SJTU SAIL (Software Architecture and Infrastructure Lab)

SHANGHAI JIAO TONG UNIVERSITY

Stream Processing

Yalun Lin Hsu

2020-11-02

SAIL

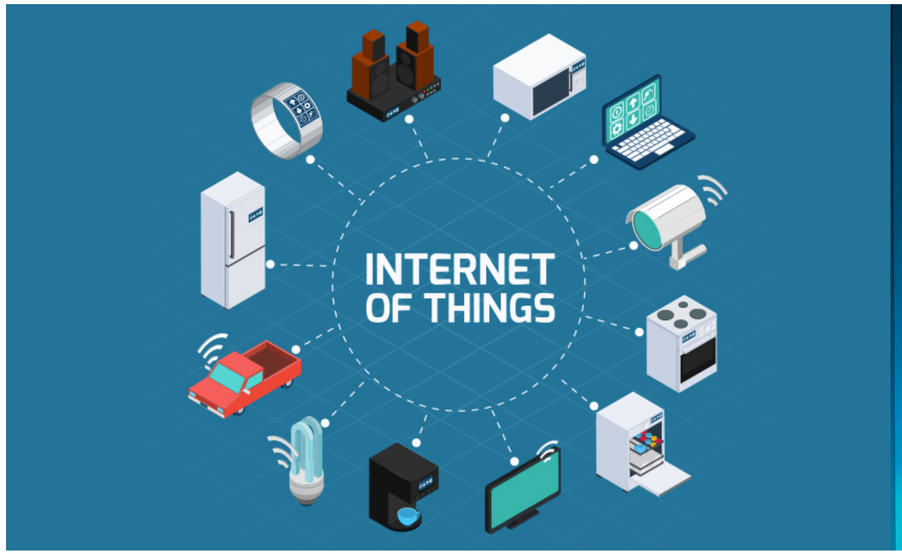
USENIX ATC ’18

TerseCades

Efficient Data Compression in Stream Processing

https://www.usenix.org/system/files/conference/atc18/atc18-pekhimenko.pdf

Streaming Data



IoT **Clouds** **Big Data**

Huge volumes of streaming data with real-time processing requirements
Enormous pressure on the capacity and bandwidth of servers' main memory

Is Data Compression Useful for Streaming?

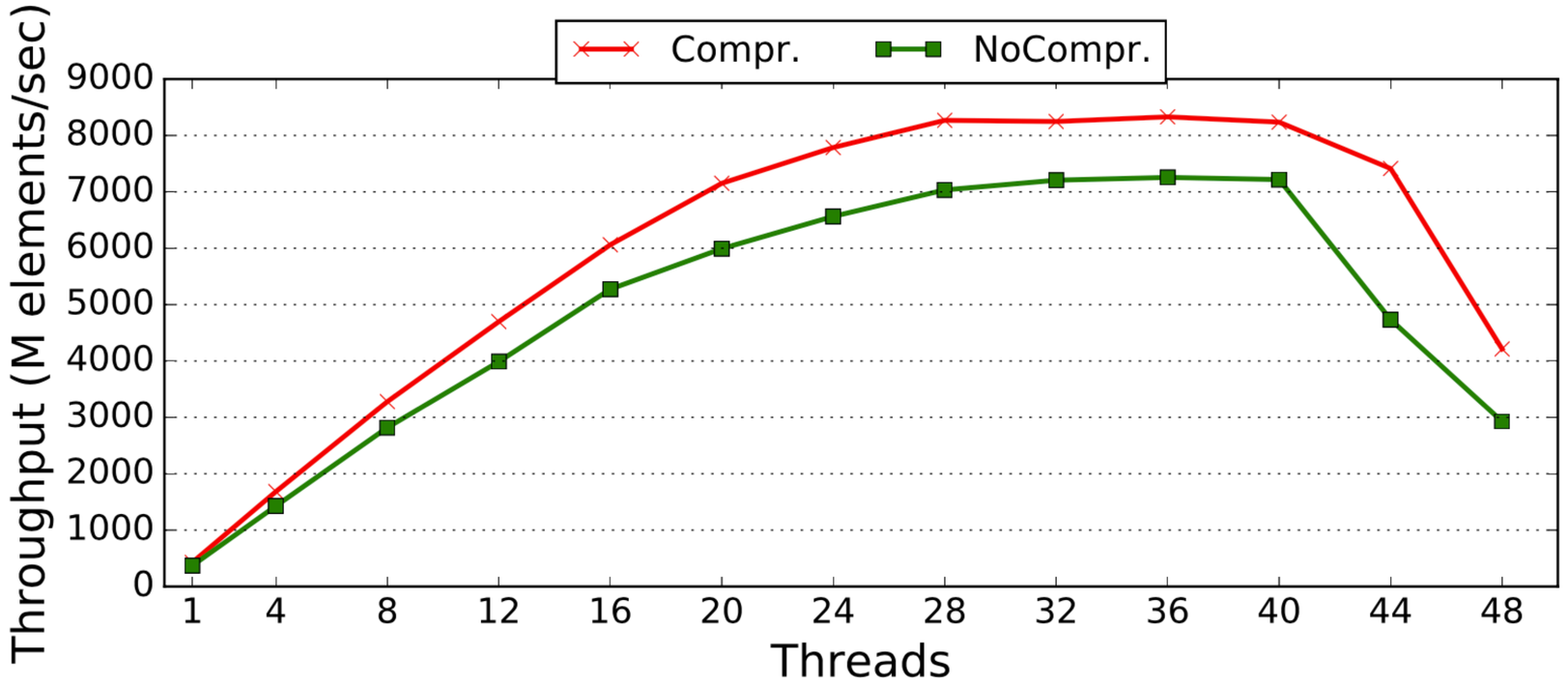
- Intuitively, streaming with simple operators should be **bandwidth-bottlenecked**: either network or memory bandwidth

- Simple single node experiment with the state-of-the-art streaming engine, **Trill**, with the ***Where*** query over large one column 8-byte field:

  E.g., Where (e => e.errorCode != 0)

- Expectation: observe **memory bandwidth** as a major bottleneck

Compressibility ≠> Performance Gain

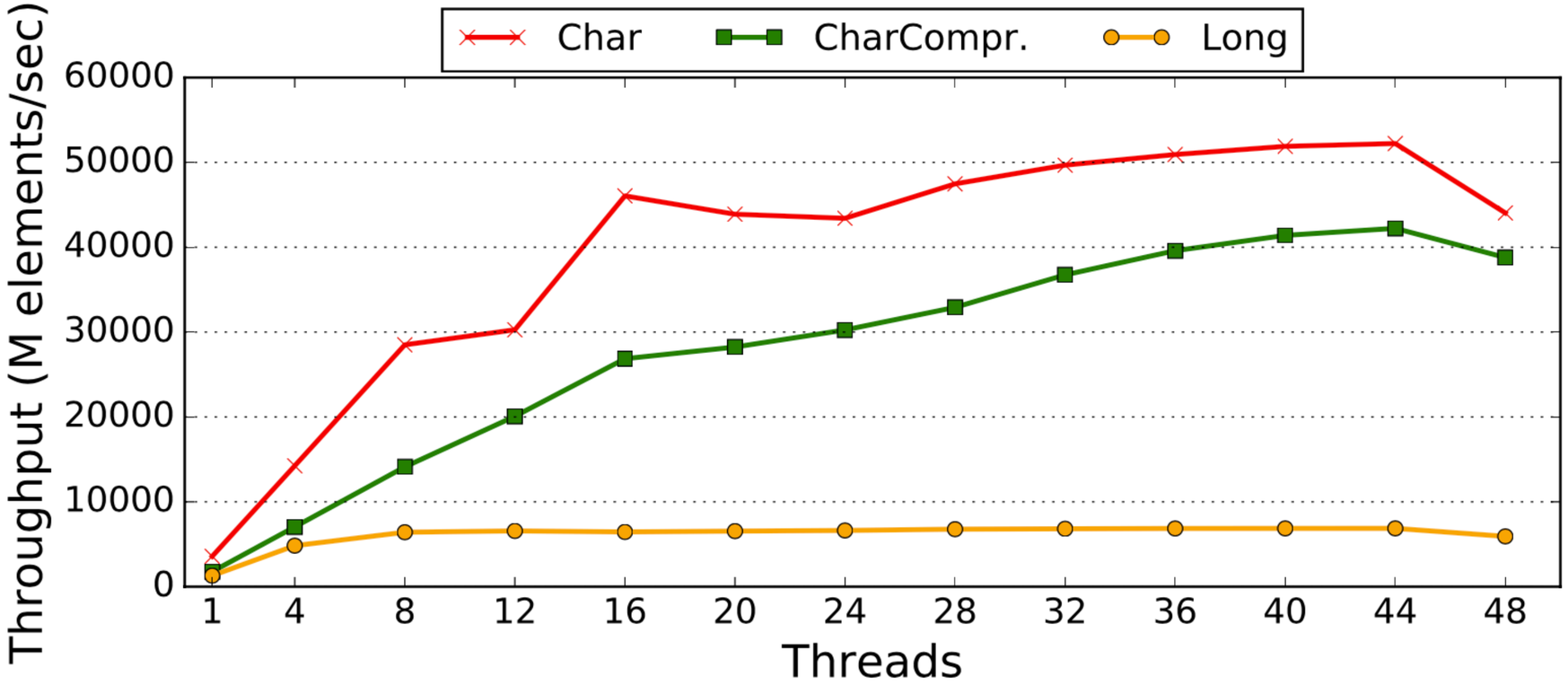


Only 10%-15% performance improvement with 8X compression

What Went Wrong?

- Memory allocation overhead:
  - just-in-time copy of payloads to create a streameable event
- Memory copying and reallocation:
  - enables flexible column-oriented data batches
- Inefficient bit-wise manipulation
- Hash tables manipulations

Compressibility => Performance Gain



If no artificial bottlenecks: performance improvement is close to compression ratio (7.6X speedup with 8X compression)
Up to 6.1X speedup with realistic compression algorithm: Base-Delta Encoding

Prerequisites for Efficient Data Streaming

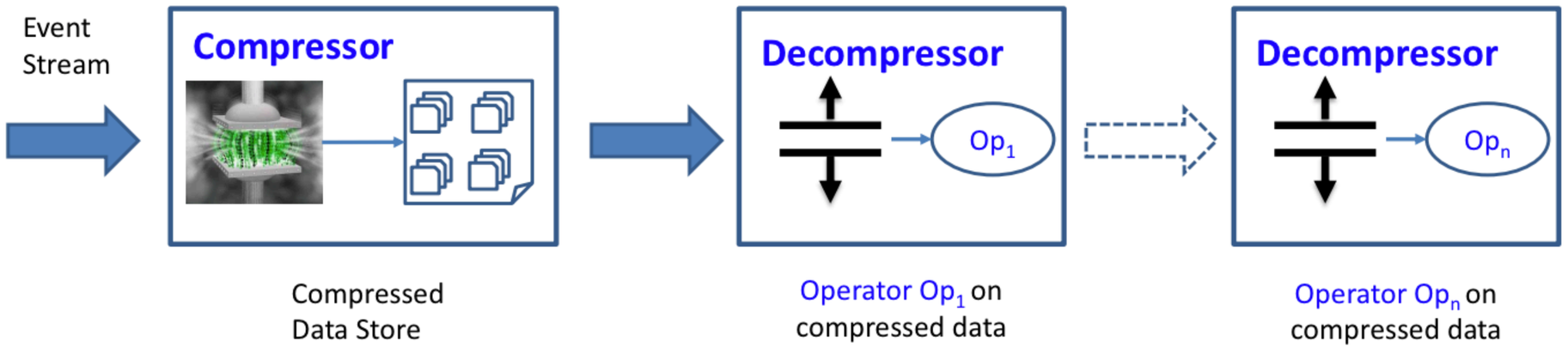
- ✓ *Fixed Memory Allocation*
- ✓ *Efficient HashMap Primitives*
- ✓ *Efficient Filtering Operations (bit-wise manipulations)*

Key Observations

- **Memory bandwidth** becomes the ***major bottleneck*** if streaming is properly optimized

- Dominant part of the data is ***synthetic*** in nature and hence has a lot of **redundancy**
  - Can be exploited through efficient **data compression**

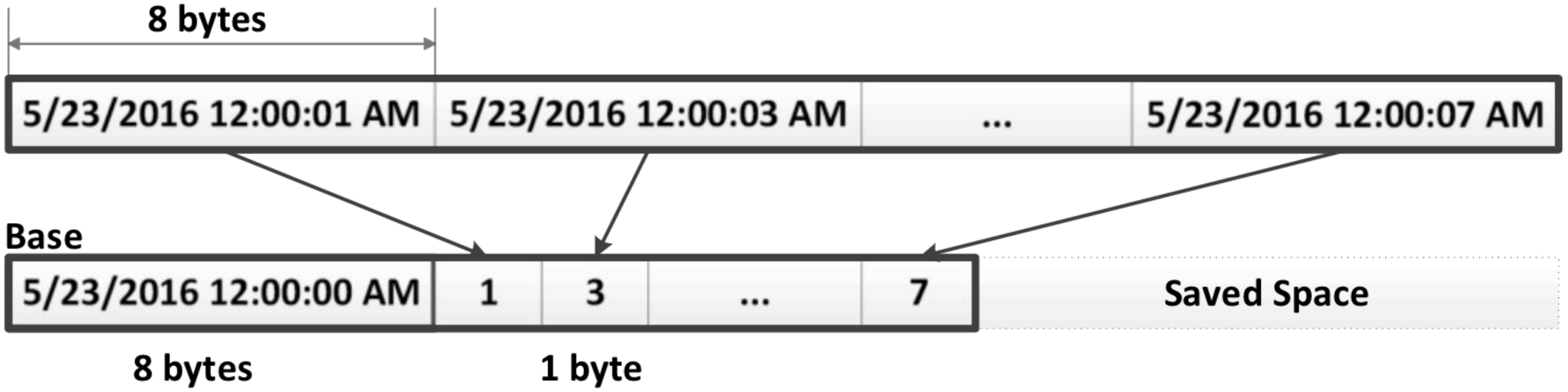
TerseCades: Baseline System Overview



Key Design Choices and Optimizations

- ✓ *Lossless Compression*
  - ✓ *Arithmetic vs. Dictionary-based Compression*
  - ✓ *Decompression is on the critical path*
- ✓ *Lossy Compression without Output Quality Loss*
  - ✓ *Integers and floating points*
- ✓ *Reducing Compression/Decompression Cost*
  - ✓ *Hardware-based acceleration: vectorization, GPU, FPGA*
- ✓ *Direct Execution on Compressed Data*

Lossless Compression: Base-Delta Encoding



✓ **Fast Decompression:** vector addition

✓ **Simple SW/HW Implementations:** arithmetic and comparison

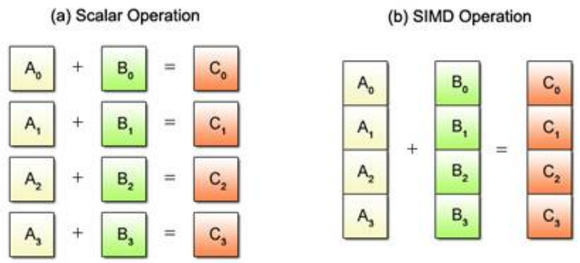
✓ **Effective:** good compression ratio

Lossy Compression Without Output Quality Loss

- **Base-Delta Encoding modification**
  - Truncate deltas when full precision not required

- **ZFP floating point compression engine**
  - Equivalent of BD in floating point domain with controlled precision

Reducing Compression Overhead



SIMD/Vectorization

Intel Xeon with 256-bit SIMD

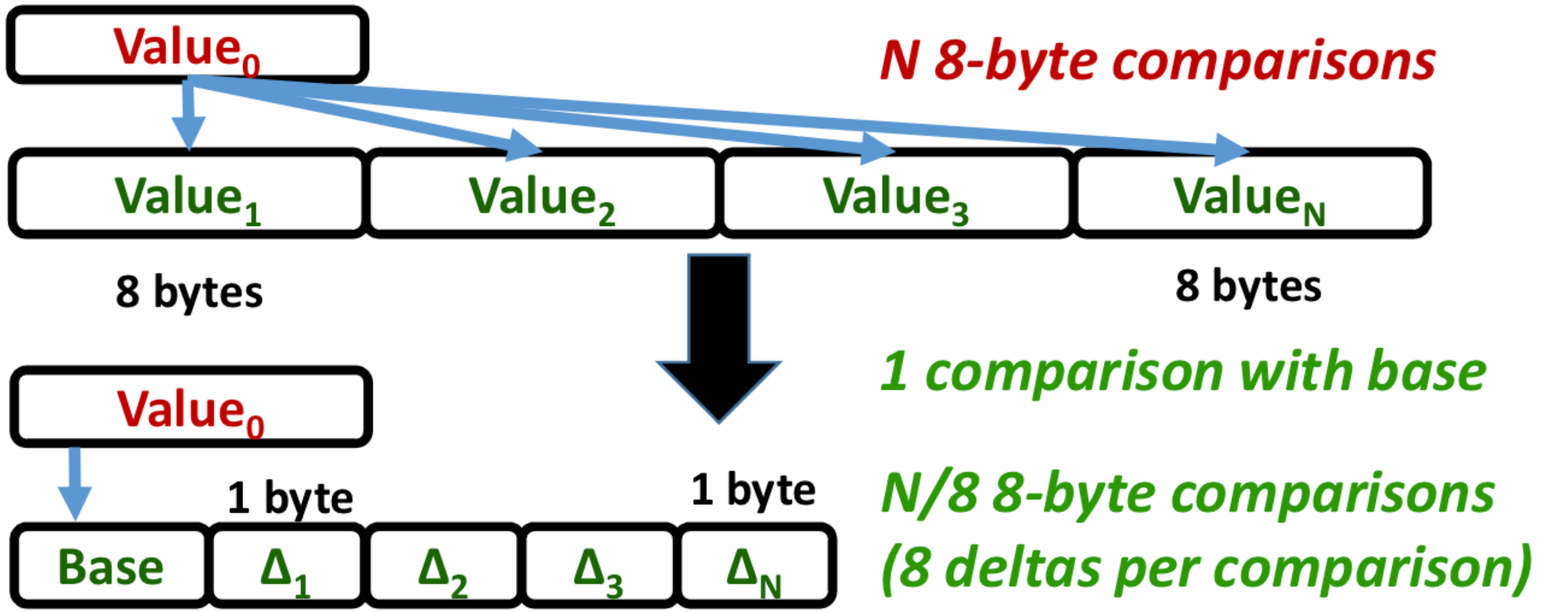
GPU

NVIDIA 1080Ti

FPGA

Altera Stratix V

Execution on Compressed Data



✓*Low Latency*

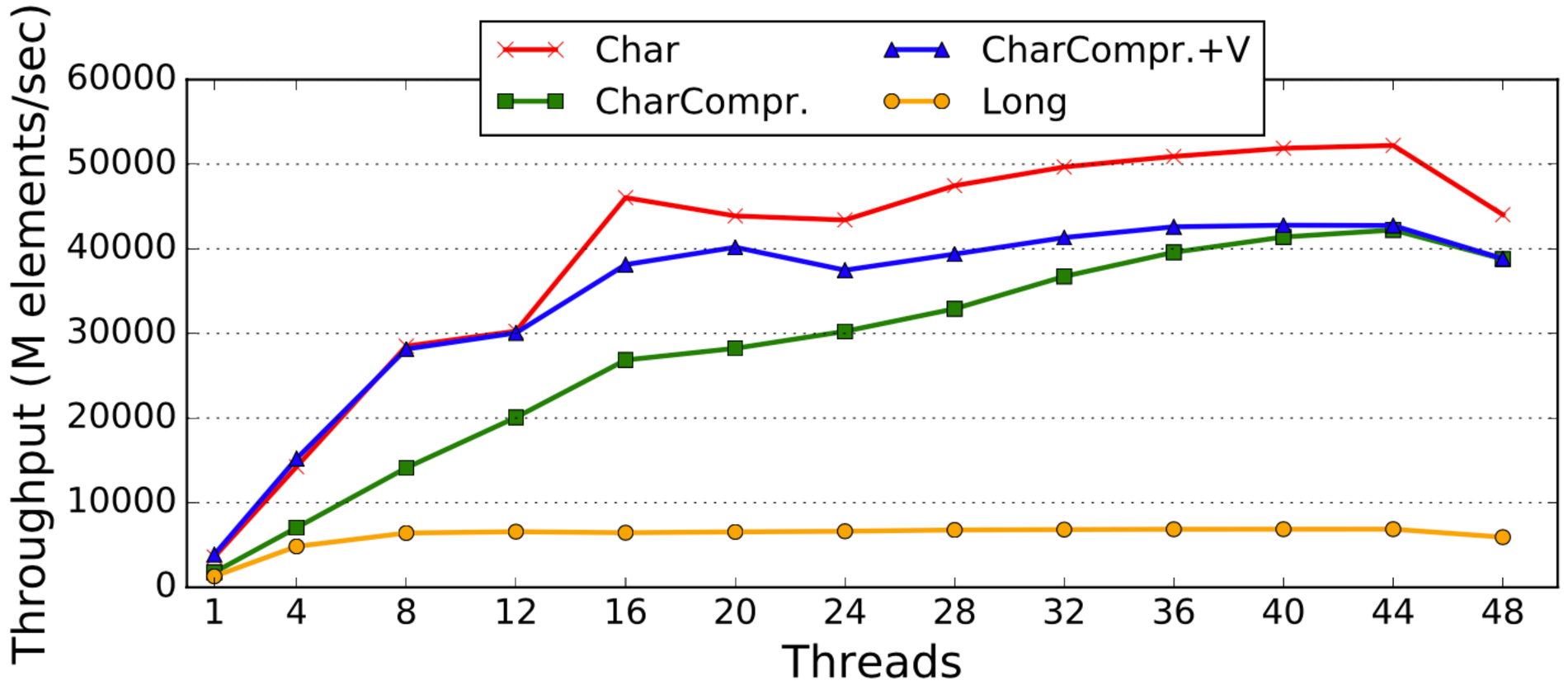
✓*Single Comparison*

✓*Narrower Operations*

Evaluation: Methodology

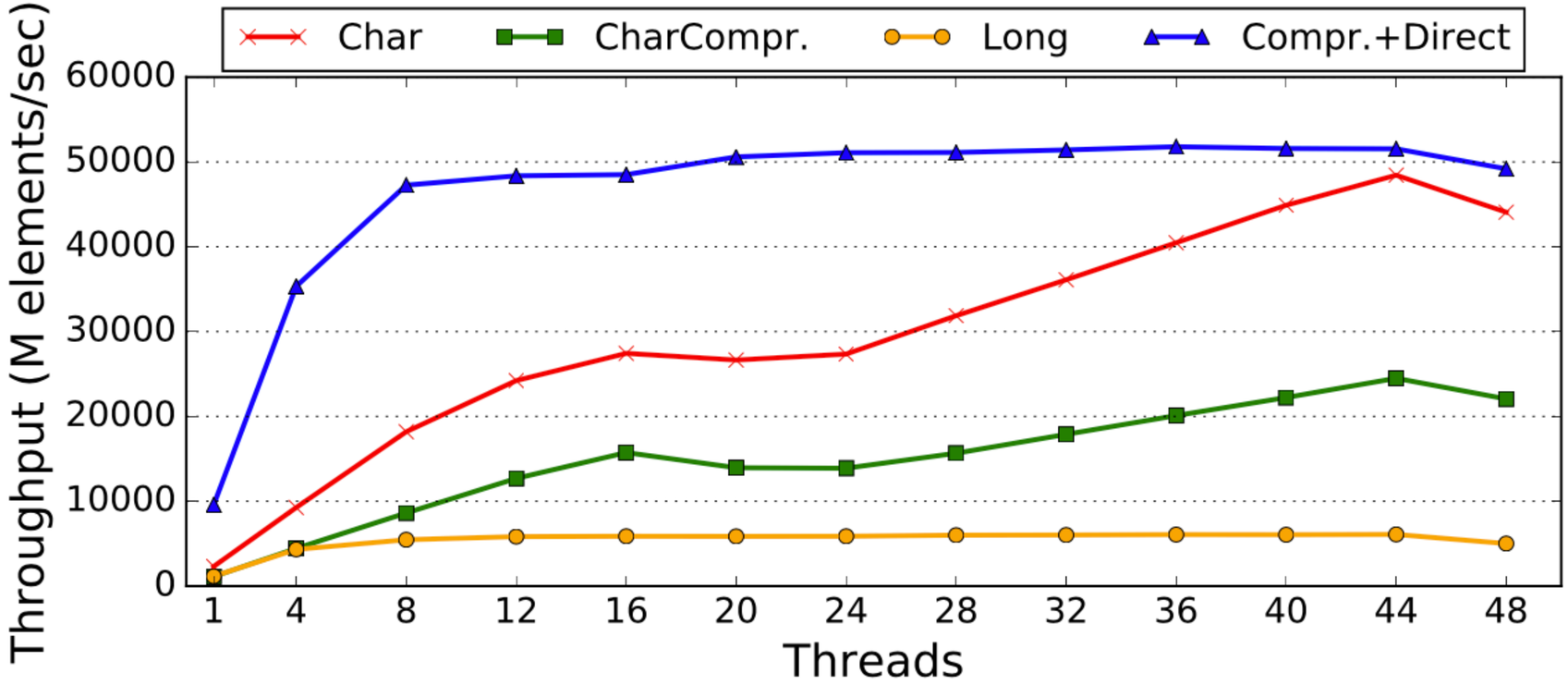
- CPU: 24-core system based on Intel Xeon CPU E5-2673, 2.40GHz with SMT-enabled, and 128GB of memory
- GPU: NVIDIA GeForce GTX 1080 Ti with 11GB of GDDR5X memory
- FPGA: Altera Stratix V FPGA, 200MHz

Benchmark



Vectorization further reduces compression/decompression overhead, especially for smaller number of threads

Benchmark



When direct execution is applicable, it can significantly improve performance as it reduces the total computation

Summary

- Q: Can **data compression** be effective in stream processing?

- A: **Yes**, TerseCades design is the proof-of-concept
  - Properly optimize the baseline system
  - Use light-weight data compression algorithms + HW acceleration
  - **Directly execute** on compressed data