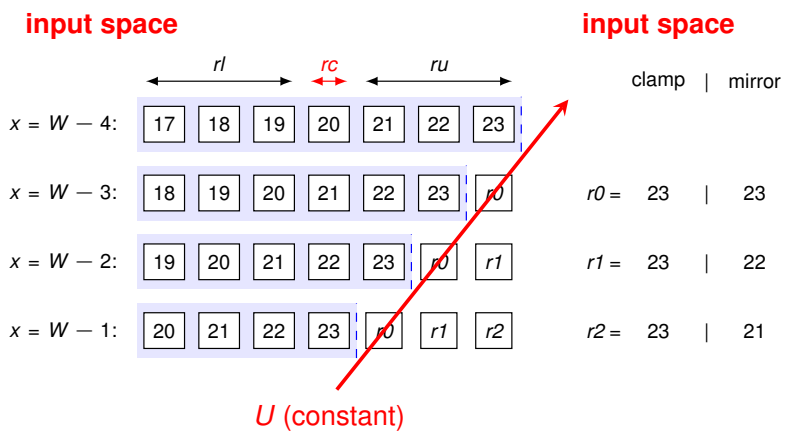


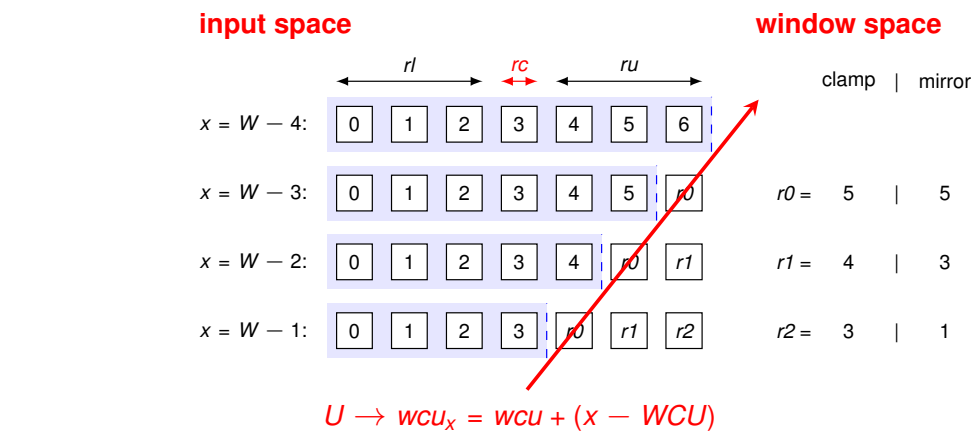
(a) input-based (b) window-based

Figure: Border handling for an example 1/d window ($L = 0$, $WCL = 3$, $WCU = 4$, $U = 7$) at the lower border

Border Handling: Data read from a window

```
fn @(?x) get_data_wnd(x: i32, i: i32, read: fn(i32) -> pixel_t, boundary: Boundary,
                      L: i32, WCL: i32, WCU: i32, U: i32, wcl: i32, wcu: i32,
                      bh_lower: BoundaryFn, bh_upper: BoundaryFn) -> pixel_t {
  let (wclx, wcux) = match boundary {
    Boundary::Lower  => (wcl - (x - L), 0),
    Boundary::Center => (0, 0),
    Boundary::Upper  => (0, wcu + (x - WCU))
  };
  get_data(i, read, boundary, wclx, wcux, bh_lower, bh_upper)
}
```



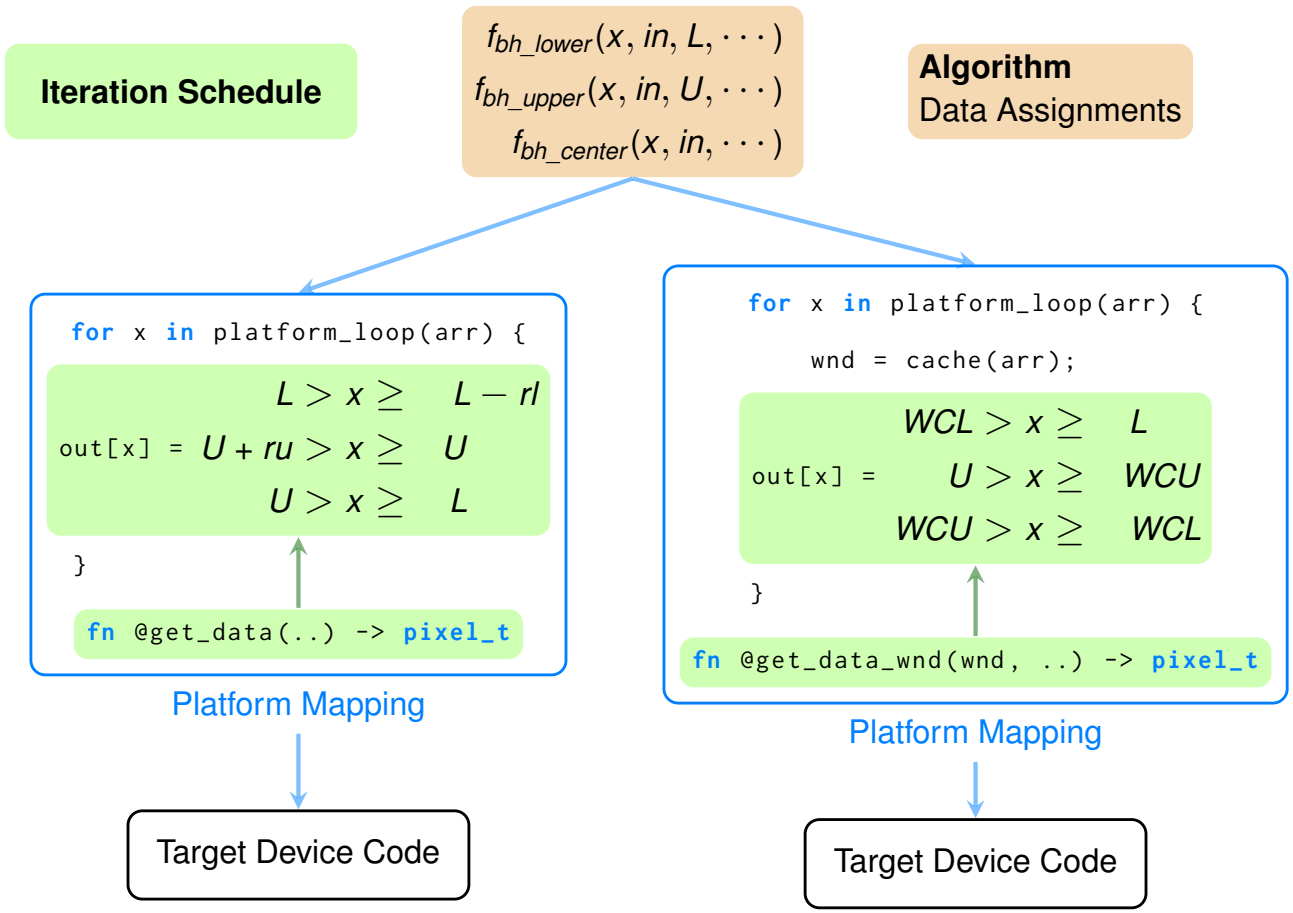


(a) input-based (b) window-based

Figure: Border handling for an example 1/d window ($L = 0$, $WCL = 3$, $WCU = 4$, $U = 7$) at the lower border

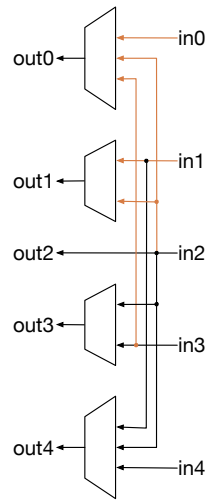
Border Handling: Platform Mapping (II/III)

The same algorithm description can be used with different platform mappings



Border Handling: MUX Network for FPGA (Design Rules)

FPGA implementations allocate corresponding MUXes for data selection

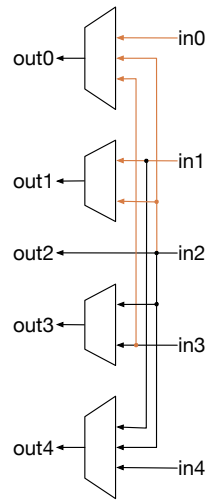


Border Handling: MUX Network for FPGA (Design Rules)

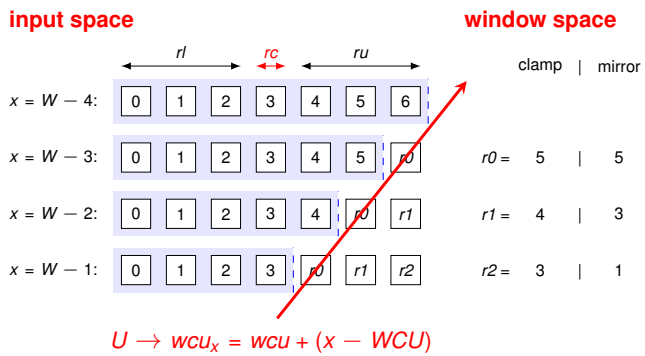
FPGA implementations allocate corresponding MUXes for data selection

R1 input and output switching for all possible conditional cases should be known at compile time

```
Boundary::Lower => {
  for wi in unroll(0, wl) { // for every possible wclx
    if x == L + i {
      // data assignments ( wclx = wcl - wi )
      ... = get_data(..., wcl - wi, 0, ...);
    }
  }
}
```



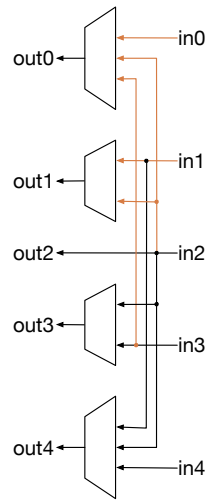

```
fn @(?x) get_data_wnd(···) -> pixel_t {
  let (wclx, wcux) = match boundary {
    Boundary::Lower  => (wcl - (x - L), 0),
    Boundary::Center => (0, 0),
    Boundary::Upper  => (0, wcu + (x - WCU))
  };

  get_data(i, read, boundary, wclx, wcux, bh_lower, bh_upper)
}
```



Border Handling: MUX Network for FPGA (Design Rules)

FPGA implementations allocate corresponding MUXes for data selection

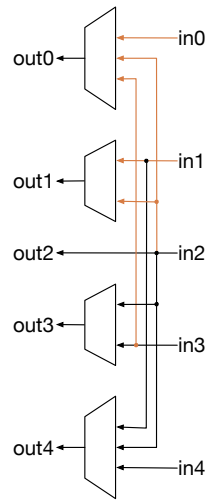
- R1 input and output switching for all possible conditional cases should be known at compile time
- R2 arithmetic operations of both *true* and *false* cases should always be calculated before the data assignments



Border Handling: MUX Network for FPGA (Design Rules)

FPGA implementations allocate corresponding MUXes for data selection

- R1 input and output switching for all possible conditional cases should be known at compile time
- R2 arithmetic operations of both *true* and *false* cases should always be calculated before the data assignments
- R3 all data assignments depending on the same conditional signal should be written in the same `if/else`




```
// instead of these
if common_cond { win(0, 3) = get_data(..., wclx, ...); }
if common_cond { win(1, 3) = get_data(..., wclx, ...); }

// write below for less area
fn @ex_assign(common_cond: bool) -> () {
  win(0, 3) = get_data(..., wclx, ...);
  win(1, 3) = get_data(..., wclx, ...);
}
if common_cond { ex_assign(...); }
```

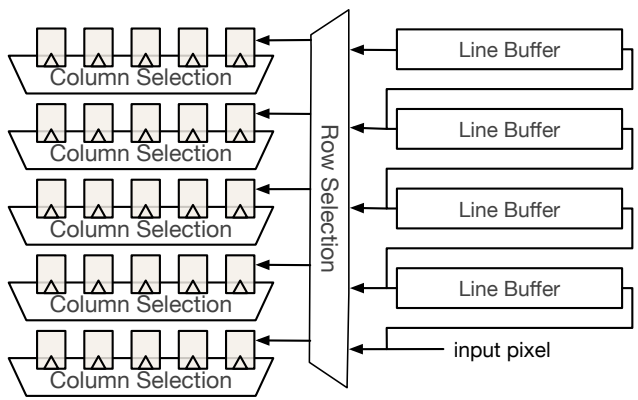
Border Handling: MUX Network for FPGA (Our Solution)

Expect as input parameter parameter a data assignment function:

```
fn @assign_data(/* both read and write functions*/) -> () {
  for /* all the assignments */ {
    let data = get_data(i, read, boundary, wclx, wcux, bh_lower, bh_upper);
    out.write(xw, ..., data);
  }
}
```

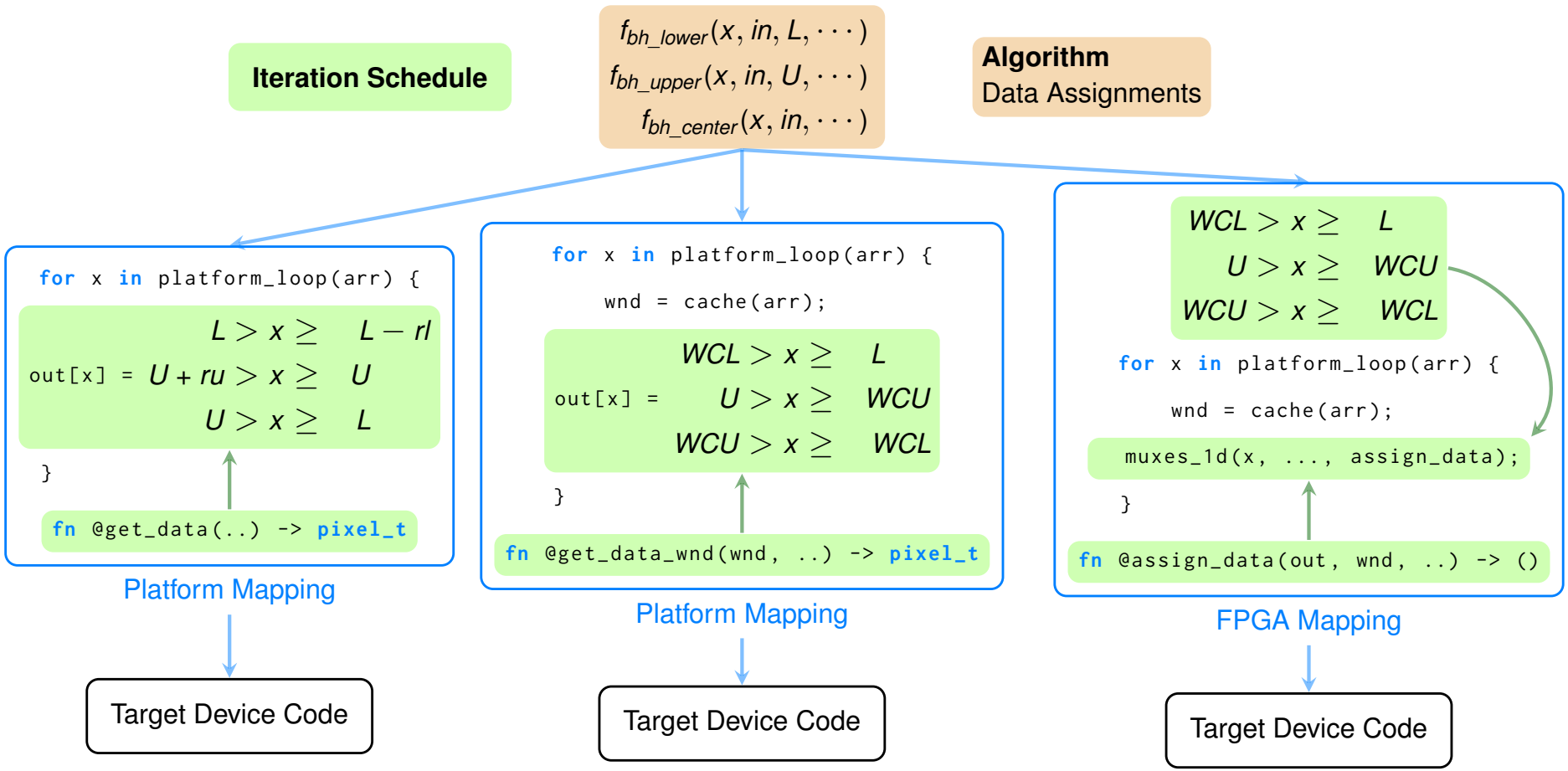
Describe a MUX-network, explicitly:

```
fn @(?x) muxes_for_the_bound(/*..., assign_data, bh_lower, bh_upper*/) -> () {
  // ...

  // extend assign_data to the boundary
  fn @for_the_bound(/* boundary, wclx, wcux */) -> () {
    for i in unroll(lower_w, upper_w) {
      assign_data(i, boundary, wclx, wcux, bh_lower, bh_upper);}}

  // Boundary::Center (default)
  for_the_bound(Boundary::Center, 0, 0);

  match boundary {
    Boundary::Lower => {
      for wi in unroll_step(0, rb, v) {
        let wclx = wcl - wi;
        if x == L + wi / v {
          for_the_bound(boundary, wclx, 0);}}
    },
    Boundary::Upper => {
      for wi in unroll_step (0 , rb, v) {
        let wcux = wcu + wi;
        if x == U - 1 - wi / v {
          for_the_bound(boundary, 0, wcux);}}
    },
  }
}
```



Border Handling: Platform Mapping (III/III)

Our hardware-inspired MUX array description deploys static assignments by generating an `if` expression for every possible corner case of border handling.



Evaluation and Results

NDRange vs DSL Approach

The cost of the border handling is very minor with our DSL approach.

Table: 5 × 5 mean filter for an input of size 1024 × 1024.

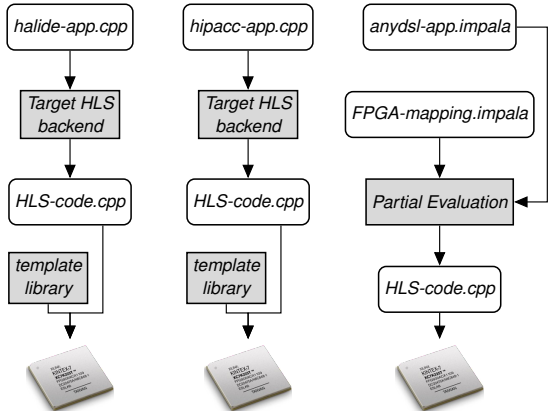
Border	latency (ms)	M10K	ALM	ALUT	FF
nobh	141.980	392	30087	40068	53688
clamp	83.359	570	31099	43178	59086
mirror	-	570	31569	44108	59679

(a) NDRange OpenCL kernel

Border	latency (ms)	M10K	ALM	ALUT	FF
nobh	12.009	346	27589	26507	45153
clamp	12.358	347	28324	27147	45968
mirror	12.276	347	28288	27010	45951

(b) Single-threaded OpenCL kernel generated from Impala

https://github.com/AnyDSL/stincilla



Thanks for listening.

Any questions?

Title A Journey into DSL Design using Generative Programming: FPGA Mapping of Image Border Handling through Refinement

Speaker M. Akif Özkan, akif.oezkan@fau.de

Backup Slides

Application Code

Convolution example as a proof-of-concept “DSL”:

```
fn main() -> () {
  // images
  let arr = create_img(width, height);
  let dx  = create_img(width, height);

  // mask
  let mask = get_mask3([[-1,  0,  1], [-2, 0, 2], [-1, 0, 1]]);

  // border handling
  let lower = mirror_lower;
  let upper = clamp_upper;

  // stencil function
  for math, out, arr, mask in iteration(math, dx, arr, mask, lower, upper) {
    out.write(stencil(arr, mask));
  }
}
```

- DSL abstractions hide implementation details from the application developer
- Syntactic sugars facilitate more user friendly descriptions

Application Code

Convolution example as a proof-of-concept “DSL”:

```
fn main() -> () {
  // images
  let arr = create_img(width, height);
  let dx  = create_img(width, height);

  // mask
  let mask = get_mask3([[-1,  0,  1], [-2, 0, 2], [-1, 0, 1]]);

  // border handling
  let lower = mirror_lower;
  let upper = clamp_upper;

  // stencil function
  for math, out, arr, mask in iteration(math, dx, arr, mask, lower, upper) {
    out.write(stencil(arr, mask));
  }
}
```

- DSL abstractions hide implementation details from the application developer
- Syntactic sugars facilitate more user friendly descriptions