Real-time Dense Passive Stereo Vision: A Case Study in Optimizing Computer Vision Applications Using OpenCL™ on ARM®

Gian Marco Iodice, ARM Ltd
Media Processing Group, Cambridge (UK)
gianmarco.iodice@arm.com

11th May 2015, Santa Clara, CA USA – Embedded Vision Summit - «Enabling Computer Vision on ARM»
Agenda

- Stereo Vision Overview
- Implementation
- OpenCL Optimizations
- Conclusion and future works
Stereo Vision Overview
What is Dense Passive Stereo Vision?

**Stereo Vision** is a visual sensing technique aimed at inferring **depth** by comparing two views of the same scene.
Fields of application

- INDUSTRIAL
- ENTERTAINMENT
- AUTOMOTIVE
- ROBOTICS
- AUGMENTED REALITY
- MEDICAL

[Diagram showing overlapping circles labeled entertainment, medical, robotics, industrial, and reality, indicating areas of application.]
How does it work? (1)

- Assuming...
  - Cameras optically identical
    - same image sensor
    - same focal length
  - Cameras horizontally aligned
  - Images rectified
    - no lens distortion
  - Images captured at the same instant

we can talk about.... **Horizontal Epipolar Line Constraint**

- **Disparity**: it is the difference in x coordinates (\(d = x_L - x_R\)) of the corresponding pixel in the left and right images
How does it work? (2)

Depth from disparity via triangulation

\[ Z = \frac{b \cdot f}{d \cdot px_size} \]

- \( Z \): distance (in meters) between the cameras and point P
- \( b \): baseline
- \( f \): focal length
- \( px\_size \): size of the pixel on the image sensor
- \( d \): disparity
Disparity Map and Point Cloud

3D back-projection (point cloud)

Point Cloud rendered with MeshLab
**Correspondence Problem**

**Template matching**

- For each pixel in the left image:
  - Extract $N \times N$ block around it (Reference Template).
  - Compare the reference template with all blocks in the search space of right image using a similarity measure (i.e. SAD, SSD, SHD,…).
  - The disparity of each pixel is simply selected by the WTA strategy (Winner-Takes-All). Best match wins.
Implementation
Recipe

- Grayscale images

- Multi-Resolution strategy *(aka coarse-to-fine strategy)*

- Modified Census Transform (MCT) 9x9 and 7x7
  - 10 bytes per pixel for MCT 9x9
  - 6 bytes per pixel for MCT 7x7
Pipeline

Level 2
- Modified Census Transform
- Stereo Matching (SHD)
- Disparity Refinement
- Up-scale 2x

Level 1
- Modified Census Transform
- Stereo Matching (SHD)
- Disparity Refinement
- Up-scale 2x

Level 0
- Modified Census Transform
- Stereo Matching (SHD)
- Disparity Refinement
Census Transform — Ramin Zabih Et al., 1994

- It is a non-parametric local image transform which result does not depend on camera gain and light condition.

- It replaces each pixel by a N-bit string which summarizes the local spatial structure.

- For each neighboring pixel (except the center one) it is associated one bit of that N bit string.
  - Each bit is set if the corresponding neighboring pixel value is greater than the center pixel value.

Ex. Census Transform 3x3

<table>
<thead>
<tr>
<th>Pixel Value</th>
<th>Bit</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>115</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Modified Census Transform — Bernhard Froba Et al., 2004

- Extension of the work did by Bernhard Froba Et al. in 2004

- Instead comparing the neighboring pixels with the center pixel, it compares the values of the neighboring pixel with the mean intensity value of the local window 3x3 centered on it

Ex. Modified Census Transform 7x7

<table>
<thead>
<tr>
<th>115</th>
<th>33</th>
<th>40</th>
<th>102</th>
<th>67</th>
<th>170</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
<td>61</td>
<td>40</td>
<td>12</td>
<td>47</td>
<td>12</td>
<td>67</td>
</tr>
<tr>
<td>118</td>
<td>63</td>
<td>42</td>
<td>14</td>
<td>47</td>
<td>19</td>
<td>82</td>
</tr>
<tr>
<td>110</td>
<td>66</td>
<td>40</td>
<td>10</td>
<td>52</td>
<td>17</td>
<td>83</td>
</tr>
<tr>
<td>115</td>
<td>68</td>
<td>41</td>
<td>10</td>
<td>54</td>
<td>18</td>
<td>82</td>
</tr>
<tr>
<td>111</td>
<td>72</td>
<td>43</td>
<td>8</td>
<td>57</td>
<td>17</td>
<td>81</td>
</tr>
<tr>
<td>110</td>
<td>70</td>
<td>40</td>
<td>10</td>
<td>59</td>
<td>21</td>
<td>80</td>
</tr>
</tbody>
</table>

Source image (grayscale)

Box Filter 3x3

Modified Census Transform

Destination image

\[(42 + 14 + 47 + 40 + 10 + 52 + 41 + 10 + 54) / 9 = 34.44\]
OpenCL Optimizations
GPU compute on Mali™

- Full profile OpenCL v1.1 in hardware for Mali-T600 / T700 GPUs
  - Backward compatibility support for OpenCL v1.0.
  - Image types supported in HW and driver.
  - printf implemented as an extension to v1.1 driver.

- Mali-T600 and T700 series GPUs have a SIMD instructions
  - Mali-T600 / T700 can natively support all CL data types.
  - Images data support.
  - Integers and floating point are supported equally quickly.
General advices (1)

- All CL memory buffers are allocated in global memory that is *physically accessible* by both CPU and GPU cores
  - However, only memory that is allocated by `clCreateBuffer()` is mapped into both the CPU and GPU virtual memory spaces.
  - Memory allocated using `malloc()`, etc, is only mapped onto the CPU.
  - So calling `clCreateBuffer()` with `CL_MEM_USE_HOST_PTR` and passing in a user created buffer requires the driver to create a new buffer and copy the data (identical to `CL_MEM_COPY_HOST_PTR`).

Buffers created by `malloc()` are not mapped into the GPU memory space

`clCreateBuffer(CL_MEM_ALLOC_HOST_PTR)` creates buffer visible by both GPU and CPU
General advices (2)

- **Try to use as much as possible vector instructions**
  - Avoid writing kernels that operate on single bytes or scalar values.
  - It can allow to execute less threads
  - It can allow to reduce the # of load/store instructions

- **Use cl built-in functions (in short cl BIF) when possible**
  - **Math cl BIFL**: cos(), sin(), atan(), log, pow,…
  - **Geometric cl BIFL**: dot(), normalize,…

- **Use correct data types for your specific problem**
  - e.g. uint16?, ushort16?, uchar16?..

Further details and general advices at [malideveloper.arm.com](http://malideveloper.arm.com) where you can find tutorials, videos and developer guides:

- [OpenCL SDK tutorial](http://malideveloper.arm.com)
- [RenderScript™ tutorial](http://malideveloper.arm.com)
Further optimizations…

- **Data layout**
  - *Modified Census Transform: case study*

- **No serialization of CPU and GPU workloads**
  - *Stereo vision pipeline*

- **Parallel tasks with a single kernel**
  - *Stereo vision pipeline*

- **Complex arithmetic expressions instead of look-up tables**
  - *Popcount: case study*

- **Avoiding branches with Padding Bytes and cl BIF**
  - *Fill Occluded Nearest Lower Pixel: case study*
Data layout (1)
Modified Census Transform: case study

- How we store data has a significant impact on the performance of single kernel and the whole pipeline.
  - Interleaved data generally requires more load/store instructions

- Sometimes it makes other stages easily vectorizable…
Data layout (2)
Modified Census Transform: case study

- Using planar data layout:
  - Performance of Modified Census Transform are improved by a factor $1.4x$ due by:
    - Reduced # of store operations
    - Reduced # of arithmetic instructions for the swizzling
  - It makes the stereo matching stage easily vectorizable

```
uchar16 ref0 = vload16(addrLeft);
uchar16 ref1 = vload16(addrLeft + offset2ndImg);
uchar16 ref2 = vload16(addrLeft + 2*offset2ndImg);
uchar16 cost;
...
for(i = 0; i < maxDisparity; i++)
  target0 = vload16(addrRight + i);
  target1 = vload16(addrRight + offset2ndImg + i);
  target2 = vload16(addrRight + 2*offset2ndImg + i);
  cost = shd(ref0, target0);
  cost += shd(ref1, target1);
  cost += shd(ref2, target2);
endfor
```
No serialization of CPU and GPU workloads (1)

- Avoid the **serialization of CPU and GPU workloads** in order to hide the driver overhead
  - Keep the GPU busy while you’re enqueuing the kernels
  - Particularly important when there are several CL kernels to enqueue

```
CPU  | Enqueue Job1 | Enqueue Job2 | Enqueue Job3 | Enqueue Job4 | Enqueue Job5 | Enqueue Job6
GPU  | Execute Job1 | Execute Job2 | Execute Job3 | Execute Job4 | Execute Job5 | Execute Job6
```

```
GPU  | Execute Job1 | Execute Job2 | Execute Job3 | Execute Job4 | Execute Job5 | Execute Job6
CPU  | Enqueue Job1 | Enqueue Job2 | Enqueue Job3 | Enqueue Job4 | Enqueue Job5 | Enqueue Job6
```

**enqueue Frame 0**
wait for Frame 0 to complete…
**enqueue Frame 1**
wait for Frame 1 to complete…
e tc.

**enqueue Frame 0**
enqueue Frame 1
wait for Frame 0 to complete…
enqueue Frame 2
wait for Frame 1 to complete…
e tc.
Parallel tasks with a single kernel (1)

- Some kernels could be executed in parallel

- The algorithm has **2 independent flows**: each one for computing respectively the left and right disparity map.
Parallel tasks with a single kernel (2)

- Assuming that both left and right images have same resolutions and same kernels parameters, we can use a single kernel and the 3rd dimension of gws (global work-group size) for accessing the right element

```c
const int addr = x + y * stride + z * offset2ndImage;
```

- It allows to reach the maximum GPU utilization at lower resolution where otherwise few threads would be dispatched per kernel
Complex expressions instead of look-up tables (1)

Popcount: case study

- The similarity measure used by the Stereo Matching stage is the Sum of Hamming Distance (SHD).

```
xor + popcount.
```

16-bit string 1: 1 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1
16-bit string 2: 1 1 1 0 0 1 1 1 0 0 0 1 0 0 1 1

```
1 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1
1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 1
```

XOR

```
0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0
```

**STAGE 1**

```
```

**STAGE 2**

```
```

SHD = 3

popcount - count the number of '1' bits
Complex expressions instead of look-up tables (2)

Popcount: case study

- **Look-up table**
  - Only scalar memory access
  - Few arithmetic instructions

- **Arithmetic parallel algorithm (Divide et Impera)**
  - No memory access
  - The # of arithmetic instructions are much more but...this approach is ~3\times faster than the look-up table using vector operations

```c
Scalar
input cost;
const byte55 = 0x55;
const byte33 = 0x33;
const byte0f = 0x0f;

const byte unchar = (byte)0x100;

input cost;
cost = (cost & unchar)0x55) + (cost >> 1 & unchar)0x55);
cost = (cost & unchar)0x33) + (cost >> 2 & unchar)0x33);
cost = (cost >> 4 & cost) & unchar)0x0f;
return cost;
```

```c
Vector
input cost;
const byte16 = (byte)0x100;

input cost;
cost = (cost & unchar16)0x55) + (cost >> 1 & unchar16)0x55);
cost = (cost & unchar16)0x33) + (cost >> 2 & unchar16)0x33);
cost = (cost >> 4 & cost) & unchar16)0x0f;
return cost
```
Avoiding Branches with Padding Bytes and cl BIF (1)

“An algorithm with many conditionals is likely not to be optimal” so try to **avoid as much as possible loops and if/else conditions:**

- **Padding bytes:** can be used for avoiding the boundary check.

![Diagram showing padding bytes](image)

- **cl BIF:** OpenCL provides **relational built-in functions** that can be used for avoiding branches
  - `select(a, b, condition):`  
    condition? b : a
  - `all(x):`  
    It returns 1 if MSB in all components of x are set
  - `clamp(x, a, b):`  
    Clamp x in the interval defined [a, b]
  - `any(x):`  
    It returns 1 if any component of x is set
  - ...
Avoiding Branches with Padding Bytes and cl BIF (2)
Fill occluded pixel nearest lower: case study

- In the _disparity refinement stage_, the invalidated disparity is replaced with the **nearest valid lower disparity** on the same scanline.
Avoiding Branches with Padding Bytes and cl BIF (3)

Fill occluded pixel nearest lower: case study

while(boundary_condition) {
    if(dispLeft == 0)
        dispLeft = *(ptrDispSrc + k - i);
    if(dispRight == 0)
        dispRight = *(ptrDispSrc + k + i);
    if(dispLeft !=0 && dispRight != 0)
        break;
    i++;
    // Boundary condition
    ....
}  // Select the lower disparity

dstDisp = dispLeft < dispRight? dispLeft : dispRight;

while(!check && boundary_condition) {
    loadLeft = vload16(dispSrc - i);
    loadRight = vload16(dispSrc + i);
    dispLeft = select(dispLeft, loadLeft, dispLeft == 0);
    dispRight = select(dispRight, loadRight, dispRight == 0);
    check = all( dispLeft!=0 && dispRight != 0);
    i++;
    // Boundary condition
    ....
}  // Select the lower disparity

dstDisp = select(dispRight, dispLeft, dispLeft < dispRight);

51x faster
Conclusion and future works
Results (1)

- The implemented algorithm:
  - is easy to parameterize
  - is configurable in terms of disparity range
  - computes disparity for occluded pixels
  - offers good reliability throughout a wide variety of scene and illumination conditions.

- The system was speed up on development platform featuring an ARM Mali GPU:
  - ~120 fps with 60 disparity levels at 320x240
  - ~52 fps with 60 disparity levels at 640x480

- Moreover good performance are obtained as well without using of coarse-to-fine strategy.
  - ~49 fps with 60 disparity levels at 320x240
Results (2)

Dataset from vision.middlebury.edu/stereo/

Coarse-to-fine

NO Coarse-to-fine
Future works

- **Use of Sparse Modified Census Transform 7x7**
  - It allows to reduce the # of load/store and arithmetic instructions
  - More erroneous disparity

- **Improve accuracy of Disparity Refinement stage**
  - Median Filter
  - Weighted Median Filter
  - Sub-Pixel estimation
Final considerations…

- Results reached by GPU compute on ARM Mali are definitely promising for stereo vision applications demonstrating the feasibility to achieve real-time performance on Mobile ARM GPU

- Small changes in OpenCL code can lead to reach big performance
  - e.g. data layout, correct data type,…

- Well optimized data layout, types, etc can help reduce the size of kernels (KISS approach)
  - It may reduce the number of registers each kernel needs allowing more work items to run on the GPU at the same time (e.g. stereo matching stage)
Before finishing...

- This project was developed with a joint cooperation between **ARM Ltd - Media Processing Group, Cambridge – UK** and the Dept. of Information Engineering of the **University of Pisa - Italy**.

- **Gian Marco Iodice**, ARM Ltd – Media Processing Group, Cambridge (UK)
- **Anthony Barbier**, ARM Ltd – Media Processing Group, Cambridge (UK)
- **Prof. Roberto Saletti**, University of Pisa – Dept. of Information Engineering (Italy)
Question time
References

- malideveloper.arm.com
- vision.middlebury.edu/stereo/


Thanks