Huffman Coding with Gap Arrays for GPU Acceleration

Naoya Yamamoto, Koji Nakano, Yasuaki Ito, Daisuke Takafuji
Hiroshima University
Akihiko Kasagi, Tsuguchika Tabaru
Fujitsu Laboratories
Huffman coding

- **Lossless** data compression scheme
- Used in many data compression formats:
  - gzip, zip, png, jpg, etc.
- Uses a **codebook**: mapping of fixed-length (usually 8-bit) symbols into codewords bits.
- **Entropy coding**: Symbols appear more frequently are assigned codewords with fewer bits.
- **Prefix code**: Every codeword is not a prefix of the other codewords.

<table>
<thead>
<tr>
<th>symbol</th>
<th>codeword bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>110</td>
</tr>
<tr>
<td>E</td>
<td>111</td>
</tr>
</tbody>
</table>

**Huffman Encoding** can be done by converting each symbol to the corresponding codeword: parallel encoding is easy.

**Huffman Decoding** can be done by reading the codeword sequence from the beginning
1. identifying each codeword
2. converting it into the corresponding codeword

**Parallel Huffman decoding is hard**:
- codeword sequence has no separator to identify codewords
- It is not possible to start decoding from the middle of the codeword sequence.
- Parallel divide-and-conquer approaches that perform decoding for every equal-sized partitioned segment do not decode correctly: a codeword may be **incomplete** and separated into two segments

<table>
<thead>
<tr>
<th>symbol sequence</th>
<th>codebook sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B D E A B D C B C B D C E</td>
<td>00 01 11 10 11 11 01 10 00 11 10 11 01 11</td>
</tr>
</tbody>
</table>

ICPP2020: Huffman Coding with Gap Arrays for GPU Acceleration
Parallel GPU decoding by self-synchronization

- **Self-synchronization** of Huffman decoding [3]
  - Decoding from a middle bit will synchronize.
  - Decoding is correct after synchronization.
  - The expected length for self-synchronization is 73 [16]
  - Decoding may never synchronize in the worst case.

- **Parallel GPU decoding by self-synchronization** [29,30]
  - The codeword sequence is partitioned into equal-sized segments.
  - Each thread is assigned to a segment and starts decoding from it.
  - It continues decoding of following segments until it finds synchronization.

**Drawbacks**
- Every segment is decoded by two times or more.
- In the worst case, thread 0 must decode all segments.

---

Decoding from the beginning

```
A B D E A B D C B C B D E
```

Decoding from the 8th bit

```
A B D E A B D C B C B D E
```

**References**


Our contribution

**First contribution:** Present a gap array, a new data structure for accelerating parallel decoding
- the bit position of the first complete codeword in each segment
- Computed and attached to a codeword sequence when encoding is performed

**Gap array is very small:** array of 4 bits
- the size overhead is less than 1.5% for 256-bit segments
- the time overhead for GPU encoding is less than 20%.

**Gap array accelerate GPU decoding**
- 1.67x – 6450x faster

**Second contribution:** Develop several acceleration techniques for Huffman encoding/decoding
1. **Single Kernel Soft Synchronization (SKSS) technique** [9]
   - Only one kernel call is performed.
   - Kernel call and global memory access overhead can be reduced
2. **Wordwise global memory access**
   - four 8-bit symbols (32 bits) are read/write by one instruction.
3. **Compact codebook:** new data structure for codebooks of Huffman coding
   - Codebook size can be 64Kbytes : too large to store it in the GPU shared memory
   - The size is reduced to less than 3 Kbytes: enough small to store it in the GPU shared memory

**Experimental results for a data set of 10 files**
- Our GPU encoding/decoding is 2.87x-7.70x and 1.26-2.63x faster than previous presented GPU implementations.
- If a gap array is available, our GPU decoding is 1.67x-6450x times faster.

---

GPU Huffman encoding with a gap array

- **Naive Parallel GPU encoding**
  - Kernel 1: The prefix-sums of codeword bits are computed.
    - The bit position of the codeword corresponding to each symbol can be determined from the prefix-sums.
  - Kernel 2: The codeword of corresponding to each symbol is written.
    - Gap arrays can be written if necessary.
  - Both Kernels 1 and 2 perform global memory access.

- **GPU encoding by the Single Kernel Soft Synchronization (SKSS)**
  - Only one kernel call is performed.
  - Reduce global memory access
  - The codeword sequence are partitioned into equal-sized segments.
  - Each CUDA block \( i \) (this number is assigned by a global counter) works for encoding segment \( i \)
  - The Prefix-sums for each segment \( i \) are computed by looking back previous CUDA blocks

```
symbol sequence: A B D E A B D C B C B D C E
codeword bits:  2 2 3 3 2 2 3 2 2 2 2 3 2 3
prefix-sums:     2 4 7 10 12 14 17 19 21 23 25 28 30 33
codeword sequence: 00011101 11000111 01001100 111010111
gap array:       0 2 1 1
```

```
<table>
<thead>
<tr>
<th>CUDA block 0</th>
<th>CUDA block 1</th>
<th>CUDA block 2</th>
<th>CUDA block 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B D</td>
<td>E A B</td>
<td>D C B</td>
<td>C B D</td>
</tr>
<tr>
<td>2 2 3</td>
<td>3 2 2</td>
<td>3 2 2</td>
<td>2 2 3</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>21</td>
<td>21 23 25 28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
000111011100011101001100111010111
```
GPU Huffman decoding with a gap array

- **SKSS technique:**
  - The codeword sequence is partitioned into equal-sized segments and the gap value of each segment is available.
  - Each CUDA block \(i\) (this number is assigned by a global counter) works for decoding a segment \(i\).
  - Since the gap value is available, each CUDA block can start decoding from the first complete codeword.
  - Similarly to GPU Huffman decoding, the prefix-sums of the number of symbols corresponding to segments are computed by the SKSS.
  - From the prefix-sums, each CUDA block can determine the position in the symbol sequence where it writes the decoded symbols.

- **Compact codebook:**
  - A 64Kbyte codebook is separated into several small codebooks.
  - Primary codebook: stores codewords with no more than 11 bits.
  - Secondary codebooks: store codewords with 11 bits or more.
  - The total size is less than 3 Kbytes.

- **wordwise memory access**
  - 4 symbols are written as a 32-bit word.
  - Global memory access throughput can be improved.
## Experimental results: Data set of 10 files

### Compression ratio

<table>
<thead>
<tr>
<th>file</th>
<th>type</th>
<th>contents</th>
<th>size(Mbyte)</th>
<th>NOGAP</th>
<th>GAP</th>
<th>GAP Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>bible</td>
<td>text</td>
<td>Collection of sacred texts or scriptures</td>
<td>4.047</td>
<td>54.82%</td>
<td>55.67%</td>
<td>+0.86%</td>
</tr>
<tr>
<td>enwiki</td>
<td>xml</td>
<td>Wikipedia dump file</td>
<td>1095.488</td>
<td>68.30%</td>
<td>69.37%</td>
<td>+1.07%</td>
</tr>
<tr>
<td>mozilla</td>
<td>exe</td>
<td>Tarred executables of Mozilla</td>
<td>51.220</td>
<td>78.05%</td>
<td>79.27%</td>
<td>+1.22%</td>
</tr>
<tr>
<td>mr</td>
<td>image</td>
<td>Medical magnetic resonance image</td>
<td>9.971</td>
<td>46.37%</td>
<td>47.10%</td>
<td>+0.72%</td>
</tr>
<tr>
<td>nci</td>
<td>database</td>
<td>Chemical database of structures</td>
<td>33.553</td>
<td>30.47%</td>
<td>30.95%</td>
<td>+0.48%</td>
</tr>
<tr>
<td>prime</td>
<td>text</td>
<td>50th Mersenne number</td>
<td>23.714</td>
<td>44.12%</td>
<td>44.80%</td>
<td>+0.69%</td>
</tr>
<tr>
<td>sao</td>
<td>bin</td>
<td>The SAO star catalog</td>
<td>7.252</td>
<td>94.37%</td>
<td>95.85%</td>
<td>+1.47%</td>
</tr>
<tr>
<td>webster</td>
<td>html</td>
<td>The 1913 Webster Unabridged Dictionary</td>
<td>41.459</td>
<td>62.54%</td>
<td>63.52%</td>
<td>+0.98%</td>
</tr>
<tr>
<td>linux</td>
<td>src</td>
<td>Linux kernel 5.2.4</td>
<td>871.352</td>
<td>70.23%</td>
<td>71.32%</td>
<td>+1.10%</td>
</tr>
<tr>
<td>malicious</td>
<td>text</td>
<td>Never self-synchronizes until the end</td>
<td>1073.742</td>
<td>25.00%</td>
<td>25.39%</td>
<td>+0.39%</td>
</tr>
</tbody>
</table>

### Compression ratio

\[
\text{Compression ratio} = \frac{\text{compressed size}}{\text{uncompressed size}}
\]

**malicious**: text that never self-synchronizes

---

**ICPP2020: Huffman Coding with Gap Arrays for GPU Acceleration**
Experimental results: GPU Huffman encoding

Running time: Nvidia Tesla V100

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bible</td>
<td>0.747ms</td>
<td>0.180ms</td>
<td>0.0605ms</td>
<td>12.35x</td>
<td>2.98x</td>
<td>0.0716ms</td>
<td>+18.35%</td>
</tr>
<tr>
<td>enwiki</td>
<td>70.8ms</td>
<td>37.7ms</td>
<td>6.53ms</td>
<td>10.84x</td>
<td>5.77x</td>
<td>7.05ms</td>
<td>+7.96%</td>
</tr>
<tr>
<td>mozilla</td>
<td>4.55ms</td>
<td>1.97ms</td>
<td>0.451ms</td>
<td>10.09x</td>
<td>4.37x</td>
<td>0.495ms</td>
<td>+9.76%</td>
</tr>
<tr>
<td>mr</td>
<td>1.11ms</td>
<td>0.407ms</td>
<td>0.119ms</td>
<td>9.33x</td>
<td>3.42x</td>
<td>0.134ms</td>
<td>+12.61%</td>
</tr>
<tr>
<td>nci</td>
<td>2.00ms</td>
<td>1.31ms</td>
<td>0.339ms</td>
<td>5.90x</td>
<td>3.86x</td>
<td>0.365ms</td>
<td>+7.67%</td>
</tr>
<tr>
<td>prime</td>
<td>1.32ms</td>
<td>0.926ms</td>
<td>0.175ms</td>
<td>8.69x</td>
<td>5.29x</td>
<td>0.193ms</td>
<td>+10.29%</td>
</tr>
<tr>
<td>sao</td>
<td>1.21ms</td>
<td>0.307ms</td>
<td>0.107ms</td>
<td>11.31x</td>
<td>2.87x</td>
<td>0.123ms</td>
<td>+14.95%</td>
</tr>
<tr>
<td>webster</td>
<td>3.27ms</td>
<td>1.62ms</td>
<td>0.303ms</td>
<td>10.79x</td>
<td>5.35x</td>
<td>0.332ms</td>
<td>+9.57%</td>
</tr>
<tr>
<td>linux</td>
<td>55.0ms</td>
<td>36.0ms</td>
<td>5.59ms</td>
<td>5.59x</td>
<td>5.37x</td>
<td>6.05ms</td>
<td>+8.23%</td>
</tr>
<tr>
<td>malicious</td>
<td>36.0ms</td>
<td>26.3ms</td>
<td>4.79ms</td>
<td>7.52x</td>
<td>7.70x</td>
<td>4.98ms</td>
<td>+3.97%</td>
</tr>
</tbody>
</table>


## Experimental results: GPU Huffman decoding

### Running time: Nvidia Tesla V100

<table>
<thead>
<tr>
<th>file</th>
<th>with no gap array by self-synchronization</th>
<th>with gap arrays</th>
<th>Speedup over CUHD [29,30]</th>
<th>Speedup over CUHD [29,30]</th>
<th>Speedup over our decoding with no gap array</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CUHD [29,30]</td>
<td>Our decoding with no gap array</td>
<td>Speedup over CUHD [29,30]</td>
<td>Our decoding with gap arrays</td>
<td>Speedup over our decoding with no gap array</td>
</tr>
<tr>
<td>bible</td>
<td>0.331ms</td>
<td>0.205ms</td>
<td>1.61x</td>
<td>0.0682ms</td>
<td>4.85x</td>
</tr>
<tr>
<td>enwiki</td>
<td>40.3ms</td>
<td>22.3ms</td>
<td>1.81x</td>
<td>10.5ms</td>
<td>3.84x</td>
</tr>
<tr>
<td>mozilla</td>
<td>3.67ms</td>
<td>2.74ms</td>
<td>1.34x</td>
<td>0.674ms</td>
<td>5.45x</td>
</tr>
<tr>
<td>mr</td>
<td>0.64ms</td>
<td>0.461ms</td>
<td>1.39x</td>
<td>0.261ms</td>
<td>2.45x</td>
</tr>
<tr>
<td>nci</td>
<td>1.90ms</td>
<td>0.923ms</td>
<td>2.06x</td>
<td>0.552ms</td>
<td>3.44x</td>
</tr>
<tr>
<td>prime</td>
<td>1.67ms</td>
<td>0.636ms</td>
<td>2.63x</td>
<td>0.280ms</td>
<td>5.96x</td>
</tr>
<tr>
<td>sao</td>
<td>0.472ms</td>
<td>0.278ms</td>
<td>1.70x</td>
<td>0.120ms</td>
<td>3.93x</td>
</tr>
<tr>
<td>webster</td>
<td>1.76ms</td>
<td>0.906ms</td>
<td>1.94x</td>
<td>0.488ms</td>
<td>3.61x</td>
</tr>
<tr>
<td>linux</td>
<td>34.6ms</td>
<td>21.3ms</td>
<td>1.62x</td>
<td>9.04ms</td>
<td>3.83x</td>
</tr>
<tr>
<td>malicious</td>
<td>106000ms</td>
<td>60000ms</td>
<td>1.77x</td>
<td>9.30ms</td>
<td>11400x</td>
</tr>
</tbody>
</table>

### Speedup

- **CUHD [29,30]**
  - 9 files: 2.45x – 5.96x malicious: 11400x
- **Our decoding with gap arrays**
  - 1.34x – 2.63x
- **Our decoding with no gap array**
  - 9 files: 1.67x – 4.07x malicious: 6450x

### References

Huffman coding with gap arrays: CPU vs. GPU

CPU encoding/decoding with no gap array

CPU memory

```
A B D E A B D

00011101110001110
```

GPU encoding/decoding with gap arrays

CPU memory

```
A B D E A B D

00011101110001110
```

GPU global memory

```
A B D E A B D

0 2 1 1
```

The time for all necessary operations are included:
- Computing symbol frequency by histogramming
- Codebook generation
- Data transfer time between CPU/GPU

CPU memory

```
0 2 1 1
```

GPU memory

```
00011101110001110
```

Running time

<table>
<thead>
<tr>
<th>file</th>
<th>CPU (ms)</th>
<th>GPU (ms)</th>
<th>Speedup</th>
<th>CPU (ms)</th>
<th>GPU (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bible</td>
<td>47.0</td>
<td>1.20</td>
<td>39.2</td>
<td>25.9</td>
<td>0.59</td>
<td>43.3</td>
</tr>
<tr>
<td>enwiki</td>
<td>3500</td>
<td>158</td>
<td>22.2</td>
<td>5930</td>
<td>159</td>
<td>37.2</td>
</tr>
<tr>
<td>mozilla</td>
<td>313</td>
<td>8.67</td>
<td>36.1</td>
<td>308</td>
<td>7.95</td>
<td>38.7</td>
</tr>
<tr>
<td>mr</td>
<td>67.0</td>
<td>2.05</td>
<td>32.7</td>
<td>52.9</td>
<td>1.50</td>
<td>35.2</td>
</tr>
<tr>
<td>nci</td>
<td>177</td>
<td>5.50</td>
<td>32.2</td>
<td>170</td>
<td>4.48</td>
<td>37.9</td>
</tr>
<tr>
<td>prime</td>
<td>80.0</td>
<td>4.27</td>
<td>18.7</td>
<td>160</td>
<td>3.06</td>
<td>52.2</td>
</tr>
<tr>
<td>sao</td>
<td>75.2</td>
<td>3.15</td>
<td>23.9</td>
<td>49.3</td>
<td>1.28</td>
<td>38.4</td>
</tr>
<tr>
<td>webster</td>
<td>174</td>
<td>7.31</td>
<td>23.8</td>
<td>248</td>
<td>5.94</td>
<td>41.7</td>
</tr>
<tr>
<td>linux</td>
<td>3130</td>
<td>128</td>
<td>24.5</td>
<td>4890</td>
<td>128</td>
<td>38.3</td>
</tr>
<tr>
<td>malicious</td>
<td>2250</td>
<td>117</td>
<td>19.2</td>
<td>4500</td>
<td>119</td>
<td>37.8</td>
</tr>
</tbody>
</table>

CPU: Intel Xeon Silver 4112 (2.60GHz)
GPU: Nvidia Tesla V100

Huffman encoding Huffman decoding

<table>
<thead>
<tr>
<th>file</th>
<th>CPU</th>
<th>GPU</th>
<th>Speedup</th>
<th>CPU</th>
<th>GPU</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bible</td>
<td>47.0</td>
<td>1.20</td>
<td>39.2</td>
<td>25.9</td>
<td>0.59</td>
<td>43.3</td>
</tr>
<tr>
<td>enwiki</td>
<td>3500</td>
<td>158</td>
<td>22.2</td>
<td>5930</td>
<td>159</td>
<td>37.2</td>
</tr>
<tr>
<td>mozilla</td>
<td>313</td>
<td>8.67</td>
<td>36.1</td>
<td>308</td>
<td>7.95</td>
<td>38.7</td>
</tr>
<tr>
<td>mr</td>
<td>67.0</td>
<td>2.05</td>
<td>32.7</td>
<td>52.9</td>
<td>1.50</td>
<td>35.2</td>
</tr>
<tr>
<td>nci</td>
<td>177</td>
<td>5.50</td>
<td>32.2</td>
<td>170</td>
<td>4.48</td>
<td>37.9</td>
</tr>
<tr>
<td>prime</td>
<td>80.0</td>
<td>4.27</td>
<td>18.7</td>
<td>160</td>
<td>3.06</td>
<td>52.2</td>
</tr>
<tr>
<td>sao</td>
<td>75.2</td>
<td>3.15</td>
<td>23.9</td>
<td>49.3</td>
<td>1.28</td>
<td>38.4</td>
</tr>
<tr>
<td>webster</td>
<td>174</td>
<td>7.31</td>
<td>23.8</td>
<td>248</td>
<td>5.94</td>
<td>41.7</td>
</tr>
<tr>
<td>linux</td>
<td>313</td>
<td>128</td>
<td>24.5</td>
<td>4890</td>
<td>128</td>
<td>38.3</td>
</tr>
<tr>
<td>malicious</td>
<td>2250</td>
<td>117</td>
<td>19.2</td>
<td>4500</td>
<td>119</td>
<td>37.8</td>
</tr>
</tbody>
</table>

Encoding: 18.7x – 39.2x

Decoding: 35.2x – 52.3x
Conclusion

• We have presented new data structure gap array for accelerating Huffman decoding on GPUs.
• We have also presented several acceleration techniques for Huffman encoding/decoding on GPUs.
• The size overhead of gap arrays is small: +0.39% − +1.47%
• The time overhead of gap arrays in GPU Huffman encoding is small: +3.97% − +18.35%
• GPU Huffman decoding is much faster if gap arrays are available:
  • 9 files: 1.67x−4.07x
  • malicious file : 6450x
• Including all operations for Huffman encoding/decoding and CPU-GPU data transfer, GPU can accelerate Huffman encoding/decoding
  • Encoding: 18.7x − 39.2x
  • Decoding: 35.2x − 52.3x
• Gap arrays should be attached if Huffman encoding/decoding are performed using GPUs.