Enabling FPGAs for the Masses

Janarbek Matai\textsuperscript{1}, Dustin Richmond\textsuperscript{1}, Dajung Lee\textsuperscript{2} and Ryan Kastner\textsuperscript{1}

\textsuperscript{1} Computer Science and Engineering

\textsuperscript{2} Electrical and Computer Engineering

University of California, San Diego
Motivation

Various Application Domains

- Image Processing
- Cloud Computing
- Computer Vision
- Machine Learning
- Wireless Comm.

Difficulties

Hardware Design Domain

- High performance
- Energy efficiency

Field Programmable Gate Array (FPGA)
Motivation

Various Application Domains

- Image Processing
- Cloud Computing
- Computer Vision
- Machine Learning
- Wireless Comm.

High Level Synthesis Tools
- (e.g., Vivado HLS, OpenCL, etc.)

Hardware Design Domain

Field Programmable Gate Array (FPGA)

✓ High performance
✓ Energy efficiency
Motivation

Various Application Domains
- Image Processing
- Cloud Computing
- Computer Vision
- Machine Learning
- Wireless Comm.

Difficulties

High Level Synthesis Tools

Hardware Design Domain
- High performance
- Energy efficiency

Field Programmable Gate Array (FPGA)
Convolution: Software

```cpp
for(int i = 0; i < rows; i++){
    for(int j=0; j < cols; j++){
        Gx = 0;
        Gy = 0;
        for(rowOffset = -1; rowOffset <= 1; rowOffset++){
            for(colOffset = -1; colOffset <= 1; colOffset++){
                Gx = Gx + ...;
                Gy = Gy + ...;
                G = ...;
            }
        }
    }
}
```

Based on Xilinx tutorial: “Zynq all programmable soc sobel filter implementation using the vivado hls tool.”
Based on Xilinx tutorial: “Zynq all programmable soc sobel filter implementation using the vivado hls tool.”
Regular programs: Software vs. Hardware

```
for(int i=0; i<rows; i++)
for(int j=0; j<cols; j++)
#
#pragma pipeline
LineBuffer[0][j]=LineBuffer[1][j];
LineBuffer[1][j]=LineBuffer[2][j];
LineBuffer[2][j]=input_image[i][j];
WindowBuffer[0][0] = LineBuffer[0][j];
WindowBuffer[1][0] = LineBuffer[1][j];
WindowBuffer[2][0] = LineBuffer[2][j];
for(int k = 0; k < 3; k++) {
WindowBuffer[k][2] = WindowBuffer[k][1];
WindowBuffer[k][1] = WindowBuffer[k][0];
}
sobel_filter(WindowBuffer);
```

Clock cycles: SW code vs. Restructured code

<table>
<thead>
<tr>
<th>Clock cycles (log scale)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW</td>
</tr>
<tr>
<td>10000000</td>
</tr>
<tr>
<td>1000000</td>
</tr>
<tr>
<td>10000</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restructured code</th>
</tr>
</thead>
<tbody>
<tr>
<td>67X</td>
</tr>
</tbody>
</table>

Regular programs: Software vs. Hardware
Problems

1. Regular programs (e.g., $C[i] = A[i]*B[i]$)
   - Efficient hardware architecture
     e.g., Polyhedral transformation, [W. Zuo et al, FPGA’13]

2. Irregular programs (e.g., $A[B[i]]$, unbounded loops)
   - Problems with pipeline/unroll
     e.g., Huffman tree creation
Huffman Tree Creation: Software

<table>
<thead>
<tr>
<th>S</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
</tbody>
</table>

Create Tree

- 7
  - A
  - 4
    - D
    - E
  - 17
    - 0
      - F
      - B
    - 1
      - 0
      - C
      - 1

Sort
Huffman Tree Creation: Software

```
while()
{
    [pipeline/unroll]
    Left = ExtractMin(List[0])
    Right = ExtractMin(List[1])
    NewNode = CreateNewNode (Left->F, Right->F)
    HuffmanTree->CurrentLeft = Left;
    HuffmanTree -> CurrentRight = Right;
    InsertToListSortedOrder(NewNode);
}
```

```
InsertToListSortedOrder(NewNode) {
    while(findLocation) {
        ...
    }
}
```

Need to change the code to create an efficient hardware architecture
Huffman Tree Creation: Hardware
Huffman Tree Creation: Hardware
Huffman Tree Creation: Hardware

```plaintext
while(No element left in SF){
    pipeline
    //Selecting left child
    if(SF.F < IN.F){
        Left = SF.S
    }
    else {
        Left = IN.F
    }
}
while(No element left in IN){
    pipeline
    //If there are any elements left in IN
    Left = IN[currentPointer]
    Right = IN[currentPointer +1]
}
```
Irregular programs: Software vs. Hardware

```
while(){
    pipeline/unroll
    Left = ExtractMin(List[0]);
    Right = ExtractMin(List[1]);
    NewNode = CreateNewNode(Left->F, Right->F);
    HuffmanTree->CurrentLeft = Left;
    HuffmanTree->CurrentRight = Right;
    InsertToListSortedOrder(NewNode);
}
InsertToListSortedOrder(NewNode) {
    while(findLocation) {
        ...
    }
}
```

Clock cycles: SW code vs. Restructured code

- **Software C code**
- **Restructured C code**

**SW** vs. **Restructured code**

- **2500X**
Applications designed with HLS tools

• *Face recognition system*, [Matai *et al*, FCCM’11]
• *Wireless Channel Emulator*, [Matai *et al*, FPT’12]
• *Cell sorting*, [Lee *et al*, FPL’13]
• *Canonical Huffman Encoding*, [Matai *et al*, ASAP’14]
• *Wireless Channel Tracker*, [Lee *et al*, FPL’14]

*Meet their performance requirement in a short design time.*
Challenges: For more non-experts

- HLS gives good QoR $\rightarrow$ Restructured code
Challenges: For more non-experts

- HLS gives good QoR → Restructured code
- Restructured code = Domain knowledge + Hardware design skill
Challenges: For more non-experts

- HLS gives good QoR → Restructured code
- Restructured code = Domain knowledge + Hardware design skill
- Challenges
  - Generating Restructured Code
  - Designing large and complex applications
  - Building an End-to-End system

**Diagram:**

1. Application Code
2. Domain Specific Language (DSL)
3. DSL Compiler
4. Restructured Code
5. HLS Tools
6. FPGA

A Suggested Tool Flow
Generating Restructured Code

- Domain Specific Language (DSL) High Level Synthesis (HLS) Template
Generating Restructured Code

1. Domain Specific Language (DSL)

2. DSL Compiler

3. Restructured Code

4. HLS Tools

Application Code

Computer Vision
Linear Algebra
Machine Learning

DSL HLS Templates

FPGA
Canny Edge Detection

Gaussian Smoothing

Edge Strength (Sobel)

Non Maxima Suppression

Double Thresholding

for (int i=0; i<IMG_H; i++){
    for (int j=0; j<IMG_W; j++){
        #pragma pipeline II=1
        ...
        ...
        ...
        sobel_filter(...);
    }
}

gaussian(unsigned char  window[M][M]){
    ...
    ...
    ...
}

for (int i=0; i<IMG_H; i++){
    for (int j=0; j<IMG_W; j++){
        #pragma pipeline II=1
        ...
        ...
        ...
        sobel_filter(unsighned char  window[M][M]){
        ...
        ...
        ...
    }
}

for (int i=0; i<IMG_H; i++){
    for (int j=0; j<IMG_W; j++){
        #pragma pipeline II=1
        ...
        ...
        ...
        nms(...);
    }
}

gaussian(unsigned char  window[M][M]){
    ...
    ...
    ...
}

for (int i=0; i<IMG_H; i++){
    for (int j=0; j<IMG_W; j++){
        #pragma pipeline II=1
        ...
        ...
        ...
        nms(unsighned char  window[M][M]){
        ...
        ...
        ...
    }
}

double_thr(...);

gaussian(unsigned char  window[M][M]){
    ...
    ...
    ...
}

double_thr(unsigned char  window[M][M]){
    ...
    ...
    ...
}

Gaussian Smoothing

Edge Strength (Sobel)

Non Maxima Suppression

Double Thresholding
Tool flow: Canny edge detection

1. Domain Specific Language (DSL)
   - **gaussian**
     ```
     gaussian(unsigned char window[N][N]){
         ....
     }
     ```
   - **sobel_filter**
     ```
     sobel_filter(unsigned char window[M][M]){
         ....
     }
     ```
   - **double_thr**
     ```
     double_thr(unsigned char window[K][K]){
         ....
     }
     ```
   - **nms**
     ```
     nms(unsigned char window[L][L]){
         ....
     }
     ```
   - **canny**
     ```
     canny(in1[SIZE], out3[SIZE]){
         #pragma Streaming Bulk Syncronous Model
         Convolution(in1, out1, N, *gaussian)
         Convolution(out1, out2, M, *sobel_filter);
         Convolution(out2, out3, K, *double_thr);
         ConvolutionHistogram (out2, out3, L, *nms);
     }
     ```

2. DSL Compiler
3. Restructured Code
4. HLS Tools

**Pattern Description:** How kernels communicate
→ Inspired by common parallel programming patterns (e.g., Fork/Join)
Our Approach

1. Domain Specific Language (DSL)
2. DSL Compiler
3. Restructured Code
4. HLS Tools

Application Code

Computer Vision
Linear Algebra
Machine Learning

DSL HLS Templates

Fork/Join
Streaming Bulk Synchronous Reduction
Partition
Scan
Embarrassingly Parallel

Pattern Descriptions

FPGA
Building an End-to-End System

1. Domain Specific Language (DSL)
2. DSL Compiler
3. Restructured Code
4. HLS Tools

DSL HLS Templates

Pattern Descriptions

Application Code

Host PC

FPGA

RIFFA

Computer Vision
Linear Algebra
Machine Learning

Fork/Join
Streaming Bulk Synchronous Reduction
Partition
Scan
Embarrassingly Parallel

Partition
Scan
Embarrassingly Parallel

Partition
Scan
Embarrassingly Parallel

Computer Vision
Linear Algebra
Machine Learning

Fork/Join
Streaming Bulk Synchronous Reduction
Partition
Scan
Embarrassingly Parallel
Conclusion and Future Work

• HLS tools generate an efficient hardware if right input code is given

• Future work to increase HLS accessibility: (Our solution)
  • Generating restructured code automatically
    • *Domain Specific HLS templates*
  • Designing large and complex applications
    • *Parallel Programming Patterns (Fork/Join, Streaming Bulk Sync Model)*
  • Building an end-to-end system
    • *Integrating RIFFA with our tool flow*
Backup slides