GPGPU-Assisted Nonlinear Denoising Filter Generation for Video Coding

Seishi Takamura and Atsushi Shimizu
NTT Corporation, Japan

Summary

- State-of-the-art video coding technologies such as H.265/HEVC employ in-loop denoising filters.
- We have developed a new type of in-loop denoising filter with Genetic Programming (GP), which is heavily nonlinear and content-specific.
- To boost the evolution, GPGPU is utilized in filter evaluation process.
- Proposed method yielded better denoising filter in 100x less time.
- The bit rate reduction of 1.492-2.569% was achieved against the reference software of H.265/HEVC.
Video Coding Block Diagram

Video Input

Transform → Quantization → Entropy Coding

Inverse Quantization → Inverse Transform

Inverse Transform

Denoising Filter (DF, SAO, ALF, etc)

Target of evolution

Reconstructed Videos

Intra-frame Prediction

Inter-frame Prediction

Compressed Bitstream
A Leap from Linear Denoising Filter

Decoded Frame (large distortion) → Restored Frame (less distortion) → Decoded Frame (large distortion)

Nonlinear filter

Restored Frame (much less distortion)
Denoising Filter Support
Nodes used by our Filter

- **Terminal nodes**
  - I: pixel value of p
  - Ixx: \((pxx + qxx) / 2\)
  - Dxx: \((pxx - qxx) / 2\)
  - Ils: least-square restored value, a linear combination of I, I00… I11 with offset.
  - x, y: horizontal and vertical coordinate of the pixel.
  - value: immediate values such as “0.3”.

- **Functional nodes**
  - min, max, average, abs, /, *, +, −,
  - exp, pow, log, sqrt, sin, cos, tan, asin, acos, atan,
  - sinh, cosh, tanh, conditional branch

In addition, followings are defined

- \(\text{and}(a, b) := (a>=0 \&\& b>=0) \ ? \ (a+b)/2 : -(|a|+|b|)/2\),
- \(\text{or}(a, b) := (a>=0 || b>=0) \ ? \ (|a|+|b|)/2 : -(|a|+|b|)/2\),
- \(\text{xor}(a, b) := (ab<=0) \ ? \ (|a|+|b|)/2 : -(|a|+|b|)/2\).
Serializations of a Tree

- Normal expression (or infix notation):
  \[
  \frac{\sin(I20) + \max(I01, \log(0.5))}{2}
  \]

- Lisp S-expression (or prefix notation):
  \[
  \frac{\text{div} (\text{add} (\text{sin} (I20))) (\max (I01) (\log 0.5)))}{2}
  \]

- Reverse Polish notation (or postfix notation):
  \[
  I20 \sin I01 0.5 \log \max \text{add} 2.0 \text{div}
  \]

- We used Reverse Polish notation (as described later).
- The fitness function in the evolution is \(D + \lambda R\), where
  - \(D\) is the squared sum of the errors between the filtered image and original image
  - \(R\) is the amount of tree information that represents the filter algorithm
  - \(\lambda\) is the same Lagrange multiplier as the encoder uses during rate-distortion optimization process
We convert the tree in Reverse Polish Notation (RPN) prior to the evaluation.

Linearized instructions are stuffed from the middle of the array (a) toward the beginning.

Immediate values are picked out and stuffed from the end (c).

Filter evaluation procedure is like following:

```c
for (index = 0; index < array_length; index++) {
    switch (funcIDs[index]) {
        case add: a=pop(); b=pop(); push(a+b); break;
        case sin: a=pop(); push(sin(a)); break;
        case imm: push(<the value>); break;
        case I: push(I); break;
        case I00: push(I00); break;
        ...
    }
}
```
### Simulation Conditions

<table>
<thead>
<tr>
<th><strong>CPU</strong></th>
<th>Intel Core i7-3960X Extreme Edition, C2 stepping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock rate</td>
<td>3.3GHz</td>
</tr>
<tr>
<td>Cores</td>
<td>6 (one core is used for the CPU-experiment)</td>
</tr>
<tr>
<td>Hyper threading</td>
<td>on</td>
</tr>
<tr>
<td>Memory</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

| **OS** | Ubuntu Linux 12.04.2 LTS x86_64 Desktop Edition |

<table>
<thead>
<tr>
<th><strong>GPU</strong></th>
<th>NVIDIA GeForce GTX 690</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA capability</td>
<td>3.0</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>1536</td>
</tr>
<tr>
<td>GPU Clock rate</td>
<td>1.020 GHz</td>
</tr>
<tr>
<td>Global memory</td>
<td>2048 MB</td>
</tr>
<tr>
<td>L2 Cache Size</td>
<td>512 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CUDA</strong></th>
<th>Driver version: 5.0.35, x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK/Toolkit version</td>
<td>5.0.35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>C++ Compiler</strong> (as the backend for nvcc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel C++ Compiler version: 12.1.5 20120612</td>
</tr>
</tbody>
</table>

### Video sequences used

- BQMall (832x480)
- BQTerrace (1920x1080)
- RaceHorces (416x240)
CPU vs. GPU Comparison

- Filter (of 121 nodes) evaluation time over BQMall (832x480)

<table>
<thead>
<tr>
<th></th>
<th>Time [sec]</th>
<th>Speed-up (vs.CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (1 core)</td>
<td>0.336489</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>0.002674</td>
<td>125.8x</td>
</tr>
</tbody>
</table>

- Filter evolution speed for BQMall (832x480)

Better fitness

100x time difference
## Coding Performance Comparison
(vs. original H.265/HEVC)

<table>
<thead>
<tr>
<th>Sequence</th>
<th>QP</th>
<th>Rate (a) [bits]</th>
<th>Y-PSNR [dB]</th>
<th>BD-rate vs. HM</th>
<th>Y-PSNR [dB]</th>
<th>BD-rate vs. HM</th>
<th>Filter info (R) [bits]</th>
<th>Total rate (a+R) [bits]</th>
<th>Y-PSNR [dB]</th>
<th>BD-rate vs. HM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQSquare</td>
<td>22</td>
<td>210,720</td>
<td>41.53</td>
<td>41.54</td>
<td></td>
<td></td>
<td>626</td>
<td>211,346</td>
<td>41.71</td>
<td>-1.492%</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>138,152</td>
<td>37.16</td>
<td>37.17</td>
<td>33.33</td>
<td></td>
<td>315</td>
<td>138,467</td>
<td>37.27</td>
<td>-1.437%        (vs. ALF on)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>88,288</td>
<td>33.30</td>
<td>33.33</td>
<td>29.70</td>
<td></td>
<td>329</td>
<td>88,617</td>
<td>33.46</td>
<td>-1.455%        (vs. ALF off)</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>55,048</td>
<td>29.65</td>
<td>29.70</td>
<td></td>
<td></td>
<td>418</td>
<td>55,466</td>
<td>29.93</td>
<td>-1.437%</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>210,944</td>
<td>41.53</td>
<td>41.54</td>
<td></td>
<td></td>
<td>520</td>
<td>211,464</td>
<td>41.69</td>
<td>-1.437%</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>138,352</td>
<td>37.16</td>
<td>37.17</td>
<td>33.35</td>
<td></td>
<td>445</td>
<td>138,797</td>
<td>37.30</td>
<td>-1.455%        (vs. ALF on)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>88,504</td>
<td>33.33</td>
<td>33.35</td>
<td>29.72</td>
<td></td>
<td>279</td>
<td>88,783</td>
<td>33.48</td>
<td>-1.455%        (vs. ALF off)</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>55,392</td>
<td>29.71</td>
<td>29.72</td>
<td></td>
<td></td>
<td>315</td>
<td>55,707</td>
<td>29.95</td>
<td>-1.437%</td>
</tr>
<tr>
<td>BQSquare</td>
<td>22</td>
<td>174,448</td>
<td>42.19</td>
<td>42.30</td>
<td></td>
<td></td>
<td>1195</td>
<td>175,643</td>
<td>42.47</td>
<td>-2.569%</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>109,264</td>
<td>37.97</td>
<td>38.10</td>
<td>34.21</td>
<td></td>
<td>698</td>
<td>109,962</td>
<td>38.18</td>
<td>-2.569%</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>63,848</td>
<td>34.08</td>
<td>34.21</td>
<td>30.71</td>
<td></td>
<td>750</td>
<td>64,598</td>
<td>34.35</td>
<td>-2.569%</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>34,696</td>
<td>30.57</td>
<td>30.71</td>
<td></td>
<td></td>
<td>536</td>
<td>35,232</td>
<td>30.86</td>
<td>-2.569%</td>
</tr>
<tr>
<td>RaceHorses</td>
<td>22</td>
<td>174,936</td>
<td>42.26</td>
<td>42.29</td>
<td></td>
<td></td>
<td>321</td>
<td>175,257</td>
<td>42.36</td>
<td>-0.843%        (vs. ALF on)</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>109,536</td>
<td>38.12</td>
<td>38.14</td>
<td>34.26</td>
<td></td>
<td>36</td>
<td>109,572</td>
<td>38.13</td>
<td>-0.843%        (vs. ALF off)</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>64,128</td>
<td>34.26</td>
<td>34.26</td>
<td>30.74</td>
<td></td>
<td>376</td>
<td>64,504</td>
<td>34.39</td>
<td>-0.843%</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>34,992</td>
<td>30.73</td>
<td>30.74</td>
<td></td>
<td></td>
<td>236</td>
<td>35,228</td>
<td>30.85</td>
<td>-0.843%</td>
</tr>
</tbody>
</table>

**Negative values mean better performance**

HM: H.265/HEVC reference software (used as an anchor)
*ALF: adaptive loop filter (state-of-the-art loop filter)
**LS filter: least square filter. Filter info(R) = 448 bits
Example of Generated Filter

RaceHorses, QP=22, ALF-off, filter information (R) = 1,195 bits

\[
(\text{add} (\text{add} (\text{add} (\text{mul} (I) 0.932803332806) (\text{mul} (I01) 0.087968140841)) (\text{add} (\text{mul} (I02) −0.051799394190) (\text{mul} (I00) 0.095137931406))) (\text{add} (\text{add} (\text{mul} (I03) −0.050682399422) (\text{mul} (I04) −0.040202748030)) (\text{add} (\text{mul} (I05) −0.052293013781) (\text{mul} (\text{ave} (I02) (\text{tan} (I12))) 0.017782183364)))) (\text{add} (\text{add} (\text{mul} (I07) 0.025515399873) (\text{mul} (I08) 0.025515399873)) (\text{sub} (\text{mul} (\text{sin} (\text{atan} (\text{and} (I09) (I21)))) 0.016251996160) (\text{mul} (\text{tanh} (\text{tanh} (\text{tanh} (\text{mul} (I02) (\text{asin} \log (\text{sinh} (\text{sqr} (\text{div} (\text{mul} (I05) (\text{sqr} (\text{div} (\text{atan} (\text{mul} (\text{asin} (\text{asin} (\text{sqr} (I)))) (\text{sqr} (\text{sqr} (\text{div} (I05) (I13)))) (\text{sqr} (\text{div} (\text{sin} (I19)) (I01)))) (\text{sqr} (I01)))) (\text{sqr} (I03)))))))))))) 0.005235218443) (\text{mul} (I29) −0.005818639882))
\]

Conclusion

- A novel method to generate denoising filter that enhances the coding performance is proposed.
- GPGPU accelerated the evolution by around 100 times than the CPU.
- Generated filters outperformed least square filter and state-of-the-art filter, i.e., ALF.