Light Loss-Less Data Compression, With GPU Implementation

A novel lossless compression method for high GPU parallelization

Shunji Funasaka, Koji Nakano, and Yasuaki Ito

Department of Information Engineering, Hiroshima University
Higashihiroshima, Japan

Hello everyone. I am . . . ,
an undergraduate student of . . . .
It is a pleasure to present the paper by . . .
from Hiroshima University, Hiroshima, Japan,
titled . . . .

Motivation

- Data compression is essential for modern computing.
- Lossless compression: Recovers original data exactly.
- Need for high-speed decompression in data-intensive applications.

Motivation
- Data compression is essential for modern computing.
- Lossless compression: Recovers original data exactly.
- Need for high-speed decompression in data-intensive applications.

From today's perspective, data compression is very crucial. We generate vast amounts of data daily, as well as, silicon chip prices are rising. Therefore, we need to reduce our data without losing any details.

At the same time, it is also important to retreve the desired data from it's compressed form.

Problem Statement

- **Issue:** Dictionary-based methods update dictionaries sequentially.
- **Challenge:** Hard to parallelize for high-performance decompression on GPUs.
- **Goal:** Design a lossless compression method that allows massive parallelization on GPU.

Problem Statement
- **Issue:** Dictionary-based methods update dictionaries sequentially.
- **Challenge:** Hard to parallelize for high-performance decompression on GPUs.
- **Goal:** Design a lossless compression method that allows massive parallelization on GPU.

So, what are the problems in this domain?
Most compression algorithms are dictionary-based, which have a sequential nature.
This nature makes it difficult to parallelize the algorithm.
To reduce time and benefit from the massive parallel processing power of GPUs, we need a lossless compression system that can be parallelized.

Existing Works / Literature Review

- **LZSS/LZW:** Classic dictionary-based methods; hard to parallelize.
- **Huffman Coding/Arithmetic Coding:** Entropy-based; high compression but computationally expensive for GPUs.
- **Gzip/Zlib:** Industry standards; CPU-centric design.
- **Previous GPU Efforts:** Focus on specific data types or limited speedups due to sequential dependencies.

Existing Works / Literature Review
- LZSS/LZW: Classic dictionary-based methods; hard to parallelize.
- Huffman Coding/Arithmetic Coding: Entropy-based; high compression but computationally expensive for GPUs.
- Gzip/Zlib: Industry standards; CPU-centric design.
- Previous GPU Efforts: Focus on specific data types or limited speedups due to sequential dependencies.

existing methods and their limitations in the context of GPU parallelization...

Main Contribution

- **LLL (Light Loss-Less):** A new lossless compression method.
- Specifically designed for **high parallelization** on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves **91.1–176**× speedup over CPU implementation.

Main Contribution
- LLL (Light Loss-Less): A new lossless compression method.
- Specifically designed for high parallelization on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves 91.1–176× speedup over CPU implementation.

New compression method proposed by the authors.

...

Main Contribution

- **LLL (Light Loss-Less):** A new lossless compression method.
- Specifically designed for **high parallelization** on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves $\mathbf{91.1–176}\times$ speedup over CPU implementation.

Main Contribution
- LLL (Light Loss-Less): A new lossless compression method.
- Specifically designed for high parallelization on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves 91.1–176× speedup over CPU implementation.

New compression method proposed by the authors.

...

Main Contribution

- **LLL (Light Loss-Less):** A new lossless compression method.
- Specifically designed for **high parallelization** on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves **91.1–176**× speedup over CPU implementation.

Main Contribution

- **LLL (Light Loss-Less):** A new lossless compression method.
- Specifically designed for **high parallelization** on GPUs.
- Optimized for scenarios where decompression happens more frequently than compression (e.g., loading images or game assets).
- Achieves **91.1–176**× speedup over CPU implementation.

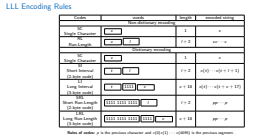
New compression method proposed by the authors.

...

LLL Encoding Rules

Codes	words	length	encoded string
Non-dictionary encoding			
SC Single Character	c	1	c
RL Run-Length	c l	$l+2$	$cc\cdots c$
Dictionary encoding			
SC Single Character	c	1	c
SI Short Interval (2-byte code)	t l	$l+2$	$x(t)\cdots x(t+l+1)$
LI Long Interval (3-byte code)	t 1111 c	$c+18$	$x(t)\cdots x(t+c+17)$
SRL Short Run-Length (2-byte code)	1111 1111 1111 l	$l+2$	$pp\cdots p$
LRL Long Run-Length (3-byte code)	1111 1111 1111 1111 c	$c+18$	$pp\cdots p$

Rules of codes: p is the previous character and $x(0)x(1)\cdots x(4095)$ is the previous segment.



Uses the previous segment (4096 bytes) as a dictionary.

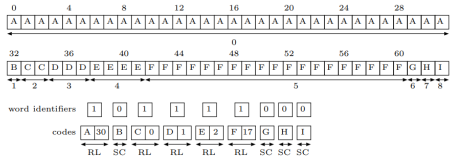
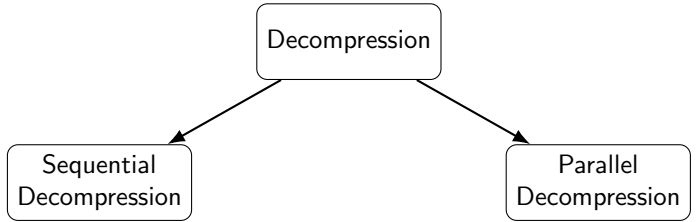


Figure: Examples of LLL-compressed data for two segments with 64 characters each.

Decompression Overview



Decompression Overview

This work has both sequential and parallel decompression technique.

Sequential Decompression Algorithm

- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.

Lets see the sequential one first.

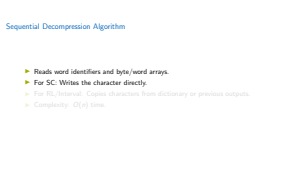
. . .

This gives the complexity of O(n).

Sequential decompression is $O(n)$, which is already fast, but we want even more speed via parallelization.

Sequential Decompression Algorithm

- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.



Lets see the sequential one first.

. . .

This gives the complexity of O(n).

Sequential decompression is $O(n)$, which is already fast, but we want even more speed via parallelization.

Sequential Decompression Algorithm

- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.

Sequential Decompression Algorithm
- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.

Lets see the sequential one first.

...

This gives the complexity of O(n).

Sequential decompression is $O(n)$, which is already fast, but we want even more speed via parallelization.

Sequential Decompression Algorithm

- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.

Sequential Decompression Algorithm
- Reads word identifiers and byte/word arrays.
- For SC: Writes the character directly.
- For RL/Interval: Copies characters from dictionary or previous outputs.
- Complexity: $O(n)$ time.

Lets see the sequential one first.

...

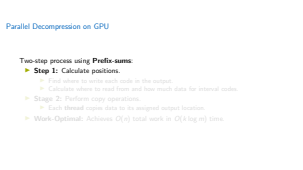
This gives the complexity of O(n).

Sequential decompression is $O(n)$, which is already fast, but we want even more speed via parallelization.

Parallel Decompression on GPU

Two-step process using **Prefix-sums**:

- **Step 1:** Calculate positions.
  - Find where to write each code in the output.
  - Calculate where to read from and how much data for interval codes.
- **Stage 2:** Perform copy operations.
  - Each **thread** copies data to its assigned output location.
- **Work-Optimal:** Achieves $O(n)$ total work in $O(k \log m)$ time.

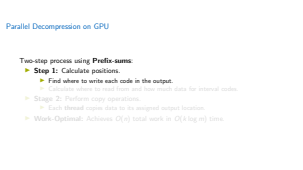


Prefix-sum determines where each parallel thread should write its data in the secondary storage.

Parallel Decompression on GPU

Two-step process using **Prefix-sums**:

- **Step 1:** Calculate positions.
  - Find where to write each code in the output.
  - Calculate where to read from and how much data for interval codes.
- **Stage 2:** Perform copy operations.
  - Each **thread** copies data to its assigned output location.
- **Work-Optimal:** Achieves $O(n)$ total work in $O(k \log m)$ time.

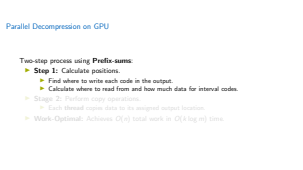


Prefix-sum determines where each parallel thread should write its data in the secondary storage.

Parallel Decompression on GPU

Two-step process using **Prefix-sums**:

- **Step 1:** Calculate positions.
  - Find where to write each code in the output.
  - Calculate where to read from and how much data for interval codes.
- **Stage 2:** Perform copy operations.
  - Each **thread** copies data to its assigned output location.
- **Work-Optimal:** Achieves $O(n)$ total work in $O(k \log m)$ time.

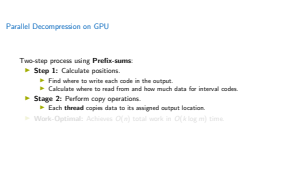


Prefix-sum determines where each parallel thread should write its data in the secondary storage.

Parallel Decompression on GPU

Two-step process using **Prefix-sums**:

- **Step 1:** Calculate positions.
  - Find where to write each code in the output.
  - Calculate where to read from and how much data for interval codes.
- **Stage 2:** Perform copy operations.
  - Each **thread** copies data to its assigned output location.
- **Work-Optimal:** Achieves $O(n)$ total work in $O(k \log m)$ time.

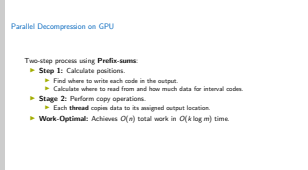


Prefix-sum determines where each parallel thread should write its data in the secondary storage.

Parallel Decompression on GPU

Two-step process using **Prefix-sums**:

- **Step 1:** Calculate positions.
  - Find where to write each code in the output.
  - Calculate where to read from and how much data for interval codes.
- **Stage 2:** Perform copy operations.
  - Each **thread** copies data to its assigned output location.
- **Work-Optimal:** Achieves $O(n)$ total work in $O(k \log m)$ time.



Prefix-sum determines where each parallel thread should write its data in the secondary storage.

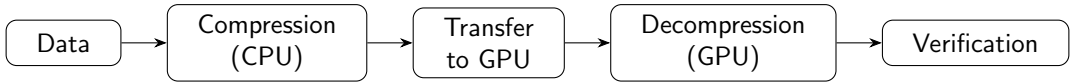
Experiment Setup

- **GPU:** NVIDIA GeForce GTX 1080 (8GB VRAM).
- **CPU:** Intel Core i7-4790 (3.60 GHz).
- **Software:** CUDA Toolkit, C++ Compiler.
- **Dataset:** Standard benchmark images and data (e.g., Crafts, Flowers, Graph, Random, Black).

Experiment Setup
- **GPU:** NVIDIA GeForce GTX 1080 (8GB VRAM).
- **CPU:** Intel Core i7-4790 (3.60 GHz).
- **Software:** CUDA Toolkit, C++ Compiler.
- **Dataset:** Standard benchmark images and data (e.g., Crafts, Flowers, Graph, Random, Black).

This work was conducted in 2016, when the NVIDIA GeForce GTX 1080 was the flagship GPU used in the experiments.

Experiment Steps



1. Compress input data using LLL on CPU.
2. Transfer compressed data from Host to Device.
3. Measure decompression time on GPU for different datasets.
4. Compare with sequential CPU decompression and other methods (LZSS, LZW).

Experiment Steps
Data
Compression (CPU)
Transfer to GPU
Decompression (GPU)
Verification
1. Compress input data using LLL on CPU.
2. Transfer compressed data from Host to Device.
3. Measure decompression time on GPU for different datasets.
4. Compare with sequential CPU decompression and other methods (LZSS, LZW).

So, how this expreriment was conducted?...

Performance Evaluation

- **Hardware:** GeForce GTX 1080 GPU vs. Core i7-4790 CPU.
- **Dataset:** Five grayscale images (Crafts, Flowers, Graph, Random, Black).
- **Compression Ratio:** Comparable to LZW and LZSS.
- **Decompression Speed (GPU):**
  - LLL is **2.49–9.13**× faster than LZW.
  - LLL is **4.30–14.1**× faster than LZSS.

Performance Evaluation
- **Hardware:** GeForce GTX 1080 GPU vs. Core i7-4790 CPU.
- **Dataset:** Five grayscale images (Crafts, Flowers, Graph, Random, Black).
- **Compression Ratio:** Comparable to LZW and LZSS.
- **Decompression Speed (GPU):**
  - LLL is **2.49–9.13**× faster than LZW.
  - LLL is **4.30–14.1**× faster than LZSS.

The authors found that . . .

Answering Research Question 1

RQ1: Can LLL achieve significant speedup on GPU while maintaining competitive compression ratios?

- **Yes.** LLL achieves 91–176× speedup over CPU.
- Compression ratio is comparable to standard LZSS/LZW for the tested datasets.

Answering Research Question 1

RQ1: Can LLL achieve significant speedup on GPU while maintaining competitive compression ratios?
- Yes. LLL achieves 91–176× speedup over CPU.
- Compression ratio is comparable to standard LZSS/LZW for the tested datasets.

Here comes our research questions...

The compression ration was similar to the CPU implementations, but the decompression speed was significantly higher on GPU.

Answering Research Question 2

RQ2: How does LLL compare to existing parallel-friendly methods?

- LLL outperforms LZW (2.49–9.13×) and LZSS (4.30–14.1×) in decompression speed on GPU.
- Its prefix-sum based approach is more work-efficient and made parllization possible.

Answering Research Question 2

RQ2: How does LLL compare to existing parallel-friendly methods?
- LLL outperforms LZW (2.49–9.13×) and LZSS (4.30–14.1×) in decompression speed on GPU.
- Its prefix-sum based approach is more work-efficient and made parllization possible.

Discussion / implication

- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

Discussion / implication
- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

Discussion / implication

- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

└Discussion / implication

Discussion / implication

- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

Discussion / implication

- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

Discussion / implication
- **Real-time Application:** Ideal for asset loading and real-time texture decompression.
- **Architecture Synergy:** LLL's design aligns perfectly with GPU execution pattern.
- **Data Transfer Bottleneck:** Highlights that GPU decompression can alleviate the PCIe bottleneck by transferring less data.

Threats to Validity

- **Internal:** Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- **External:** Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the **repetitiveness** of data (common in images but less so in random binary).

Threats to Validity
- Internal: Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- External: Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the repetitiveness of data (common in images but less so in random binary).

Threats to Validity

- **Internal:** Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- **External:** Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the **repetitiveness** of data (common in images but less so in random binary).

Threats to Validity
- Internal: Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- External: Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the repetitiveness of data (common in images but less so in random binary).

Threats to Validity

- **Internal:** Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- **External:** Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the **repetitiveness** of data (common in images but less so in random binary).

Threats to Validity
- **Internal:** Prefix-sum overhead may be more noticeable on very small datasets; Moreover, grayscale image samples were used in the experiment.
- **External:** Results might vary on different GPU architectures (e.g., AMD vs. NVIDIA).
- Performance depends on the **repetitiveness** of data (common in images but less so in random binary).

Conclusion

- LLL provides a highly parallelizable lossless compression scheme.
- Massive acceleration achieved on GPU compared to standard methods.
- Future work could include multi-GPU (cluster) support or hybrid compression schemes.

Conclusion
- LLL provides a highly parallelizable lossless compression scheme.
- Massive acceleration achieved on GPU compared to standard methods.
- Future work could include multi-GPU (cluster) support or hybrid compression schemes.

LLL decompression on the GPU is much faster than previously published LZW decompression and LZSS decompression.

This shows that GPU LLL decompression can be useful for many applications.

Thank You!

Questions?

Thank You!
Questions?

Thank you very much for your time and attention