Fractal Video Compression in OpenCL: An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platforms

Doris Chen, Deshanand Singh
Jan 24th, 2013
Platform Evaluation Challenges

- **Conducting a fair platform evaluation is difficult**
  - Platforms have different programming models
    - CPUs – C/C++
    - GPU - vendor-specific languages (CUDA)
    - FPGAs – RTL languages (Verilog, VHDL)

- **Hard to predict behaviour ahead of time without actual implementation**

- **Designers often select the platform based on device specs**
  - Need to consider the actual functionality needed
OpenCL Standard

- **A platform-independent standard**
  - Target Host + Accelerator applications

- **Data parallelism is explicitly specified**
  - **Host** – manages data and control flow
  - **Kernel** – highly parallel section to be accelerated (multi-core CPU, GPU, or FPGA)

- **Same code can be easily targeted to different platforms for performance comparison**

```c
__kernel void sum(__global float *a,
                 __global float *b,
                 __global float *y) {  
    int gid = get_global_id(0);
    y[gid] = a[gid] + b[gid];
}
```

```c
main() {
    read_data( ... );
    manipulate( ... );
    clEnqueueWriteBuffer( ... );
    clEnqueueNDRange(...,sum,...);
    clEnqueueReadBuffer( ... );
    display_result( ... );
}
```
Objective

- A programming model study using OpenCL
- Implement fractal compression in OpenCL
  - An video encoding algorithm
- Code is ported to and optimized for multi-core CPUs, GPUs and FPGAs
  - Introduce Altera’s OpenCL-to-FPGA Compiler
- Compare performance between multi-core CPUs, GPUs, and FPGAs
Based on the theory of iterated function systems (IFS)

Consider a pixel value of 4

- Can be expressed as the recurrence relation:
  \[ x \leftarrow x \times 0.5 + 2 \]

- Can be shown that the relation resolves to 4 regardless of initial value

We can represent the pixel with 2 values \{0.5, 2\}
Image Compression

- Express a N x N image as a 1-D vector
  \[ P = \{I(0, 0), I(0, 1), \ldots, I(N-1, N-1)\} \]

- Find transformation matrix A, and vector C such that
  \[ P \leftarrow AP + C \]

- We would recursively apply this to arrive at the image
  - Do not need the original image
  - Regenerate the image from random data with A and C

- Not quite useful when performing image compression!
  - Instead of \(N^2\) pixel values, have a \(N^2 \times N^2\) matrix and a \(N^2\) vector!!
Fractal Image Compression

- Creation of a codebook
- Take 8x8 regions of the image, and compress into 4x4
  - Each 4x4 pixel is an average of a 2x2 region
- Generates a library of codebook images
Using the Codebook

- Codebook library help generate matrix A and vector C
- For each region $R_i$, find the codebook entry that best minimizes error
  - After applying a scale factor $s_i$, and an offset $o_i$
- Compute the summation of absolute differences (SAD)
  - Compare $R_i$ with all $D_j$ in the codebook

\[
\min_{D_j \in \text{Codebook}} \min_{s_i, o_i} \left\| R_i - (s_i * D_j + o_i) \right\|
\]
After finding the best approximation for each $R_i$, we may have a set of equations such as the following:

$$R_0 \approx 0.25 \times D_{58} + 107$$

$$R_1 \approx 0.75 \times D_{1023} + 15$$

$$\vdots$$

$$R_{4095} \approx 1.1 \times D_{99} + 68$$

$$P \approx \begin{bmatrix} 0.25 \times TD_{58} \\ 0.75 \times TD_{1023} \\ \vdots \\ 1.1 \times TD_{99} \end{bmatrix}$$
Fractal Code

- Each $R_i$ can be expressed as:
  \[ R_i = s_i \times D_j + o_i \]

- Transmit 3 values for each 4x4 region
  - Best matching codebook entry $j$, scale factor $s_i$, and offset $o_i$

- This forms the fractal code for the image
  - Compression achieved due to transmission of a small number of coefficients
Image Encoding / Decoding

Encoding

1. Read Image Data
2. Process to Frames
3. Create Codebook
4. SAD-based Search
5. Generate Fractal Codes

Decoding

1. Start with blank Image
2. Create Codebook
3. Apply Fractal Codes
4. Converged?
   - No
   - Yes

*convergence typically occurs within 4-5 iterations*
Decoding Example

Original Image

Iteration 2
SNR = 29.0003 dB
Fractal Video Encoding General Approach

- **Encode new frames using the codebook of the original frame**
- **Cross-coding experiment showed average PSNR loss of only ~1dB**
  - Use the codebook of one image to encode another
  - Sequential frames are generally similar in nature (less diverse than random images)

Encode $f_1$ using $c_1$

Send fractal code

Regenerate $f_1$ from code ($f_1'$)

Generate codebook $c_1'$ from $f_1'$

Regenerate $f_2$ from code using $c_1'$ → $f_2'$

Generate codebook $c_1$ from frame $f_1$

Encode $f_1$ using $c_1$

Encode $f_2$ using $c_1$
__kernel void compute_fractal_code( short* currImage, short* codeBook, ...) {
    short myImage[16], centry[16];

    int average = 0;
    int imageOffset = get_global_id(0) * 16;
    for (each pixel i in region) { //loop is unrolled
        short val = currImage[imageOffset+i];
        average += val;
        myImage[i] = val;
    }
    average >>= 4; //divide by 16 to get average
    for (each pixel in region) //loop is unrolled
        myImage[i] -= average;

    ushort best_sad = 16 * 256 * 2;
    int bestCodebookEntry = -1, bestScale = -1;
    for (each codebook entry icode ) {
        for (i=0; i<16; i++) //loop is unrolled
            centry[i] = codeBook[icode*16+i];
        for (each scale factor sFac) {
            ushort sad = 0;
            for (i=0; i<16; i++) //loop is unrolled
                sad += abs(sFac * centry[i] - myImage[i]);
            if (sad < best_sad) {
                best_sad = sad;
                bestCodebookEntry = icode;
                bestScale = sFac;
            }
        }
    }
}

Compute average (offset)

Compare against all codebook entries

7 scale factors
{1.0, 0.75, 0.5, 0.25, -0.25, -0.5, -0.75, -1.0}
Platforms Evaluated

- Used the latest platforms available at time of writing

<table>
<thead>
<tr>
<th>Test Platform</th>
<th>Representative</th>
<th>Process</th>
<th>Memory Bandwidth</th>
<th>Cache Size</th>
<th>Board Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-Core CPU</td>
<td>Intel Xeon W3690</td>
<td>32nm</td>
<td>32 GB/s</td>
<td>12 MB</td>
<td>130W</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA Fermi C2075</td>
<td>40nm</td>
<td>144 GB/s</td>
<td>768 KB</td>
<td>215W</td>
</tr>
<tr>
<td>FPGA</td>
<td>Altera Stratix IV 530</td>
<td>40nm</td>
<td>12.8 GB/s</td>
<td>none</td>
<td>21W</td>
</tr>
<tr>
<td>FPGA</td>
<td>Altera Stratix V 5SGXA7</td>
<td>28nm</td>
<td>25.6 GB/s</td>
<td>none</td>
<td>25W</td>
</tr>
</tbody>
</table>

- Used publicly available video sequences
  - Full-color 704x576 videos
  - SNRs > 30dB indicate a high quality result
  - Can achieve ~9.7x compression for Y plane
    - Higher for Cr and Cb planes

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SNR (Y)</th>
<th>SNR (Cb)</th>
<th>SNR (Cr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>30.9</td>
<td>43.3</td>
<td>46.6</td>
</tr>
<tr>
<td>crew</td>
<td>34.3</td>
<td>43.0</td>
<td>40.3</td>
</tr>
<tr>
<td>harbour</td>
<td>32.4</td>
<td>43.3</td>
<td>45.1</td>
</tr>
<tr>
<td>soccer</td>
<td>33.3</td>
<td>42.2</td>
<td>44.7</td>
</tr>
<tr>
<td>ice</td>
<td>37.0</td>
<td>44.2</td>
<td>46.7</td>
</tr>
</tbody>
</table>
Multi-Core CPU Results

- **Intel Xeon W3690**
  - 6 cores running at 3.46GHz, plus hyperthreading
  - 12 MB on-chip cache

- **Used Intel OpenCL SDK**
  - No communication overhead since kernels run on same device as host
  - Kernel time defined as the average kernel runtime per frame processed

<table>
<thead>
<tr>
<th>Variant</th>
<th>Kernel Time (ms)</th>
<th>Transfer Time (ms)</th>
<th>Overall FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Code</td>
<td>1639</td>
<td>0</td>
<td>0.60</td>
</tr>
<tr>
<td>OpenCL (float-pt)</td>
<td>196.1</td>
<td>0</td>
<td>4.6</td>
</tr>
<tr>
<td>OpenCL (fixed-pt)</td>
<td>325.9</td>
<td>0</td>
<td>2.8</td>
</tr>
</tbody>
</table>

- **8x improvement when parallelized**
  - Run 1 thread per 4x4 region (no inter-thread communication)

- **Fixed-point implementation actually hurts performance**
  - The number of instructions increased
GPU Architecture

- Highly optimized processors known for graphics computation
- Designed to maximize application throughput
- High memory bandwidth (144GB/s)
- An array of compute units
  - Streaming Multiprocessor
- Process tens of thousands of threads in parallel
  - Hardware support for context switching between groups of threads
- Programmed using CUDA or NVIDIA OpenCL SDK

![Fermi GPU Streaming Multiprocessor](image)
GPU Results

- **Fractal encoding algorithm seems to be a good fit for GPU**
  - Works on small independent regions at a time (1 thread per 4x4 region)
  - GPU can launch thousands of threads in parallel for each region
  - Low demand for shared memory or local registers

- **Specify workgroups (W) due to GPU hierarchy**
  - A group of threads that cooperate to solve a sub-problem
  - All threads in the workgroup run on the same SMP (CUDA core)

- **Tradeoff in workgroup size**
  - More threads → Allows for a greater ability for the GPU to hide high-latency operations
  - More threads → Greater demand on shared resources (eg. Registers)

<table>
<thead>
<tr>
<th>Variant</th>
<th>Kernel Time (ms)</th>
<th>Transfer Time (ms)</th>
<th>Overall FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL(W=16)</td>
<td>10.79</td>
<td>3.1</td>
<td>39.1</td>
</tr>
<tr>
<td>OpenCL(W=32)</td>
<td>6.15</td>
<td>3.1</td>
<td>50.1</td>
</tr>
<tr>
<td>OpenCL(W=64)</td>
<td><strong>5.17</strong></td>
<td>3.1</td>
<td><strong>53.1</strong></td>
</tr>
<tr>
<td>OpenCL(W=128)</td>
<td>5.28</td>
<td>3.1</td>
<td>52.8</td>
</tr>
<tr>
<td>OpenCL(W=64, fixed-pt)</td>
<td>16.62</td>
<td>3.1</td>
<td>31.0</td>
</tr>
</tbody>
</table>
FPGAs as Accelerators

- An array of programmable logic connected with a grid of programmable routing wires
- Flexible, able to implement a customized datapath
- Much lower power in comparison to CPUs and GPUs
- Memory bandwidth generally much lower
  - Depends on board design
- Traditionally implemented using hardware description languages
  - Intimate knowledge of device architecture and cycle accuracy required
  - Can be a challenge for HPC adoption
Altera OpenCL SDK

- A high-level synthesis tool that compiles OpenCL code to HDL for FPGA implementation

- Translates the kernel to hardware by creating a circuit to implement each operation

```c
__kernel void sum(__global const float *a, __global const float *b, __global float *answer) {
    int xid = get_global_id(0);
    answer[xid] = a[xid] + b[xid];
}
```
Pipeline Parallelism

- Run threads in parallel using the same hardware
- Each parallel thread is associated with an ID
  - Indicates the subset of data it operates on

(a) First Cycle
(b) Second Cycle
(c) Third Cycle
System Solution

- Compiler creates interfaces to external and internal memory
- Automatically timing closed
 extern __device__ void compute_fractal_code( short* currImage, short* codeBook, ...) {
    short myImage[16], centry[16];
    int average = 0;
    int imageOffset = get_global_id(0) * 16;
    for (each pixel i in region) { //loop is unrolled
        short val = currImage[imageOffset+i];
        average += val;
        myImage[i] = val;
    }
    average >>= 4; //divide by 16 to get average
    for (each pixel in region) //loop is unrolled
        myImage[i] -= average;
    ushort best_sad = 16 * 256 * 2;
    int bestCodebookEntry = -1, bestScale = -1;
    for (each codebook entry icode ) {
        for (i=0; i<16; i++) //loop is unrolled
            centry[i] = codeBook[icode*16+i];
        for (each scale factor sFac) {
            ushort sad = 0;
            for (i=0; i<16; i++) //loop is unrolled
                sad += abs(sFac * centry[i] - myImage[i]);
            if (sad < best_sad) {
                best_sad = sad;
                bestCodebookEntry = icode;
                bestScale = sFac;
            }
        }
    }
}
FPGA Results

- **Used Stratix IV and V FPGAs**
  - Easy to retarget the OpenCL code; no code change required

- **Because of pipeline parallelism, performance increases with more copies of pipeline**
  - Control this with loop unrolling

<table>
<thead>
<tr>
<th>Variant</th>
<th>Kernel Time (ms)</th>
<th>Transfer Time (ms)</th>
<th>Overall FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix IV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCL (U=24, fixed-pt)</td>
<td>2.0</td>
<td>2.2</td>
<td>70.9</td>
</tr>
<tr>
<td>Stratix V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCL (U=1, fixed-pt)</td>
<td>20.20</td>
<td>1.9</td>
<td>28.4</td>
</tr>
<tr>
<td>OpenCL (U=24, fixed-pt)</td>
<td>1.72</td>
<td>1.9</td>
<td>74.4</td>
</tr>
<tr>
<td>OpenCL (U=2, float-pt)</td>
<td>11.84</td>
<td>1.9</td>
<td>38.8</td>
</tr>
</tbody>
</table>

- **Fixed-point computations improve performance significantly**
Summary

- Performed platform evaluation using OpenCL
- Implemented fractal encoding and map it efficiently for multi-core CPUs, GPUs, and FPGAs

<table>
<thead>
<tr>
<th>Platform</th>
<th>Kernel Runtime</th>
<th>FPS</th>
<th>Board Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-Core CPU (Intel Xeon W3690)</td>
<td>196.1</td>
<td>4.6</td>
<td>130 W</td>
</tr>
<tr>
<td>GPU (NVIDIA Fermi 2075)</td>
<td>5.17</td>
<td>53.1</td>
<td>215 W</td>
</tr>
<tr>
<td>FPGA (Altera Stratix V 5SGXA7)</td>
<td>1.72</td>
<td>74.4</td>
<td>25 W</td>
</tr>
</tbody>
</table>

- Showed that core computation can be 3x faster on FPGA vs. GPU with 8x less power
- Using a High Level Synthesis tool can dramatically reduce the time required for FPGA implementation
Thank You
Backup
Objective

- A programming model study using OpenCL
- Implement fractal compression in OpenCL
  - An video encoding algorithm based on iterated function systems (IFS)
- Code is ported to and optimized for multi-core CPUs, GPUs and FPGAs
- Introduce Altera’s OpenCL-to-FPGA Compiler
- Performance comparisons
  - Between multi-core CPUs, GPUs, and FPGAs
  - Between OpenCL and hand-coded RTL implementations
Fractal Video Compression

- **Encode new frames using the codebook of the original frame**
- **Cross-coding experiment**
  - Use the codebook of one image to encode another

<table>
<thead>
<tr>
<th></th>
<th>Aerial</th>
<th>Airplane</th>
<th>Balloon</th>
<th>Girl</th>
<th>Lenna</th>
<th>Mandrill</th>
<th>Parrots</th>
<th>Pepper</th>
<th>Sailboat</th>
<th>Couple</th>
<th>Milkdrop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerial</td>
<td>27.5</td>
<td>25.7</td>
<td>34.1</td>
<td>30.8</td>
<td>27.6</td>
<td>24.6</td>
<td>27.9</td>
<td>26.5</td>
<td>27.1</td>
<td>30.5</td>
<td>30.0</td>
</tr>
<tr>
<td>Airplane</td>
<td>27.0</td>
<td>27.3</td>
<td>34.0</td>
<td>31.0</td>
<td>28.2</td>
<td>24.3</td>
<td>28.5</td>
<td>27.1</td>
<td>27.4</td>
<td>30.3</td>
<td>31.7</td>
</tr>
<tr>
<td>Balloon</td>
<td>26.9</td>
<td>25.1</td>
<td>34.9</td>
<td>30.2</td>
<td>27.4</td>
<td>24.1</td>
<td>27.5</td>
<td>25.8</td>
<td>26.5</td>
<td>30.3</td>
<td>29.6</td>
</tr>
<tr>
<td>Girl</td>
<td>27.0</td>
<td>25.1</td>
<td>34.6</td>
<td>32.1</td>
<td>28.0</td>
<td>24.2</td>
<td>27.6</td>
<td>26.4</td>
<td>26.9</td>
<td>30.9</td>
<td>29.7</td>
</tr>
<tr>
<td>Lenna</td>
<td>26.9</td>
<td>26.0</td>
<td>34.6</td>
<td>31.3</td>
<td>29.5</td>
<td>24.3</td>
<td>28.4</td>
<td>27.1</td>
<td>27.2</td>
<td>31.1</td>
<td>31.1</td>
</tr>
<tr>
<td>Mandrill</td>
<td>27.0</td>
<td>25.4</td>
<td>34.0</td>
<td>30.7</td>
<td>27.8</td>
<td>24.6</td>
<td>27.8</td>
<td>26.6</td>
<td>26.8</td>
<td>30.9</td>
<td>29.6</td>
</tr>
<tr>
<td>Parrots</td>
<td>27.1</td>
<td>26.3</td>
<td>34.8</td>
<td>31.2</td>
<td>28.5</td>
<td>24.2</td>
<td>28.7</td>
<td>27.0</td>
<td>27.3</td>
<td>30.7</td>
<td>30.9</td>
</tr>
<tr>
<td>Pepper</td>
<td>27.4</td>
<td>26.6</td>
<td>35.0</td>
<td>31.9</td>
<td>29.1</td>
<td>24.6</td>
<td>28.7</td>
<td>28.7</td>
<td>28.3</td>
<td>31.8</td>
<td>31.6</td>
</tr>
<tr>
<td>Sailboat</td>
<td>27.5</td>
<td>27.0</td>
<td>34.8</td>
<td>31.6</td>
<td>28.5</td>
<td>24.7</td>
<td>28.7</td>
<td>27.5</td>
<td>28.3</td>
<td>31.3</td>
<td>31.5</td>
</tr>
<tr>
<td>couple</td>
<td>27.1</td>
<td>25.1</td>
<td>34.7</td>
<td>30.9</td>
<td>27.7</td>
<td>24.3</td>
<td>27.6</td>
<td>26.6</td>
<td>26.8</td>
<td>31.6</td>
<td>29.8</td>
</tr>
<tr>
<td>milkdrop</td>
<td>26.9</td>
<td>25.8</td>
<td>34.1</td>
<td>31.0</td>
<td>27.5</td>
<td>23.9</td>
<td>28.0</td>
<td>26.3</td>
<td>26.7</td>
<td>30.1</td>
<td>31.1</td>
</tr>
<tr>
<td>Average (dB)</td>
<td>26.9</td>
<td>25.1</td>
<td>34.0</td>
<td>30.2</td>
<td>27.4</td>
<td>23.9</td>
<td>27.5</td>
<td>25.8</td>
<td>26.5</td>
<td>30.1</td>
<td>29.6</td>
</tr>
<tr>
<td>Loss (dB)</td>
<td>-0.6</td>
<td>-2.3</td>
<td>-1.0</td>
<td>-1.9</td>
<td>-2.1</td>
<td>-0.7</td>
<td>-1.2</td>
<td>-2.9</td>
<td>-1.8</td>
<td>-1.5</td>
<td>-1.5</td>
</tr>
</tbody>
</table>

- Show an average loss in PSNR of only ~1dB
Hand-coded RTL Implementation

- Onchip frame buffer and codebook

From Frame Buffer SRAM

Image Block Register $R_i$

16-Value Average

$R_{i,0}[8..0]$ $R_{i,15}[8..0]$ $sD_{j,0}[8..0]$ $sD_{j,15}[8..0]$

Scaling Block ($s \cdot D_j$)

Codebook RAM

Addr[9..0] Counter[9..0]

Min Error[11..0] $E_{\text{best}}$

Absolute Value

Error < Min Error

© 2013 Altera Corporation—Public
## OpenCL vs. Handcoded

- Core computation replicated for each scale factor

<table>
<thead>
<tr>
<th></th>
<th>Handcoded</th>
<th>OpenCL SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame buffer</td>
<td>On-chip memory</td>
<td>Global memory</td>
</tr>
<tr>
<td>Codebook</td>
<td>On-chip memory</td>
<td>Global memory</td>
</tr>
<tr>
<td>Unroll</td>
<td>4x</td>
<td>24x</td>
</tr>
<tr>
<td>Development Time</td>
<td>&gt;1 month</td>
<td>Hours</td>
</tr>
</tbody>
</table>

- **OpenCL design flow greatly simplifies FPGA implementation**
  - Automates external interfacing such as DDR and PCIe