VLD in H.264

By C.G Liu
Agenda

- 1 Overview
- 2 Huffman
- 3 Unary Coding (Golomb Coding)
- 4 CAVLC
- 5 CABAC
- 6 VLD in ds2
- 7 Q&A
Overview

• H262 (MPEG-2) & VC-1
  – Uses static Huffman coding for entropy coding

• H264 (MPEG-4 Part 10)
  – 4 profiles
  – Baseline & Extended – Context Adaptive Variable Length Coding (CAVLC)
  – Main and High – CAVLC or Context Adaptive Binary Arithmetic Coding (CABAC)
Huffman

- Huffman encoding
  - Generate statistics of symbols occurring
  - Generate Prefix code (uniquely decodable) based on probability of occurrence
Huffman Coding Example

![Huffman Coding Diagram](image)
Huffman Coding

• Good – Simple to encode & decode
  – Can be two pass (Generate Table, Encode)
• MPEG-2 uses static Huffman coding
  – Tables generated after lots of experiments
  – Transmitting new tables not negligible
  – Very close to optimal, but must assign whole bits
In H264

1. Unary Coding (Golomb Coding)
   - Header information
   - 4 Types – Unsigned Unary, Signed Unary, Truncated and Mapped

2. CAVLC-context-adaptive variable length coding

3. CABAC
Unary Coding (Golomb Coding)

- `codeNum` can be decoded as follows:
  1. Read in M leading zeros followed by 1.
  2. Read M-bit INFO field.
  3. $\text{codeNum} = 2^M + \text{INFO} - 1$

<table>
<thead>
<tr>
<th>Bit string</th>
<th>codeNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>3</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 1 1 0</td>
<td>5</td>
</tr>
<tr>
<td>0 0 1 1 1</td>
<td>6</td>
</tr>
<tr>
<td>0 0 0 1 0 0</td>
<td>7</td>
</tr>
<tr>
<td>0 0 0 1 0 1</td>
<td>8</td>
</tr>
<tr>
<td>0 0 0 1 0 0</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Unary Coding (Golomb Coding)

- For $ue$, $code\_num = k$
- For $se$, $code\_num = 2|k| (k \leq 0)$ ; $code\_num = 2|k|-1 (k > 0)$
- For $te$, if $x > 1$ te=$ue$; else te=!”read_bit(1)
- For $me$,

<table>
<thead>
<tr>
<th>codeNum</th>
<th>coded_block_pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intra_4x4, Intra_8x8</td>
</tr>
<tr>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>
CAVLC

• When \( \text{entropy\_coding\_mode} \) is 0, for \( \text{residual\_block} \)
Which features are used in CAVLC

- Blocks are typically sparse. (run-level)
- The highest nonzero coefficients after the zig-zag scan are often sequences of ±1.
- The number of nonzero coefficients in neighbouring blocks is correlated. (nC,6t)
- The level of nonzero coefficients tends to be larger at the start and smaller towards the higher frequencies. (level)
## CAVLC Example

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff_token</td>
<td>TotalCoeffs=5, T1s=3</td>
<td>0000100</td>
</tr>
<tr>
<td>T1 sign (4)</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>T1 sign (3)</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>T1 sign (2)</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Level (1)</td>
<td>+1 (use Level_VLC0)</td>
<td>1</td>
</tr>
<tr>
<td>Level (0)</td>
<td>+3 (use Level_VLC1)</td>
<td>0010</td>
</tr>
<tr>
<td>TotalZeros</td>
<td>3</td>
<td>111</td>
</tr>
<tr>
<td>run_before(4)</td>
<td>ZerosLeft=3; run_before=1</td>
<td>10</td>
</tr>
<tr>
<td>run_before(3)</td>
<td>ZerosLeft=2; run_before=0</td>
<td>1</td>
</tr>
<tr>
<td>run_before(2)</td>
<td>ZerosLeft=2; run_before=0</td>
<td>1</td>
</tr>
<tr>
<td>run_before(1)</td>
<td>ZerosLeft=2; run_before=1</td>
<td>01</td>
</tr>
<tr>
<td>run_before(0)</td>
<td>ZerosLeft=1; run_before=1</td>
<td>No code required; last coefficient.</td>
</tr>
</tbody>
</table>

The transmitted bitstream for this block is 000010001110010111101101.
• When entropy_coding_mode_flag is 1, for residual_block
CABAC Overview
Initialising CABAC Engine

- Need to initialise before first symbol in slice
- Initialise Context
  - Need sliceQPY, cabac_init_idc
  - Initialise all 460 contexts
  - Output: pStateIdx[ctx], valMPS[ctx]
- Initialise Arithmetic Engine
  - codIRange = 0x01FE
  - codIOffset = Read(9)
Binarization process

- Given Syntax Element (SE)
- SE has associated binarization method
  - Special (mb_type, sub_mb_type)
  - Unary
  - Truncated Unary
  - Kth order Exp-Golomb
  - Fixed-length
  - Concatenation (FL+TU, TU+EGk)
- Produce possible bin string (codeword)
Context modeling

• For the control info in MB and Slice
  – ctxIdx = ctxIdxOffset+ctxIdxIncr

• For the residual data
  – cIdx =
    ctxIdxOffset+ctxIdxBlockCatOffset+ctxIdxIncr
Binary Arithmetic Decoding

- Probability model
  - MPS
  - State
- Decoding Engine State Update
  - Range
  - Ioffset
DecodeBin(ctxldx)

bypassFlag == 1?

No

Yes

DecodeBypass

ctxldx==276?

Yes

DecodeTerminate

No

DecodeDecision(ctxldx)

Done
Decoded Decision (ctxIdx)

qCodIRangIdx = (codIRange >> 6) & 3
codIRangeLPS = rangeTableLPS[pStateIdx][qCodIRangIdx]
codIRange = codIRange - codIRangeLPS

Yes  codIOffset >= codIRange  No

binVal = !valMPS
codIOffset = codIOffset - codIRange
codIRange = codIRangeLPS

pStateIdx == 0?

Yes  valMPS = 1 - valMPS  No

pStateIdx = transIdxLPS[pStateIdx]

ReNormD

Done
Figure 9-5 – Flowchart of bypass decoding process

- **Decoding Bypass**
  - `codlOffset = codlOffset << 1`
  - `codlOffset = codlOffset | read_blts(1)`
  
  - Decision: `codlOffset >= codlRange`
    - **Yes**:
      - `binVal = 1`
      - `codlOffset = codlOffset - codlRange`
      - **Done**
    - **No**:
      - `binVal = 0`