Hardware Design and Analysis of Efficient Loop Coarsening and Border Handling for Image Processing

M. Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich
Hardware/Software Co-Design, Friedrich-Alexander University Erlangen-Nürnberg
ASAP, July 11, 2017, Seattle
Motivation: Coarse-grained parallelism on FPGA

Memory bandwidth limits can be reached by processing multiple pixels per cycle:

- A memory bandwidth of 12 GBytes/s for each DDR3 channel is feasible on a “modern” FPGA, which leads to 512 bit wide interfaces for around 200 MHz logic frequency (Zynq zc706)
- High-speed serial transceiver technology on FPGAs enables the communication interfaces to operate at high data rates
Motivation: Coarse-grained parallelism on FPGA

Memory bandwidth limits can be reached by processing multiple pixels per cycle:
- A memory bandwidth of 12 GBytes/s for each DDR3 channel is feasible on a “modern” FPGA, which leads to 512 bit wide interfaces for around 200 MHz logic frequency (Zynq zc706)
- High-speed serial transceiver technology on FPGAs enables the communication interfaces to operate at high data rates

How to provide efficient coarse-grained parallelism for image processing applications?
Motivation: Image Processing Applications

We can define three characteristic data operations in image processing applications:

**Point Operators:**
Output data is determined by single input data

**Local Operators:**
Output data is determined by a local region of the input data (stencil pattern-based calculations)

**Global Operators:**
Output data is determined by all of the input data
Motivation: Image Processing Applications

A great portion of image processing applications can be described as task graphs of point, local, and global operators:

An example task graph for Harris Corner Detection (square: local operator, circle: point operator)
Motivation: Parallelization of Image Processing Applications

A naive way would be replicating the accelerator hardware:
Motivation: Parallelization of Image Processing Applications

Is there a more resource-efficient approach?
Motivation: Parallelization of point operators

Coarse-grained parallelization of point operators is rather straightforward:

The throughput is linear with the resource usage (when further data-path optimizations are ignored).
Motivation: Parallelization of point operators

Coarse-grained parallelization of point operators is rather straightforward:

The throughput is linear with the resource usage (when further data-path optimizations are ignored).

What are efficient parallelization methods for local operators?
Motivation: Image border handling

- a fundamental image processing issue for local operators
- mostly overlooked by the digital hardware design community
- should be considered together with coarse-grained parallelization

Common border handling modes.
Outline

Loop Coarsening

Border Handling

Best Architecture Selection

Evaluation and Results
Loop Coarsening
Loop Coarsening: Schmid’s Approach

Coarsening the outer horizontal loop of a 2D input by a factor of $v$:

```c
for (int y = 0; y < IMAGE_HEIGHT; y++){
    for (int x = 0; x < IMAGE_WIDTH; x + v){
       (DataBeatType*)(out[y][x]) = local_op(stencil_p1(y, x), ..);
    }
}
```

Raster order processing facilitates burst mode read, thus highest external memory bandwidth!

---

Loop Coarsening: Schmid’s Approach

The line buffer and sliding window are modified to store so-called *data beats*.

The throughput is *sub-linear* with the resource usage.
Loop Coarsening: Schmid’s Approach

The line buffer and sliding window are modified to store so-called *data beats*.

The throughput is *sub-linear* with the resource usage.
Loop Coarsening: Schmid’s Sliding Window

current timestamp: (coarsening) initial latency - 1

FETCH: 0 1 2 3  
CALC: 
OUT:

(kernel width) \( w = 3 \), (coarsening factor) \( v = 4 \)
Loop Coarsening: Schmid’s Sliding Window

current timestamp: (coarsening) initial latency

FETCH: 0 1 2 3 4 5 6 7
CALC: 0 1 2 3
OUT: 0 1 2 3

(kernel width) $w = 3$, (coarsening factor) $v = 4$
Loop Coarsening: Schmid’s Sliding Window

current timestamp: (coarsening) initial latency +1

FETCH: 0 1 2 3 4 5 6 7 8 9 10 11  CALC: 0 1 2 3 4 5 6 7  OUT: 0 1 2 3 4 5 6 7

(kernel width) \( w = 3 \), (coarsening factor) \( v = 4 \)

Deploys additional registers when \( r_w \mod v \neq 0 \)
Loop Coarsening: Fetch and Calc (F&C)

Redundant registers in Schmid’s architecture are eliminated

(kernel width) $w = 3$, (coarsening factor) $v = 4$
Loop Coarsening: Calc and Pack (C&P)

current timestamp: (coarsening) initial latency - 1

FETCH: 0 1 2 3
CALC: x 0 1 2
OUT:

(kernel width) $w = 3$, (coarsening factor) $v = 4$
Loop Coarsening: Calc and Pack (C&P)

current timestamp: (coarsening) initial latency

FETCH: 0 1 2 3 4 5 6 7
CALC: x 0 1 2 3 4 5 6
OUT: 0 1 2 3

(kernel width) \( w = 3 \), (coarsening factor) \( v = 4 \)

\[
\text{output}([x, x + v], t) = \text{pack}\{\text{out}([0, v - r_w - 1], t - 1), \text{out}([v - r_w, v - 1], t)\}
\]
Loop Coarsening: Calc and Pack (C&P)

current timestamp: (coarsening) initial latency + 1

**FETCH:** 0 1 2 3 4 5 6 7 8 9 10 11  **CALC:** x 0 1 2 3 4 5 6 7 8 9 10  **OUT:** 0 1 2 3 4 5 6 7

(kernel width) \( w = 3 \), (coarsening factor) \( v = 4 \)

\[
\text{output}([x, x + v], t) = \text{pack}\{\text{out}([0, v - r_w - 1], t - 1), \text{out}([v - r_w, v - 1], t)\}
\]
Loop Coarsening Overview

Resource usage is the # of registers when border handling is ignored:

**Schmid’s:**
\[ k_{in} \cdot h \cdot (v + 2 \cdot (v \cdot \lceil r_w/v \rceil)) \]

**Fetch And Calc:**
\[ C_{F&C}^{reg} = k_{in} \cdot h \cdot (r_w + v \cdot (\lceil r_w/v \rceil + 1)) \]

**Calc and Pack:**
\[ C_{C&P}^{reg} = k_{in} \cdot h \cdot (2 \cdot r_w + v) + k_{out} \cdot (v - (r_w \mod v)) \]
Border Handling
Image Border Handling

**Conditional Selection** appropriate input is selected before the calculation

**Padding** input is enlarged with the appropriate border pixels
output is smaller than the input: \((W + \lfloor w/2 \rfloor, H + \lfloor h/2 \rfloor) \rightarrow (W, H)\)

**Modification of the calculation** calculation depends on the image coordinates
Image Border Handling

**Conditional Selection**  appropriate input is selected before the calculation
Focus of this work!

**Padding**  input is enlarged with the appropriate border pixels
output is smaller than the input: \((W + \lfloor w/2 \rfloor, H + \lfloor h/2 \rfloor) \rightarrow (W, H)\)
Stalls the stream before (or during) the processing of a local operator!

**Modification of the calculation**  calculation depends on the image coordinates
Specific to the target algorithm!

```
10  9  8  9 10 11 12 13 14 15 14 13
  6  5  4  5  6  7  8  9 10 11 10  9
   2  1  0  1  2  3  4  5  6  7  6  5
   6  5  4  5  6  7  8  9 10 11 10  9
10  9  8  9 10 11 12 13 14 15 14 13
14 13 12 13 14 15 16 17 18 19 18 17
18 17 16 17 18 19 20 21 22 23 22 21
22 21 20 21 22 23 24 25 26 27 26 25
26 25 24 25 26 27 28 29 30 31 30 29
30 29 28 29 30 31 32 33 34 35 34 33
26 25 24 25 26 27 28 29 30 31 30 29
22 21 20 21 22 23 24 25 26 27 26 25
```
mirror-101
Analysis of Border Handling: Selections

Is there an optimal implementation in terms of area resources, and speed? How many selections does the optimal implementation require?
Analysis of Border Handling: Selections

Let $P_{i,j}(x,y)$ be a mapping to the $(i,j)$th output from the sliding window.

**ex:** $P_{0,2}(1,0): sw(2,2) \rightarrow out(0,2)$,
$P_{1,2}(1,0): sw(1,2) \rightarrow out(1,2)$,
$P_{2,2}(1,0): sw(2,2) \rightarrow out(2,2)$,
$P_{3,2}(1,0): sw(3,2) \rightarrow out(3,2)$,
$P_{4,2}(1,0): sw(4,2) \rightarrow out(4,2)$
Analysis of Border Handling: Selections

Let $P_{i,j}(x,y)$ be a mapping to the $(i,j)$th output from the sliding window $I_s$ be the union of all input combinations; then the conditional selections in $x$ and $y$ are orthogonal to each other:

$$|I_s| = |I_w \times I_h| = |I_w| \cdot |I_h| = (1 + 2 \cdot \sum_{i=2}^{\lceil w/2 \rceil} i) \times (1 + 2 \cdot \sum_{i=2}^{\lceil h/2 \rceil} i) = 121 \text{ (for 5-by-5)}$$

$$I_s$$ for a $5 \times 5$ local op. with mirror-101 border mode. Coordinates $\textit{else}$ and $\textit{all}$ cover redundant indices.
Analysis of Border Handling: Selections

Let $P_{i,j}(x,y)$ be a mapping to the $(i,j)$th output from the sliding window, and $I_s$ be the union of all input combinations; then the conditional selections in $x$ and $y$ are orthogonal to each other:

$$|I_s| = |I_w \times I_h| = |I_w| \cdot |I_h| = (1 + 2 \cdot \sum_{i=2}^{\lfloor w/2 \rfloor} i) \times (1 + 2 \cdot \sum_{i=2}^{\lfloor h/2 \rfloor} i) = 121 \text{ (for 5-by-5)}$$


\[
\begin{array}{|l|cc|cc|cc|cc|}
\hline
i,j & x,y & 0 & 1 & 0 & 1 & 2 & 3 & 4 \\
\hline
& else & (0,0) & (2,0) & (4,0) & (1,0) & (3,0) & (2,0) & (4,0) \\
0 & 1 & (0,2) & (2,2) & (4,2) & (1,2) & (3,2) & (2,2) & (4,2) \\
0 & else & (0,4) & (2,4) & (4,4) & (1,4) & (3,4) & (2,4) & (4,4) \\
1 & else & (0,1) & (2,1) & (4,1) & (1,1) & (3,1) & (2,1) & (4,1) \\
0 & & (0,3) & (2,3) & (4,3) & (1,3) & (3,3) & (2,3) & (4,3) \\
1 & all & (0,2) & (2,2) & (4,2) & (1,2) & (3,2) & (2,2) & (4,2) \\
3 & else & (0,3) & (2,3) & (4,3) & (1,3) & (3,3) & (2,3) & (4,3) \\
& H-1 & (0,1) & (2,1) & (4,1) & (1,1) & (3,1) & (2,1) & (4,1) \\
4 & else & (0,4) & (2,4) & (4,4) & (1,4) & (3,4) & (2,4) & (4,4) \\
& H-2 & (0,2) & (2,2) & (4,2) & (1,2) & (3,2) & (2,2) & (4,2) \\
& H-1 & (0,0) & (2,0) & (4,0) & (1,0) & (3,0) & (2,0) & (4,0) \\
\hline
\end{array}
\]

$I_s$ for a $5 \times 5$ local op. with mirror-101 border mode. Coordinates else and all cover redundant indices.
Analysis of Border Handling: Selections

Let \( P_{i,j}(x, y) \) be a mapping to the \((i, j)\)th output from the sliding window, and \( I_s \) be the union of all input combinations; then the conditional selections in \( x \) and \( y \) are orthogonal to each other:

\[
|I_s| = |I_w \times I_h| = |I_w| \cdot |I_h| = (1 + 2 \cdot \sum_{i=2}^{w/2} i) \times (1 + 2 \cdot \sum_{i=2}^{h/2} i) = 121 \text{ (for 5-by-5)}
\]

This implies that

\[
|I_h| + h \cdot |I_w| = 66 \text{ (for 5-by-5)}
\]

number of mappings is sufficient when \( x \) and \( y \) selections are separated.
Hardware Architecture: Type-0

- Separated border handling architecture
- Uses the row selection for the column selection

Separated border handling architecture

Row selection (mirror-101)
Hardware Architecture: Type-0

- Separated border handling architecture
- Uses the row selection for the column selection

![Diagram showing hardware architecture](image-url)

Column selection (mirror-101)

Row selection (mirror-101)
Hardware Architecture: Type-0

- Separated border handling architecture
- Uses the row selection for the column selection

\[ C_{\text{reg}}^{\text{Type-0}}(b) = h \cdot k_{\text{in}} \cdot (2 \cdot r_w) \]

\[ T_{\text{CriticalPath}}^{\text{Type-0}} = \begin{cases} 
T(\text{MUX}[r_w + 1]), & \text{mirror-101, mirror, clamp} \\
T(\text{MUX}[2]), & \text{clamp2, constant}
\end{cases} \]
Hardware Architecture: Type-0

- Separated border handling architecture
- Uses the row selection for the column selection

\[ C_{\text{reg}}^{\text{Type-0}}(b) = h \cdot k_{\text{in}} \cdot (2 \cdot r_w) \]

\[ T_{\text{CriticalPath}}^{\text{Type-0}} = \begin{cases} 
T(\text{MUX}[r_w + 1]), & \text{mirror-101, mirror, clamp} \\
T(\text{MUX}[2]), & \text{clamp2, constant}
\end{cases} \]

Exploit the temporal locality in raster-order processing, improve the column selection!
Analysis of Border Handling: Temporal Locality

Temporal data flow for column data selection with a local operator of size $w = 7$. Blue background denotes valid image regions, $lX$ and $rX$ represent the appropriate pixel values for the corresponding border handling mode.
Assuming that the streaming is not stalled and one pixel is read in each cycle, at least \( r_w \) pixels per row must be fetched at \( x = 0 \) in order be able to initialize all column pixels. Therefore, the minimum number of registers in a row is

\[
C_{\text{reg}}^{\text{min}} = h \cdot k_{\text{in}} \cdot (w + r_w)
\]
### Analysis of Border Handling: Temporal Locality

Only the data selection before $R_{\text{fetch}}$ and blue portion of $R'_{\text{right}}$ can be optimized, when minimum number of registers is used in raster order processing.
Column Selection Architectures: Type-1

Let's minimize the selection at the $R_{right'}$ according to the analysis:
Column Selection Architectures: Type-2

Let's minimize the selection before the $R_{\text{fetch}}$ according to the analysis:
Column Selection Architectures: Mirror border mode

Type-0:
- not resource efficient
+ full flexibility for all the border modes

Type-1:
+ resource efficient for a great portion of design space, $w$, $v$, border mode.

Type-2:
+ fastest architecture
+ pareto-optimal depending on $w$, $v$, and technology mapping
Best Architecture Selection
Best Architecture Selection

input : \( w, h, \text{borderMode}, \nu, k_{\text{out}}, k_{\text{in}}, \text{designGoal} \)

output: BorderHandlingPattern, CoarseningArch

```
func selectParetoOptimal(BorderHandlingPattern, CoarseningArch, w, h, \text{borderMode}, \nu, k_{\text{out}}, k_{\text{in}}, \text{designGoal})

\( r_w = \left\lfloor \frac{w}{2} \right\rfloor \)

if borderMode = UNDEFINED then

if \( k_{\text{out}} < k_{\text{in}} \cdot h \) then

CoarseningArch ← Calc and Pack

else

CoarseningArch ← Fetch and Calc

end

BorderHandlingPattern ← none

else

if \( r_w \cdot (k_{\text{in}} \cdot h - k_{\text{out}} + 1) < \nu \cdot (k_{\text{in}} \cdot h - k_{\text{out}}) \) then

CoarseningArch ← Calc and Pack

else

CoarseningArch ← Fetch and Calc

end

if borderMode = (CLAMP \lor CONSTANT) then

BorderHandlingPattern ← Type-1

else

// borderMode = (MIRROR \lor MIRROR-101)

if (designGoal = \{speed\} \lor ((r_w + 1) \cdot \text{MUX}[2] - \text{MUX}[r_w + 1] - \text{MUX}[2] < 0) then

BorderHandlingPattern ← Type-2

else

BorderHandlingPattern ← Type-1

end

end

end
```
Evaluation and Results
Comparison of Loop Coarsening Architectures

HLS estimation results of the proposed coarsening architectures (target clock frequency is 200 MHz, and no border handling is applied)
Comparison of Border Handling Architectures

HLS estimation results of the proposed border handling architectures (target clock frequency is 200 MHz, and border mode is mirror)
Type-1 vs Type-2

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Implementation (CPtar = 3.1 ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>w/h</td>
</tr>
<tr>
<td></td>
<td>BH Patt.</td>
</tr>
<tr>
<td>SLICE</td>
<td>BRAM</td>
</tr>
<tr>
<td>1 7 Naïve</td>
<td>482</td>
</tr>
<tr>
<td>1 7 Type-0</td>
<td>415</td>
</tr>
<tr>
<td>1 7 Type-1</td>
<td>381</td>
</tr>
<tr>
<td>1 7 Type-2</td>
<td>316</td>
</tr>
</tbody>
</table>

**Table**: HLS implementation results for a Mean Filter.
Discussion

for(int y = 0; y < IMAGE_HEIGHT; y++){
    for(int x = 0; x < IMAGE_WIDTH; x + v){
        (DataBeatType*)(out[y][x]) = local_op(stencil_p1(y, x), ..);
    }
}

Architectures

Loop Coarsening

- Schmid's
- Fetch and Calc (F&C)
- Calc and Pack (C&P)

Border Handling Types

- Naive
- Separated
- Type-0
- Type-1
- Type-2

Border Handling Modes

- Mirror
- Mirror-101
- Clamp
- Constant

Thanks for listening.

Any questions?

Title  Hardware Design and Analysis of Efficient Loop Coarsening and Border Handling for Image Processing

Speaker  M. Akif Özkan, akif.oezkan@fau.de
Backup Slides
A Deeper Look into Implementation Results
# Coarsening Architectures

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimation (CPtar = 20 ns)</th>
<th>Estimation (CPtar = 3.1 ns)</th>
<th>Implementation (CPtar = 3.1 ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v w/h Coars.</td>
<td>BRAM FF LUT CPes</td>
<td>BRAM FF LUT CPes</td>
<td>SLICE BRAM FF LUT DSP CPsyn CPimp</td>
</tr>
<tr>
<td>1 5 C&amp;P</td>
<td>4 304 93 13.50</td>
<td>4 378 93 3.10</td>
<td>152 4 600 270 29 2.48 2.55</td>
</tr>
<tr>
<td>1 5 F&amp;C</td>
<td>4 304 93 13.50</td>
<td>4 378 93 3.10</td>
<td>139 4 600 269 29 2.48 2.61</td>
</tr>
<tr>
<td>2 5 C&amp;P</td>
<td>4 339 86 14.43</td>
<td>4 446 88 3.09</td>
<td>215 4 873 448 14 2.52 2.63</td>
</tr>
<tr>
<td>2 5 F&amp;C</td>
<td>4 339 86 14.43</td>
<td>4 446 88 3.09</td>
<td>222 4 873 449 14 2.52 2.46</td>
</tr>
<tr>
<td>8 5 C&amp;P</td>
<td>8 663 82 14.43</td>
<td>8 954 84 3.06</td>
<td>675 8 2589 1565 6 2.39 2.56</td>
</tr>
<tr>
<td>8 5 F&amp;C</td>
<td>8 855 82 14.43</td>
<td>8 1146 84 3.06</td>
<td>603 8 2781 1566 6 2.39 2.74</td>
</tr>
<tr>
<td>32 5 C&amp;P</td>
<td>32 1995 75 14.43</td>
<td>32 3045 77 3.03</td>
<td>1951 32 9256 5367 6 2.38 2.83</td>
</tr>
<tr>
<td>32 5 F&amp;C</td>
<td>32 2955 75 14.43</td>
<td>32 4005 77 3.03</td>
<td>2023 32 10216 5367 6 2.38 3.09</td>
</tr>
</tbody>
</table>

Table: HLS estimation results for a local operator and Implementation results for a Mean Filter.
Border Handling Architectures

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimation (CPtar = 20 ns)</th>
<th>Estimation (CPtar = 3.1 ns)</th>
<th>Implementation (CPtar = 3.1 ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BRAM</td>
<td>FF</td>
<td>LUT</td>
</tr>
<tr>
<td>v w/h BH Patt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 5 Naïve</td>
<td>4</td>
<td>471</td>
<td>575</td>
</tr>
<tr>
<td>1 5 Type-0</td>
<td>4</td>
<td>438</td>
<td>381</td>
</tr>
<tr>
<td>1 5 Type-1</td>
<td>4</td>
<td>398</td>
<td>341</td>
</tr>
<tr>
<td>1 5 Type-2</td>
<td>4</td>
<td>403</td>
<td>459</td>
</tr>
<tr>
<td>2 5 Naïve</td>
<td>4</td>
<td>495</td>
<td>538</td>
</tr>
<tr>
<td>2 5 Type-0</td>
<td>4</td>
<td>432</td>
<td>344</td>
</tr>
<tr>
<td>2 5 Type-1</td>
<td>4</td>
<td>432</td>
<td>344</td>
</tr>
<tr>
<td>2 5 Type-2</td>
<td>4</td>
<td>435</td>
<td>501</td>
</tr>
<tr>
<td>1 7 Naïve</td>
<td>6</td>
<td>855</td>
<td>1443</td>
</tr>
<tr>
<td>1 7 Type-0</td>
<td>6</td>
<td>806</td>
<td>867</td>
</tr>
<tr>
<td>1 7 Type-1</td>
<td>6</td>
<td>693</td>
<td>642</td>
</tr>
<tr>
<td>1 7 Type-2</td>
<td>6</td>
<td>698</td>
<td>808</td>
</tr>
<tr>
<td>2 7 Naïve</td>
<td>6</td>
<td>957</td>
<td>1308</td>
</tr>
<tr>
<td>2 7 Type-0</td>
<td>6</td>
<td>862</td>
<td>730</td>
</tr>
<tr>
<td>2 7 Type-1</td>
<td>6</td>
<td>806</td>
<td>674</td>
</tr>
<tr>
<td>2 7 Type-2</td>
<td>6</td>
<td>923</td>
<td>1287</td>
</tr>
<tr>
<td>1 11 Naïve</td>
<td>10</td>
<td>2033</td>
<td>5390</td>
</tr>
<tr>
<td>1 11 Type-0</td>
<td>10</td>
<td>1952</td>
<td>2997</td>
</tr>
<tr>
<td>1 11 Type-1</td>
<td>10</td>
<td>1597</td>
<td>1583</td>
</tr>
<tr>
<td>1 11 Type-2</td>
<td>10</td>
<td>1601</td>
<td>1845</td>
</tr>
<tr>
<td>2 11 Naïve</td>
<td>10</td>
<td>2184</td>
<td>4562</td>
</tr>
<tr>
<td>2 11 Type-0</td>
<td>10</td>
<td>2025</td>
<td>2160</td>
</tr>
<tr>
<td>2 11 Type-1</td>
<td>10</td>
<td>1760</td>
<td>1719</td>
</tr>
<tr>
<td>2 11 Type-2</td>
<td>10</td>
<td>1989</td>
<td>3564</td>
</tr>
</tbody>
</table>

Table: HLS estimation results for a local operator and Implementation results for a Mean Filter.
Border Handling in case of Loop Coarsening (F&C)
Loop Coarsening: Architecture Selection

(a) F&C Type-1, which basically is $\min(r_w, v) = 2$ parallel Type-1 column selection for $w = 3$

(b) F&C Type-1 mirror border handling for $w = 9$ and $v = 2$, which basically is $\min(r_w, v) = 2$ parallel Type-1 column selection for $w = 5$. 
Loop Coarsening Overview
Loop Coarsening: Architecture Selection

Resource usage is the # of registers when border handling is ignored:

**Schmid’s:**
\[ k_{in} \cdot h \cdot (v + 2 \cdot (v \cdot \lceil \frac{r_w}{v} \rceil)) \]

**Fetch And Calc:**
\[ C_{F&C}^{reg} = k_{in} \cdot h \cdot (r_w + v \cdot (\lceil \frac{r_w}{v} \rceil + 1)) \]

**Calc and Pack:**
\[ C_{C&P}^{reg} = k_{in} \cdot h \cdot (2 \cdot r_w + v) + k_{out} \cdot (v - (r_w \mod v)) \]
Loop Coarsening: Architecture Selection

The region that C&P is better when border handling is ignored:

\[
C_{\text{reg}}^{\text{F&C- C&P}} > 0
\]

\[
C_{\text{reg}}^{\text{F&C- C&P}} = k_{\text{in}} \cdot h \cdot (r_w + v \cdot (\lceil r_w/v \rceil + 1)) - k_{\text{in}} \cdot h \cdot (2 \cdot r_w + v) + k_{\text{out}} \cdot (v - (r_w \mod v))
\]

\[
C_{\text{reg}}^{\text{F&C- C&P}} = (k_{\text{in}} \cdot h - k_{\text{out}}) \cdot ((r_w \mod v) - v)
\]

How significant the improvement is?

Coarsening of an algorithm \( v = 1024/32 = 32 \)

\( w = h = 5 \)

\[
C_{\text{reg}}^{\text{Schmid's}} = 32 \cdot 5 \cdot (32 + 2 \cdot (32 \cdot \lceil 5/32 \rceil)) = 15360 \text{bits}
\]

\[
C_{\text{reg}}^{\text{F&C}} = 32 \cdot 5 \cdot (2 + 32 \cdot (\lceil 5/32 \rceil + 1)) = 11040 \text{bits}
\]

\[
C_{\text{reg}}^{\text{C&P}} = 32 \cdot 5 \cdot (2 \cdot 2 + 32) + 32 \cdot (32 - 2) = 6720 \text{bits}
\]
Analyses of Border Handling
### Analysis of Border Handling: Temporal Locality

Temporal data flow for column data selection with a local operator of size $w = 7$. Blue background denotes valid image regions, $lX$ and $rX$ represent the appropriate pixel values for the corresponding border handling mode.

<table>
<thead>
<tr>
<th>$x = W - 4$:</th>
<th>$x = W - 3$:</th>
<th>$x = W - 2$:</th>
<th>$x = W - 1$:</th>
<th>$x = 0$:</th>
<th>$x = 1$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 18 19 20 21 22 23</td>
<td>18 19 20 21 22 23 $r0$</td>
<td>19 20 21 22 23 $r0$ $r1$</td>
<td>20 21 22 23 $r0$ $r1$ $r2$</td>
<td>$l0$ $l1$ $l2$ 0 1 2 3</td>
<td>$l1$ $l2$ 0 1 2 3 4</td>
</tr>
</tbody>
</table>
Analysis of Border Handling: Temporal Locality

Assuming that the streaming is not stalled and one pixel is read in each cycle, at least \( r_w \) pixels per row must be fetched at \( x = 0 \) in order be able to initialize all column pixels. Therefore, the minimum number of registers in a row is

\[
C_{\text{reg}}^{\text{min}} = h \cdot k_{\text{in}} \cdot (w + r_w)
\]
Analysis of Border Handling: Temporal Locality

Except at $x = 0$, border handling can be achieved only through data selection that appropriately feeds $R_{\text{fetch}}$ and shifts the content stored in $R_{\text{right}}$, $R_{\text{mid}}$ and $R_{\text{left}}$. 
Analysis of Border Handling: Temporal Locality

All registers, except $R_{fetch}$, should be able to read from $R'_{right}$ in order to initialize all column pixels at $x = 0$. 

M. Akif Özkan | Hardware/Software Co-Design | Efficient Loop Coarsening and Border Handling for Image Processing | ASAP'17
Analysis of Border Handling: Temporal Locality

There must be at least one MUX[2] before any register in $R_{\text{right}}$ and $R_{\text{left}}$. 
Analysis of Border Handling: Temporal Locality

Only the data selection before $R_{\text{fetch}}$ and blue portion of $R'_{\text{right}}$ can be optimized, when minimum number of registers is used in raster order processing.