Image Processing on the eXtreme Processing Platform®

Robert Strzodka
Numerical Analysis and Scientific Computing
University of Duisburg
http://www.numerik.math.uni-duisburg.de
strzodka@math.uni-duisburg.de
Overview

- Introduction
- Data-Flow & Architectures
- Implementations on the XPP
- Performance & Configurations
- Conclusions
Some Key-Tasks in Image Processing

• **Denoising**

• **Segmentation**

• **Matching**

• **Visualization**
Denoising by anisotropic diffusion
Segmentation by the level-set method
Matching by a gradient-flow method

original image

deformed image

matching error

matching result
Visualization of a vector field

Initial image

Step 1

Step 2

Step 3

Step 4

Step 5

Step 7

Step 10
Data-Flow in One Iteration

\[ F \left( \left( X^n_{\beta} \right)_{|\beta - \alpha| \leq C} \right) \]

\[
\sum_{\beta:|\beta - \alpha| \leq C} W_{\alpha, \beta - \alpha} X^n_{\beta}
\]
Data Processing-Bandwidth

\[
\text{total - bandwidth } b_{\text{total}} = \min \{ b_{\text{input}}, b_{\text{read}}, b_{\text{write}}, b_{\text{output}} \}
\]

**Task:** Maximize total - bandwidth \( b_{\text{total}} \)
Current SD and RD RAM-Types for MPUs

Bandwidth in GB/s
Latency (RAS cycle time) in 10ns
Data Processing-Bandwidth

\[
\text{total - bandwidth } b_{\text{total}} = \min \{ b_{\text{input}}, b_{\text{read}}, b_{\text{write}}, b_{\text{output}} \}
\]

**Tasks:**
1. Keep the whole pipeline busy
2. Maximize total - bandwidth \( b_{\text{total}} \)
## Comparison XPP-FPGA

<table>
<thead>
<tr>
<th>XPP</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ <strong>fast reconfigurability</strong></td>
<td>+ <strong>low level optimization</strong></td>
</tr>
<tr>
<td>+ implicit synchronization</td>
<td>+ large local memory with variable access</td>
</tr>
<tr>
<td>+ higher level programming</td>
<td>+ many IO channels</td>
</tr>
<tr>
<td>+ easier debugging</td>
<td></td>
</tr>
</tbody>
</table>
Overview

• Introduction
• Data-Flow & Architectures
  • Implementations on the XPP
  • Performance & Configurations

• Conclusions
Implementations on the XPP

**Aim:** Maximal total-bandwidth, i.e.: one input and output in each clock cycle

\[ F\left(\left(X^n_\beta\right)_{|\beta-\alpha|\leq C}\right) \]

- fifo caches for neighbor pixels
- local computation
- dual ported access to exram
- address generator
Boundary Conditions

constant
boundary condition

natural
boundary condition
Data Travers

If local memory is too small to cache all neighbour pixels, traverse the data in smaller subvolumes.

Costs: multiple transfer of border elements.
Implemented Filters in 2D and 3D in a 8+4x8 XPP array

\[ X_{\alpha}^{n+1} = F(\{X_{\beta}^{n}\}_{|\beta-\alpha|\leq C}) = \sum_{\beta:|\beta-\alpha|\leq C} W_{\alpha,\beta-\alpha} X_{\beta}^{n} \]

<table>
<thead>
<tr>
<th>2D Stencil</th>
<th>3D Stencil</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x3</td>
<td>3x3x3</td>
</tr>
<tr>
<td>5x5</td>
<td>5x5x5</td>
</tr>
<tr>
<td>7x7</td>
<td>in array</td>
</tr>
<tr>
<td></td>
<td>10+4x15</td>
</tr>
</tbody>
</table>
## Performance

<table>
<thead>
<tr>
<th></th>
<th>stencil 7x7 XPP 12x8</th>
<th>stencil 3x3x3 XPP 12x8</th>
<th>stencil 5x5x5 XPP 14x15</th>
</tr>
</thead>
<tbody>
<tr>
<td>operations per clock cycle</td>
<td>49 MAC</td>
<td>27 MAC</td>
<td>125 MAC</td>
</tr>
<tr>
<td>output pixel per clock cycle</td>
<td>1</td>
<td>0.73</td>
<td>0.58</td>
</tr>
<tr>
<td>number of passes at 100MHz</td>
<td>256^2 data 1525</td>
<td>256^3 data 4.373</td>
<td>256^3 data 3.454</td>
</tr>
</tbody>
</table>

⇒ **real-time** for 2d applications
⇒ **interactivity** for 3d applications
for each timestep n {
  \textbf{configure} the array for weight computation
  compute weights for each $|\gamma| \leq C$
  \[ W_{\alpha,\gamma}^n = G_{\gamma} \left( (X^n_{\beta})_{|\beta-\alpha| \leq C} \right) \]

  \textbf{configure} the array for data computation
  apply weights to data
  \[ X^{n+1}_{\alpha} = \sum_{\beta:|\beta-\alpha| \leq C} W_{\alpha,\beta-\alpha}^n X^n_\beta \]
}
Configuration for an implicit solver

for each timestep \( n \) {

configure the array for weight computation

compute weights for each \( \gamma \leq C \)

\[
W_{\alpha,\gamma}^n = G_{\gamma} \left( (X_{\beta}^n)_{|\beta - \alpha| \leq C} \right)
\]

configure the array for data computation

for each iteration \( k \) {

apply weights to data

\[
X_{\alpha, k+1}^{n+1} = \sum_{\beta : |\beta - \alpha| \leq C} W_{\alpha,\beta}^n X_{\beta}^{n+1,k}
\]

}

}
Solving the weight transmission problem

1. Instead of pre-computing 27 weights $W_{\alpha,\gamma}$, $|\gamma| \leq 1$ for a 3x3x3 stencil, pre-compute only a smaller vector of intermediate results $\tilde{w}_{\alpha}$ from which all the weights can be quickly evaluated $W_{\alpha,\gamma}(\tilde{w}_{\alpha})$.

2. Increase the number of available IO channels by shifting the task of address generation and memory access to a processor outside of the XPP array, such that all the available 8 IO channels can be used for data input or output.

3. Increase the overall number of IO channels, such that applications will be able to access more than 6 intermediate results simultaneously.
Solving the weight transmission problem

**Aim**: Maximal total bandwidth, i.e.: one input and output in each clock cycle

\[
\begin{align*}
&\text{fifo} \quad \text{caches for neighbor pixels} \\
&\text{local computation} \quad F\left(\left(\begin{array}{c}
X^n_eta \\
W_{\alpha,\gamma}
\end{array}\right)_{|\alpha|,|\gamma| \leq C}\right) \\
&\text{address generator} \\
&\text{dual ported access to exram} \quad \text{exram}
\end{align*}
\]
Solving the weight transmission problem

Aim: Maximal total-bandwidth, i.e.: one input and output in each clock cycle

\[
F\left(\left(\begin{array}{c}
X^n \\
\bar{w}_\alpha \\
\end{array}\right)_{|\beta-\alpha|\leq C}\right)
\]
Overview

• Introduction
• Data-Flow & Architectures
• Implementations on the XPP
• Performance & Configurations

• Conclusions
Conclusions

• A wide range of image processing application could be accelerated.

• The test implementations at estimated 100MHz suggest a performance gain of 10-20 over common PC solutions in full-grown applications.

• In our experience the XPP wins over other architectures such as GPUs or FPGAs either in speed or programmability.

• Finally, improved memory availability and IO access would further facilitate and accelerate image processing on the XPP.