Parallel techniques in Video compression algorithms

Nicholas George

Why parallelization?

- Decoding/Encoding needs to be done at a certain minimum speed.
- Higher Quality algorithms require more computing power
MPEG Video Standards

• A family of related video compression algorithms
• MPEG-1, 2, and 4
• Lossy compression of input video
• Also audio standards as well

MPEG Basics

• Video generally shows redundancy in both spatial and temporal domains
• 8 x 8 pixel “pels”
• Discrete Cosine Transform used to exploit spatial similarities
• Motion-compensated prediction exploits temporal similarities
MPEG Basics

• I-Frame: Completely independent of nearby frames
• P-Frame: Uses information about preceding frame
• B-Frame: Uses information about preceding and succeeding frames

Motion Estimation

• Best results come from a full-search of all possible motion vectors
• Lots of comparisons needed – Imagine an 8x8 block with a 22x22 search window
• 8x8 = 64 subtractions needed for each location
• 15 x 15 = 225 possible locations!
Parallel Motion Estimation Prototype

- Use 15 multiplexers set up in a staggered configuration. (0-7, 1-8, 2-9, etc.)
- Determine error between original and selected pixels. 8 times per row done in parallel. 8 rows for a total of 64 cycles.
- One cycle delay for adding to calculate Mean Absolute Error for each block
- 65 cycles total for each row.
MPEG-4 Parallelized Encoder

• Macroblocks can be encoded independently
• **BUT** need information about their surrounding blocks for motion estimation.
• Solution: Give each processor local access to overlapping blocks of the source for motion estimation needs.

MPEG-4 Parallelized Encoder

• Every processor can compress equal numbers of macroblock
• **BUT** not all macroblocks are created equal.
• MPEG-4 allows arbitrarily shaped objects.
• Solution: Intelligently divide the used MBs among processors, expand their regions to be rectangular, and set unwanted MBs transparent.
Huffman Coding

- Simple, well-known algorithm
- Widely used in more complex encoding algorithms (MPEG Audio and Video, JPEG, etc)
- HuffYUV and other related lossless video codecs use it
- Shannon-Fano coding ensures that symbols with equal probability have equal length code strings.

Pro/Con

- Completely independent encoding and decoding

- Variable length encoding scheme means that you can’t arbitrarily divide the stream
Programmable Parallel Huffman Decoder

• **Must** be programmable for widespread use

• **Desired** to be parallel because of the high bitrates and requirement of timely decoding

How to add parallelization

• There is a minimum and maximum length for any given set of Huffman codes.
• Feed input to a set of decoders that detect each codelength.
• The decoded output of the first decoder to find a valid sequence is used.
• After the first codeword, one can be detected every cycle
H.26L

- Think “MPEG-4 on steroids”
- P and B-frames can predict from up to 5 other frames.
- Frames divided into slices, slices into macroblocks and macroblocks can use 7 different prediction modes.
How to parallelize?

- On slices
- On reference frames in inter-frames.
- On prediction modes in inter-frames.
- On macroblocks in intra-frames.

How much is parallel?

<table>
<thead>
<tr>
<th>Video sequence</th>
<th>Video feature</th>
<th>Runtime percentage (%)</th>
<th>Scheme 7</th>
<th>Scheme 2.47</th>
<th>Scheme 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>Slow motion of head</td>
<td>95.9</td>
<td>94.8</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>Miss, run</td>
<td>Body shake of body</td>
<td>97.3</td>
<td>95.0</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Mist, fast</td>
<td>Low motion of upper torso</td>
<td>97.8</td>
<td>93.6</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>Superman</td>
<td>Slow motion</td>
<td>96.9</td>
<td>95.7</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>Suzie</td>
<td>Hand and shoulder motion</td>
<td>96.8</td>
<td>94.2</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>Nemo</td>
<td>Background motion: feet</td>
<td>97.7</td>
<td>93.9</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>Superman</td>
<td>Violent motion and background change quickly</td>
<td>97.8</td>
<td>94.1</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>Emperor</td>
<td>Fast motion: head, body</td>
<td>97.0</td>
<td>94.0</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>Empress</td>
<td>Fast background motion and violent body motion</td>
<td>97.3</td>
<td>94.7</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>Superman</td>
<td>Fast change of background and screen</td>
<td>97.9</td>
<td>92.2</td>
<td>1.4</td>
<td></td>
</tr>
</tbody>
</table>
Load-balancing with scheme 2

- The 7 modes aren’t used in near equal amounts
- Some modes take longer than others

Load-balancing with scheme 3

- The 5 allowed frame references aren’t used equal amounts.
- In addition, B-frames have references in opposite directions of P-Frames.
Discussion Questions

• 1. MPEG-4 Based Interactive Video using Parallel Processing uses a data-parallel approach to parallelizing video compression, but they also mention object-oriented and control-parallel as parallel paradigms. What is the object-oriented paradigm and how might these two paradigms affect parallelizing video compression.

• 2. Parallel Huffman coding is the topic of one of the included texts. While it is good with coding text and images to the jpeg format, would there be any issues using it to encode these Video Object Planes (VOP) that the MPEG-4 specification describes?
• 3. Data partitioning seems to be a big issue with parallelizing MPEG-4 compression. What problems could this have if the video is highly partitioned in parallel during compression but the machine doing the decompression has no parallel abilities?