CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

Adnan Ozsoy

School of Informatics and Computing
Indiana University, Bloomington, USA

Hacettepe University, TURKEY
"If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?"

Seymour Cray, Father of the Supercomputer
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Era of heterogeneous computing
  • General Purpose Graphics Processing Units (GPGPUs)
  • CPU / GPU
  • World’s fastest supercomputers
    • TITAN, Tianhe-1A, Nebulae, Tsubame 2.0, ...etc.
• General Purpose Graphic Processing Units in HPC
• The common parallelization approach applied
  • Parallelize the best known sequential algorithm
  • Optimize for GPUs.

• Problem:
  • Solutions only limited to a single application
  • Directly apply CPU solution to GPU
    • Not fully utilize the resources on a GPU.
  • GPU has different architecture than CPU
Problem:
- Scaling data usage in applications with **BIG** data
- Reduce storage requirements
- Improve data communication performance

Data compression
- Trade off in increasing running time

GPGPUs

Solution:
- CULZSS - Parallel Streaming Compression on GPGPUs
- Lempel-Ziv-Storer-Szymanski (LZSS) Algorithm
- CULZSS-Bit – Improvement over the algorithm
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

- Lempel-Ziv-Storer-Szymanski (LZSS) Algorithm
  - A variant of LZ77
  - Dictionary encoding
    - Sliding history buffer
    - Uncoded lookahead buffer
    - Visited and upcoming data
    - Finding the max substring match
• LZSS Algorithm
  • 2 stages – Matching and Encoding
  • Example:

```
sliding history
abcabdaabdaabcd...  
  1 match
  2 match
  3 match

encoded lookahead
abcde...  
  2 match
  4 match

search for substrings starting from the first character
```
• LZSS Algorithm
  
  Finding matching information
  
  \[a \ b \ c \ a \ b \ d \ a \ a \ b \ d \ a \ b \ c \ d \ d - a \ b \ c \ d \ e \ldots\]
• LZSS Algorithm
  Finding matching information
  a b c a b d a a b d a b c d d – a b c d e ..
  0 1 2 3 4 5 6 7 8 9 10 ... 4 match – position 10
• LZSS Algorithm
  Encoding information
  a b c a b d a a b d a b c d d – a b c d e ..

  (4,10) → write to file

  b d a a b d a b c d d a b c d – e ..
• Observations on Longest Prefix Match

Longest Prefix matching

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Observations on LPM
  • Matching information of every element required
  • Binary matrix

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Observations on LPM
  • Core regular problem
    • Matching information of every element required
  • Binary matrix
  • Bit parallelism
    • Bits packed into a word
    • Using bit operations on words

Word Size

a 1001001..

abcabda..
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Observations on LPM
  • Matching information of every element required
  • Binary matrix
  • Bit parallelism
  • Pre-compute matching data for given query string
    • Alphabet-strings

<table>
<thead>
<tr>
<th></th>
<th>char a</th>
<th>char b</th>
<th>char c</th>
<th>char d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Bit Vector Approach
  • Matching information of every element required
  • Binary matrix
  • Bit parallelism
  • Pre-compute matching data for given query string
    • Alphabet-strings
  • Combining all of these → Bit-Vector Approach
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Bit Vector Approach
  • Very Suitable GPGPU
  • Less code divergence
  • Reuse of data – fast lookup memory
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Shift-Or Algorithm (Exact Matching)

```plaintext
Algorithm 1 Shift-Or (Courtesy of Baeza-Yates)
loc ← 0, first ← pattern[0]
while loc < end do
  while loc < end AND text[loc] ≠ first do
    loc ++
  end while
  state ← (~ 0)
  while loc ≤ end AND state ≠ initial do
    state = (state ◦ ◦ B) | alpha[text[loc++]]
    matches += (state < lim)
  end while
end while
```
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Shift-Or

**Algorithm 8** Simplified Shift-Or (no skip)

\[
\text{loc} \leftarrow 0, \text{state} \leftarrow (\sim 0) \\
\text{while} \; \text{loc} < \text{end} \; \text{do} \\
\quad \text{state} = (\text{state} \ll B) | \\
\quad \alpha[\text{text}[\text{loc}++]] \\
\quad \text{matches} += \text{state} < \text{lim} \\
\text{end while}
\]

<table>
<thead>
<tr>
<th>state</th>
<th>init</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

- Bit-vector Longest Prefix Match

<table>
<thead>
<tr>
<th>state</th>
<th>init</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>smax</th>
<th>init</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3: New statemax variable to hold longest prefix match information
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

• Bit-vector Longest Prefix Match

Algorithm 2 Bit-Vector Longest Prefix Match Algorithm

\[
\begin{align*}
\text{state} & \leftarrow (\sim 0), \text{initial} \leftarrow (\sim 0), \text{statemax} \leftarrow (\sim 0) \\
\text{while} & \quad \text{loc} < \text{end} + 1 \quad \text{do} \\
\quad & \quad \text{state} = (\text{state} \ll B) | \alpha [\text{text}[\text{loc}++]] \\
\quad & \quad \text{statemax} = \text{statemax} \& \text{state} \\
\text{end while}
\end{align*}
\]
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

Loop
1. Build alpha for 64 chars
2. Limit longest match size to 32, so each fit into one alpha word
3. Do search in window

Lookahead buffer as pattern

Window - 128 chars or more

64 chars

32 threads
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

Loop
1. Build alpha for 64 chars
2. Limit longest match size to 32, so each fit into one alpha word
3. Do search in window
• Bit-vector Compression

```
Algorithm 3 Bit-Vector Compression Algorithm

state ← (~ 0), initial ← (~ 0), statemax ← (~ 0), loc ← 0
initAlpha(alpha)

while loc < end do
    buildAlpha(alpha, loc)
    i ← loc
    while i < loc + BUFFER - SIZE do
        state = (state << B) | alpha[text[tid + (i++)]] >> tid
        statemax = statemax & state
    end while

FindMaxZeroLoc(statemax)
loc += NUM - OF - THREADS ,
statemax ← (~ 0), state ← (~ 0)
end while
```
Evaluation

- CUDA capability of 3.5
- K20 Kepler architecture GPUs
- 5GB device memory, 2496 cores
- CULZSS implementation
  - Five sets; C files, State Maps, Dictionary, Tar file, Iperf data
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

- 25% occupancy
- 38% occupancy
- 56M shared mem
- 2300 Mbps
- 0 conflicts
- 1.3x faster
- 2.33x faster
- 40% execution time wasted on divergent code

Graph showing throughput (MBps) vs GPU version.

V1: Shared alpha - Texture text 32 Thread
V2: Shared alpha - Texture text 64 Thread
V3: Shared alpha - Texture text 128 Thread
V4: Optimized code 256 Alphabet 64 Thread
V5: Optimized code 32 Alphabet 64 Thread
V6: 256 Buffer 256 Alphabet
V7: 256 Buffer 32 Alphabet
V8: 512 Buffer 256 Alphabet
V9: 512 Buffer 32 Alphabet
• Design Choices and More Experiments
  • 33% of the time goes to recalculating alphabet strings
  • Asynchronous memory copies & Concurrent kernel

• Decompression & Compression Ratio
  • Same as CULZSS since we keep the same buffer sizes
Our Approach:

• Identify the relatively regular data parallel executions
• Identify algorithms that lead to regular executions
• CPU-GPU heterogeneous model
  • Consider both CPU and GPU capabilities
• Re-design algorithms

• First bit-parallel solution for lossless data compression
• Improved performance using bit-vector approach
CULZSS-Bit: A Bit-Vector Algorithm for Lossless Data Compression on GPGPUs

Q & A

Thanks to

INDIANA UNIVERSITY

NVIDIA Hardware Request Program