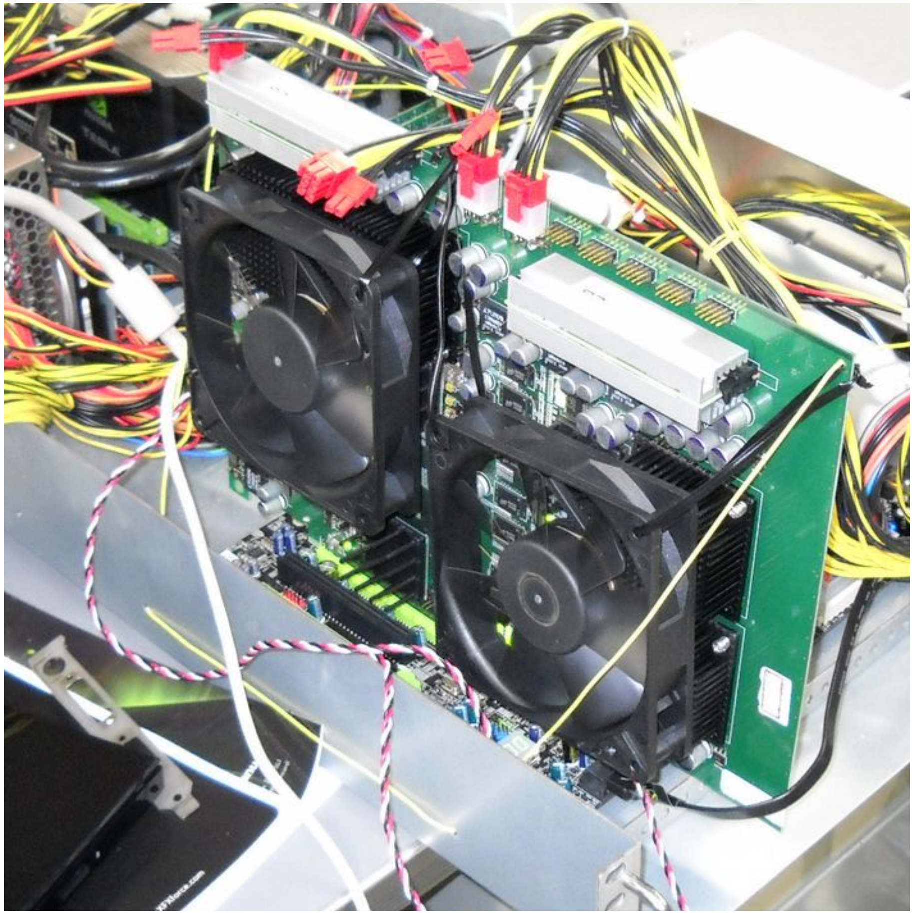
Compressing Floating-Point Number Stream for Numerical Applications

Hisanobu Tomari, Mary Inaba, Kei Hiraki

The University of Tokyo

Recent Computing Systems

- Accelerators (e.g. GRAPE-DR, GPGPU) and cluster systems are widely used
  - Inexpensive
  - High performance
  - Low power consumption



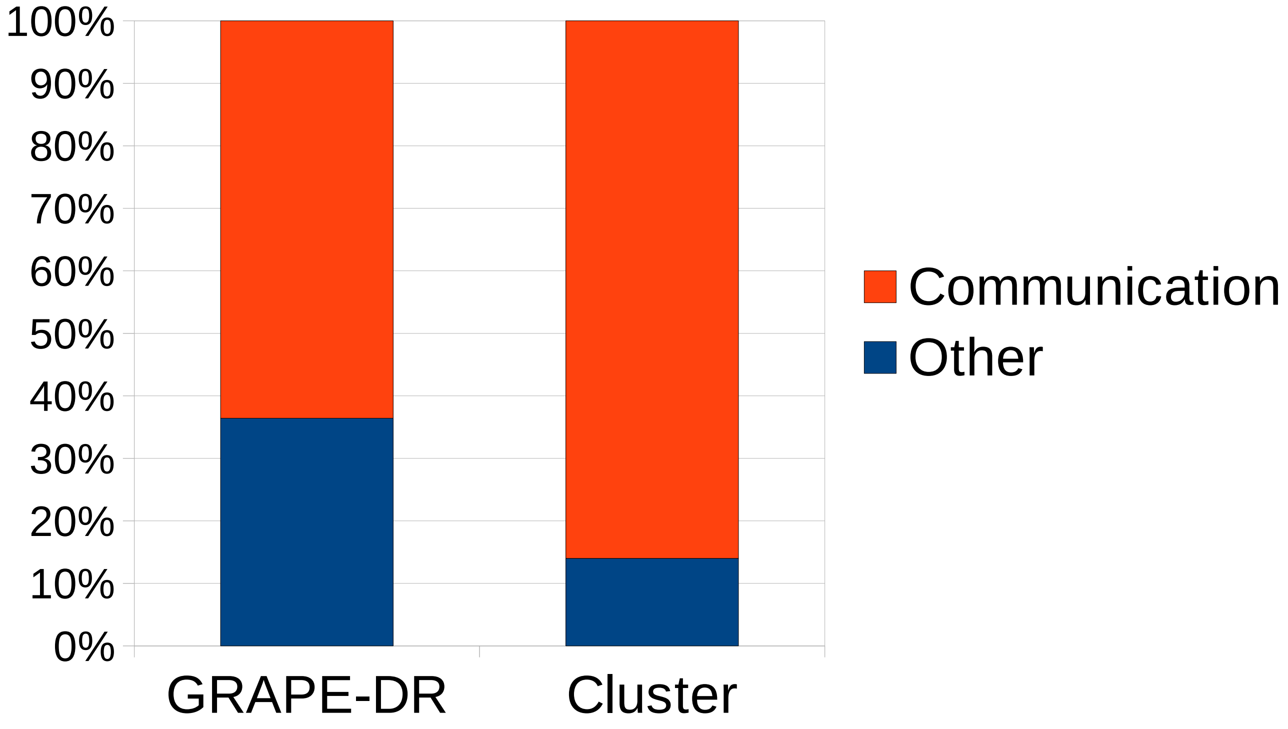
Bandwidth bottleneck

- e.g. Ethernet for cluster
- PCI-express for accelerator

- Processor becomes idle while waiting data

Example of the problem

- Matrix multiplication on GRAPE-DR
- FFT on 8-node cluster (Gigabit Ethernet)



Breakdown of execution time

Memory Wall problem to Network Wall problem

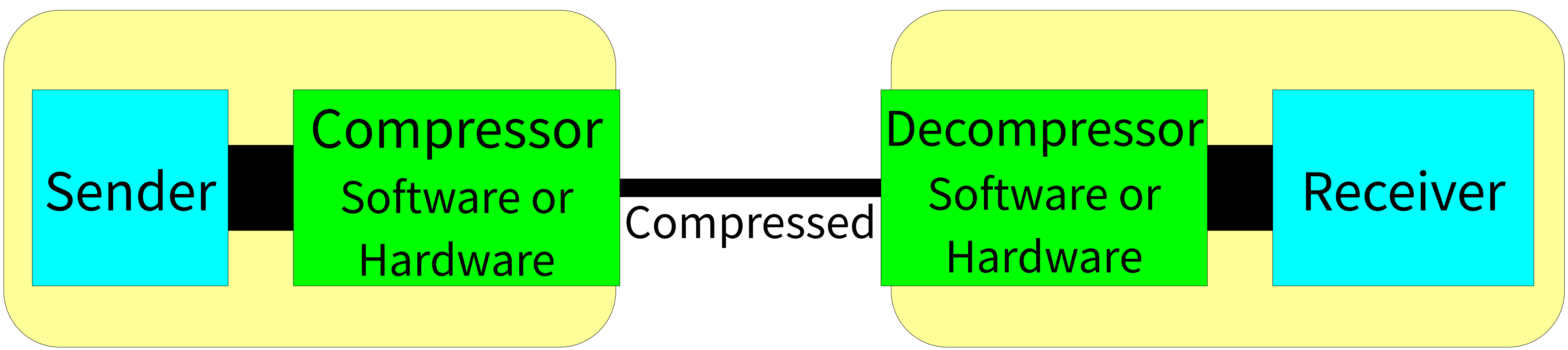
- Memory Wall: Barrier between processor and memory speed goes larger [Wulf+, 95]
  - #I/O pads limited in a chip
  - Processor still becomes faster
- Same constraints also apply to interconnection

Existing Methods

- Increase cache size
  - Dataset is unlikely to be reused in accelerators or cluster systems
- Modify algorithms so that it fit narrow-bandwidth systems
  - Not always possible or wanted

Our proposal: Compression

- Compress data to transfer
- Add pair of compressor/decompressor to each end of data channel



Design issues

- Use software or hardware to increase bandwidth
- Only small software modification needed
- Adds small extra latency
- High bandwidth compression required
  - Current compression algorithms are too slow
    - Gzip, Bzip2

Compression of floating point numbers

- Scientific programs transfer **floating-point numbers**
  - Input and output of board almost exclusively fp numbers on accelerators
  - Many scientific simulations transfer **double-precision fp** numbers between nodes or boards

Floating Point Number Format

- IEEE 754-2008 double-precision FP number

$$v = (-1)^s \cdot 2^{exponent - offset} \cdot (1.\text{mantissa})$$

Redundancy in notation of FP Numbers

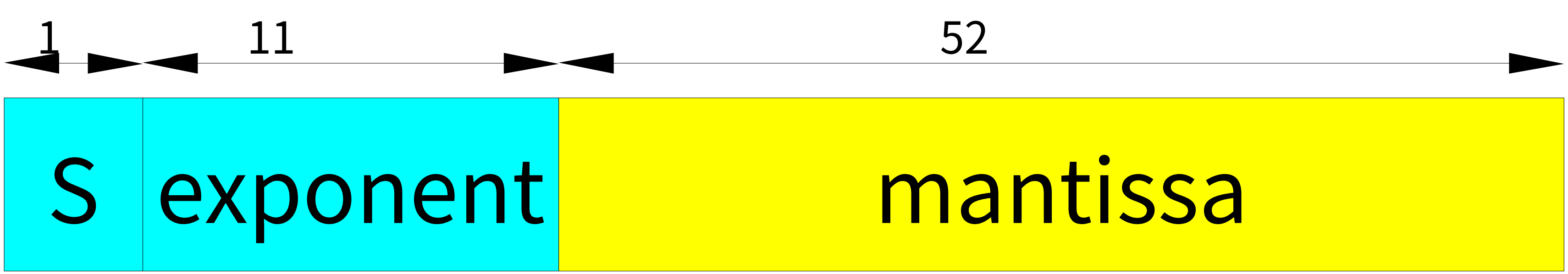
- Exponent parts in array take similar values in real-world simulation applications
- Too distant exponent parts lead to loss of significant digits

Example: 3.1E256 + 1.3E0 = 3.1E256

Lost digits

Our algorithm: **MAF**

- Compress sign and exponent part
- Leave mantissa part unmodified
- Can be implemented on hardware
- Compress 4 FP numbers together
  - Increases compression ratio



Compressed Sign and Exponent Formats

- Keep recently sent sign and exponent parts in memory
  - data=history[map[i]] 0≦i<0xF

i

  - data=history[map[i]]+diff 0≦i,diff<0xF

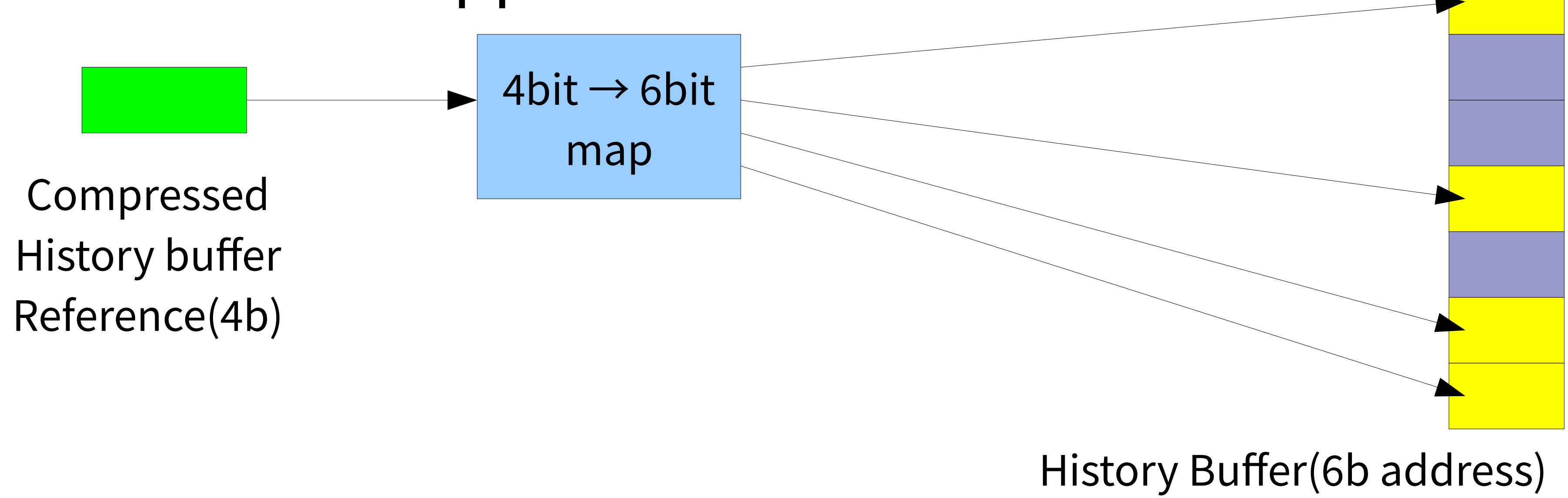
0xF	i	diff

  - otherwise

0xF	0xF				

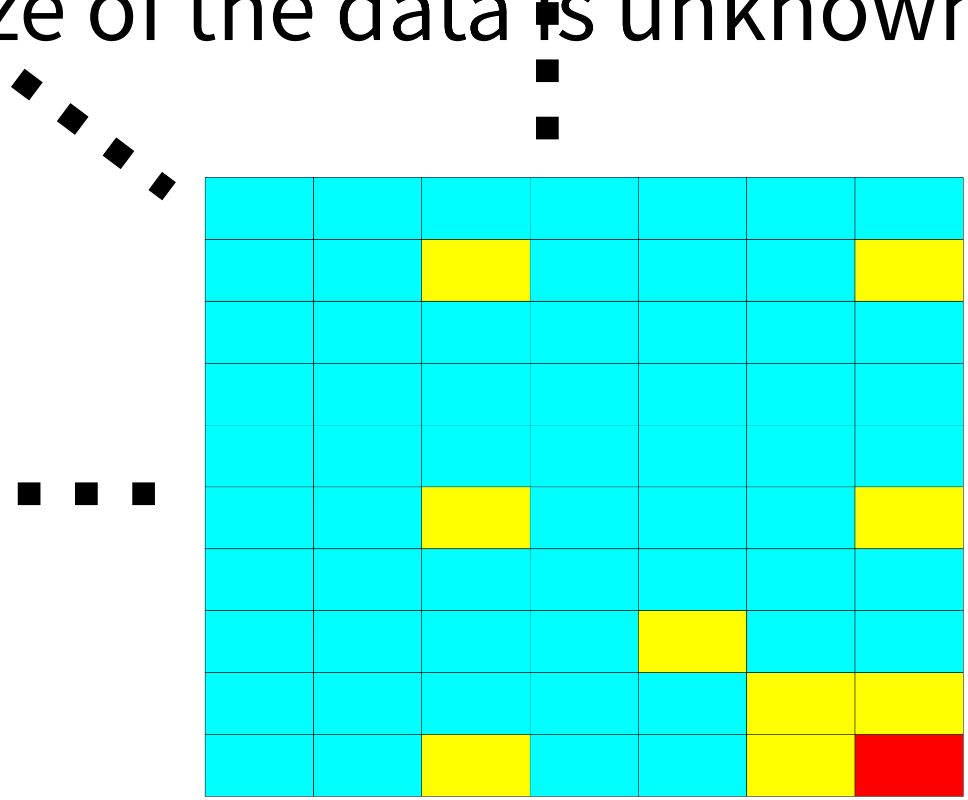
Exploit locality in data structure

- Search history memory for similar value
  - Use look-up table to map **4-bit compressed address** to **6-bit history buffer address**
  - Designed to match **multi-dimensional structures** in applications



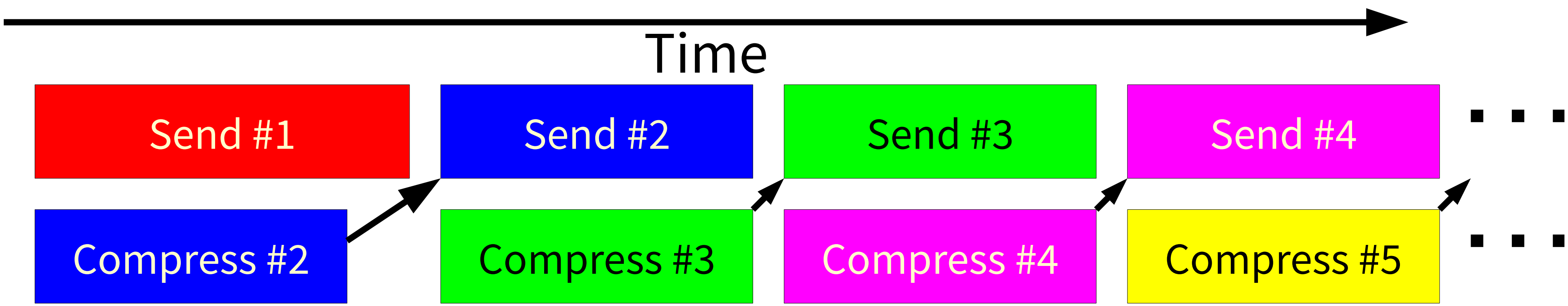
Example of locality: matrix

- Similar value to the red value is expected at yellow places
- Exact size of the data is unknown



Compression: Overlapping

- Compress next segment while sending a segment
  - Sending/Compressing overlapped



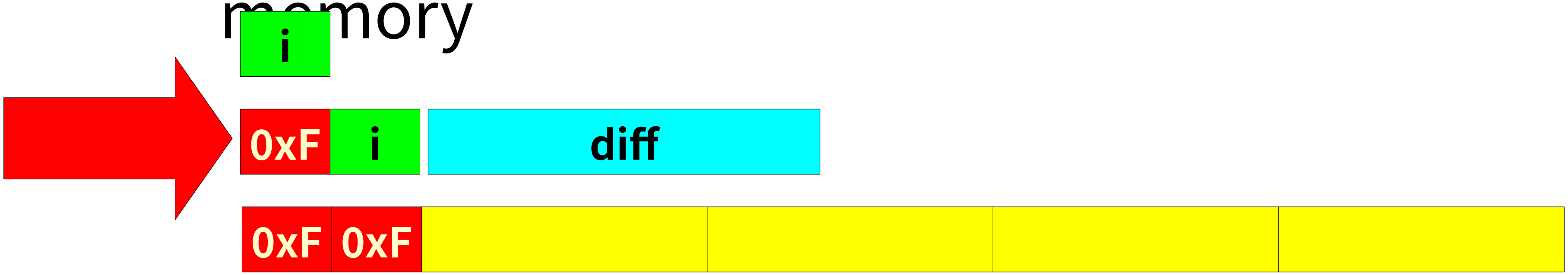
Decompression (Format 1)

If a record starts with nibble(4b) 0x0-0xE then it's in format 1

- Read history memory
- Concatenate with mantissa parts
- Keep sign and exponent part in history memory

Decompression (Format 2)

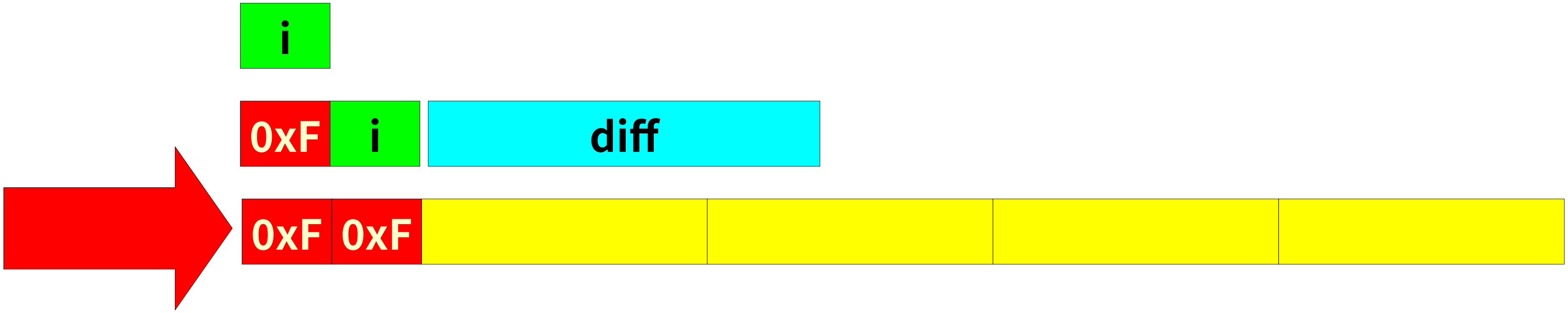
- If a record starts with a byte 0xF0-0xFE then it's in format 2
  - Read history memory, add difference
  - Concatenate with mantissa parts
  - Keep resulting sign and exponent part in history memory



Decompression (Format 3)

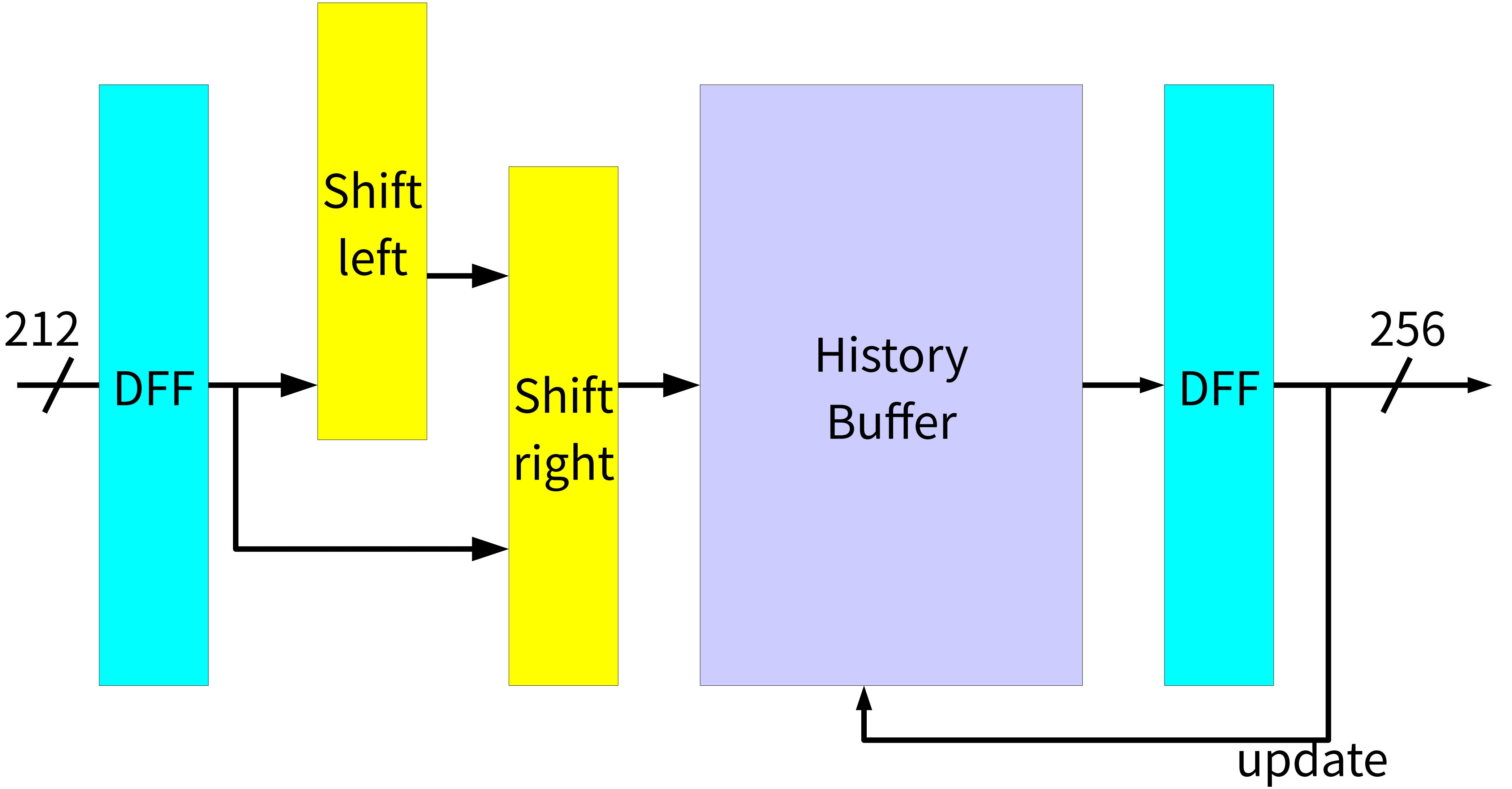
If a record starts with a byte 0xFF then it's in format 3

- Concatenate with mantissa parts
- Keep sign and exponent part in history memory



Hardware Decompression Pipeline

- 6.4GB/s on FPGA(Xilinx Virtex-5, 240 MHz)



Evaluation

- Performance of scientific application
  - Matrix multiplication on GRAPE-DR
  - Fast Fourier Transform on cluster
- Compression ratio
- Compression speed

Data for Evaluation

- Compression ratio/speed depend on input
- Input from actual scientific simulations are used

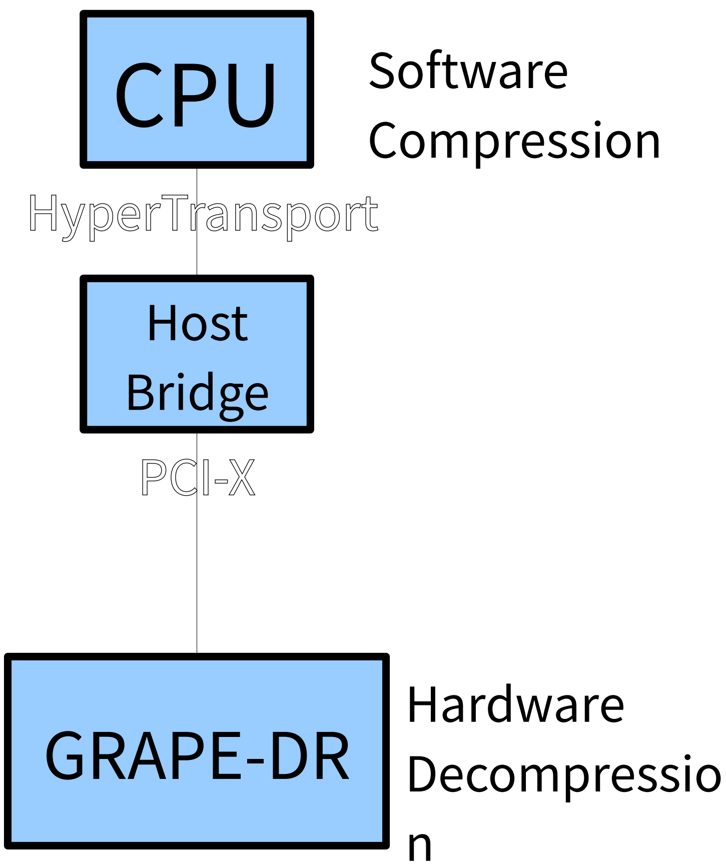
- Random input: bad case for compression
  - Used same initialization as FT in NAS Parallel Benchmarks

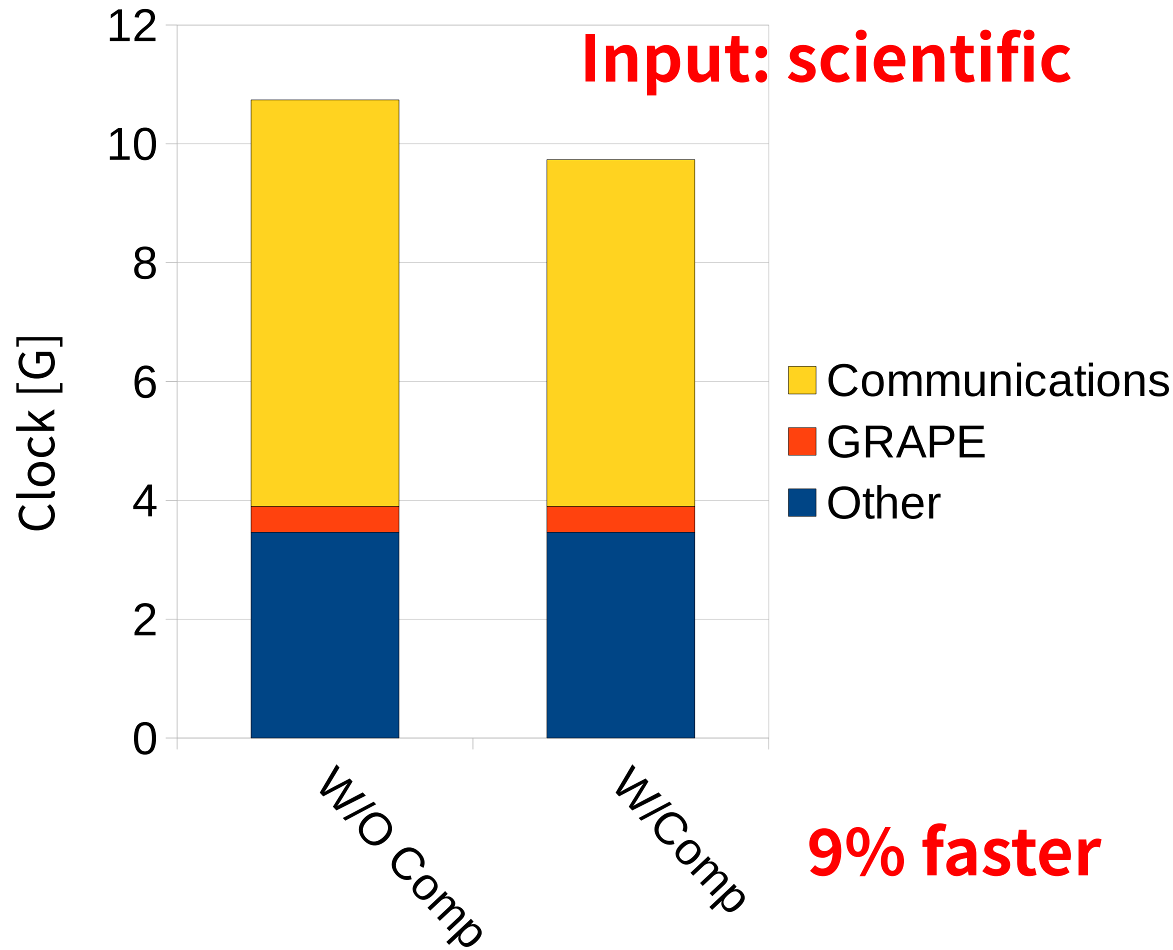
Hardware Configuration

- Accelerator System
  - 2*Opteron 2.6 GHz
  - 4 GB DDR
  - Linux 2.6.12
  - GRAPE-DR PCI-X
- Cluster
  - CPU: 2*Xeon E5530
  - 8-node
  - 12 GB DDR3
  - Linux 2.6.18
  - Gigabit Ethernet

Speed Increase on Accelerator

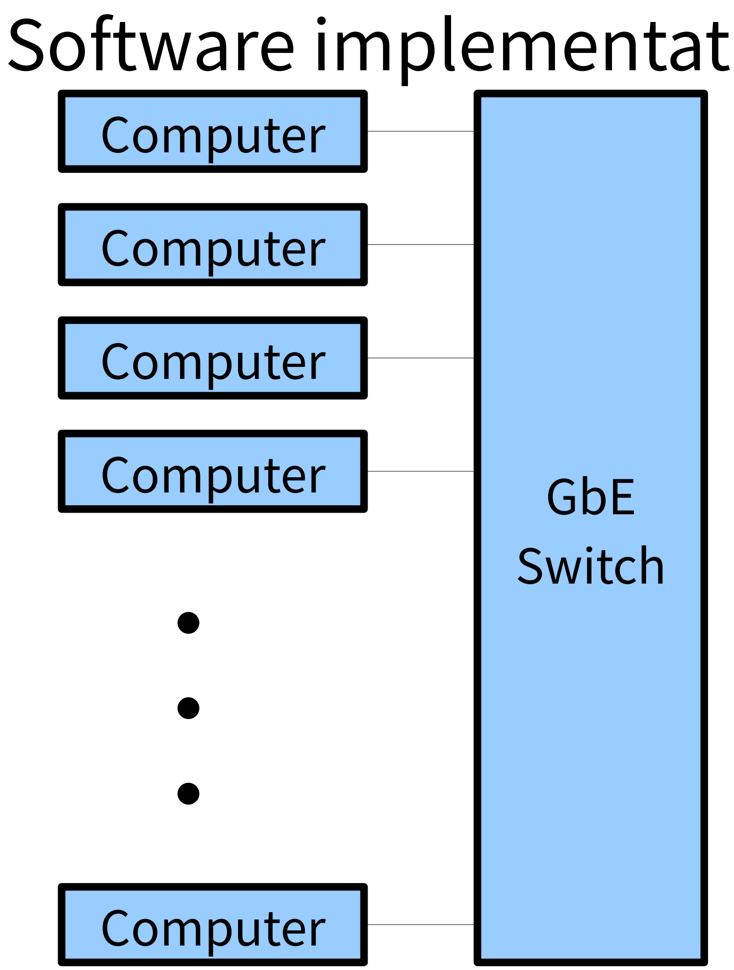
- Matrix multiply on GRAPE-DR system

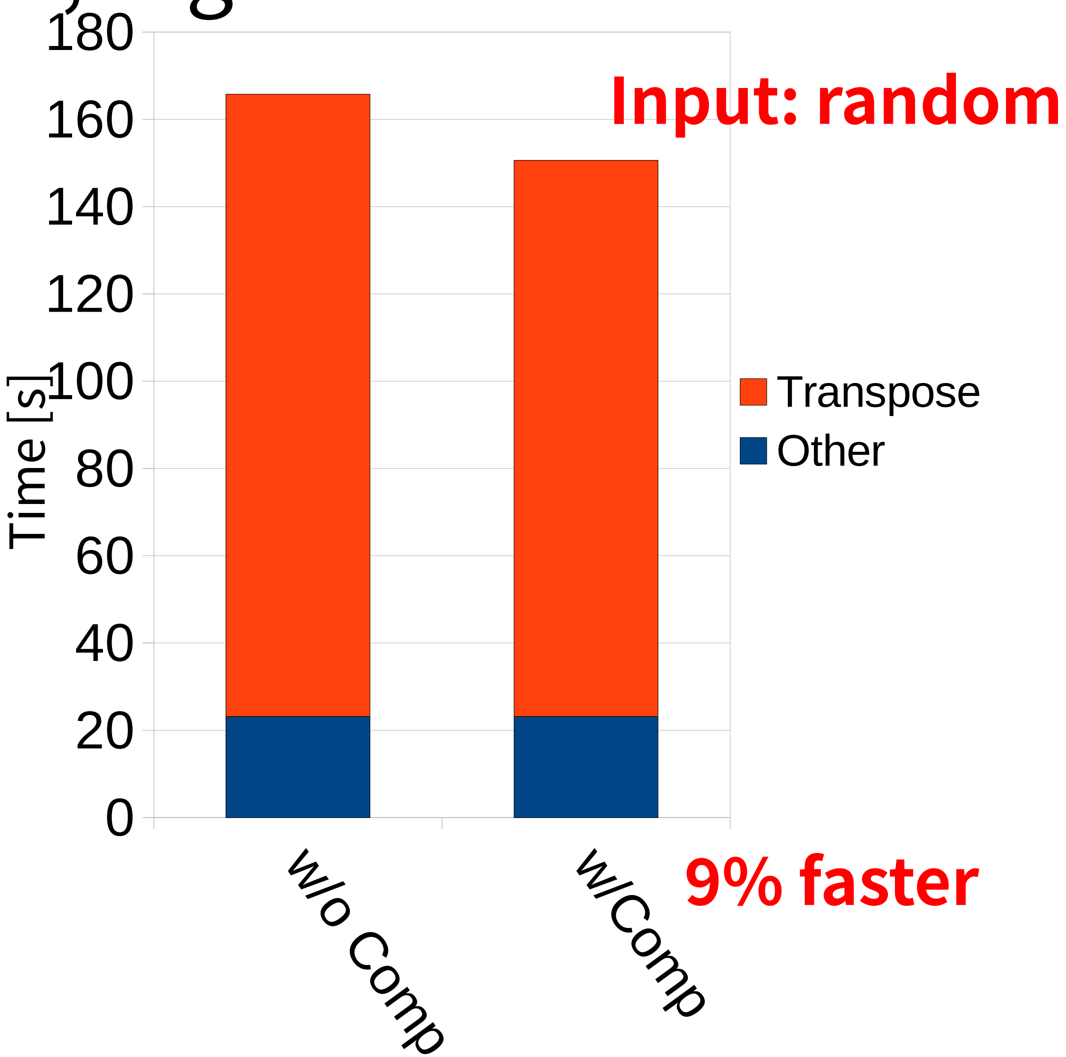




Speed Increase on Cluster

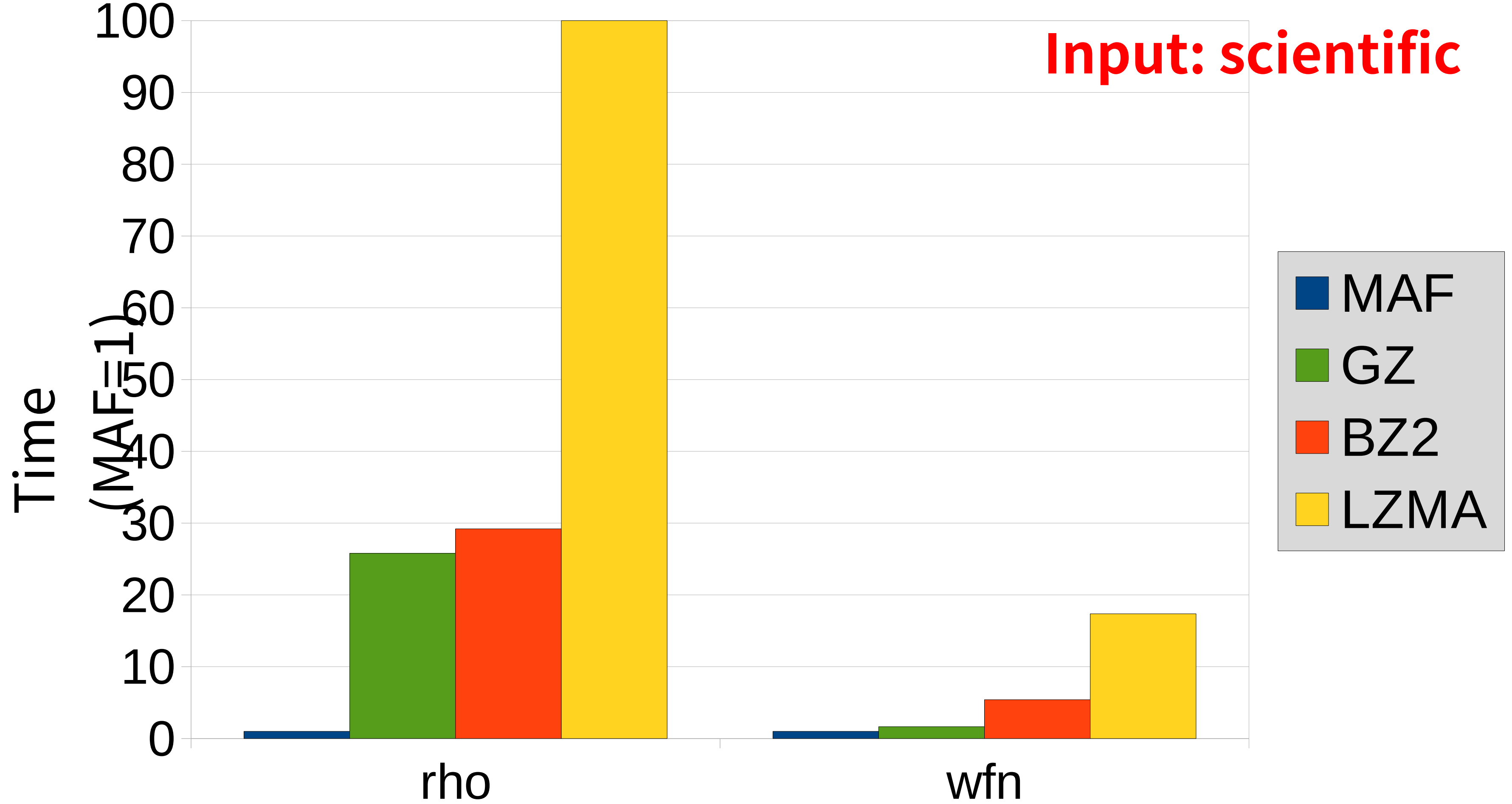
- FFT, 8*DP Xeon E5530, Gigabit Ethernet





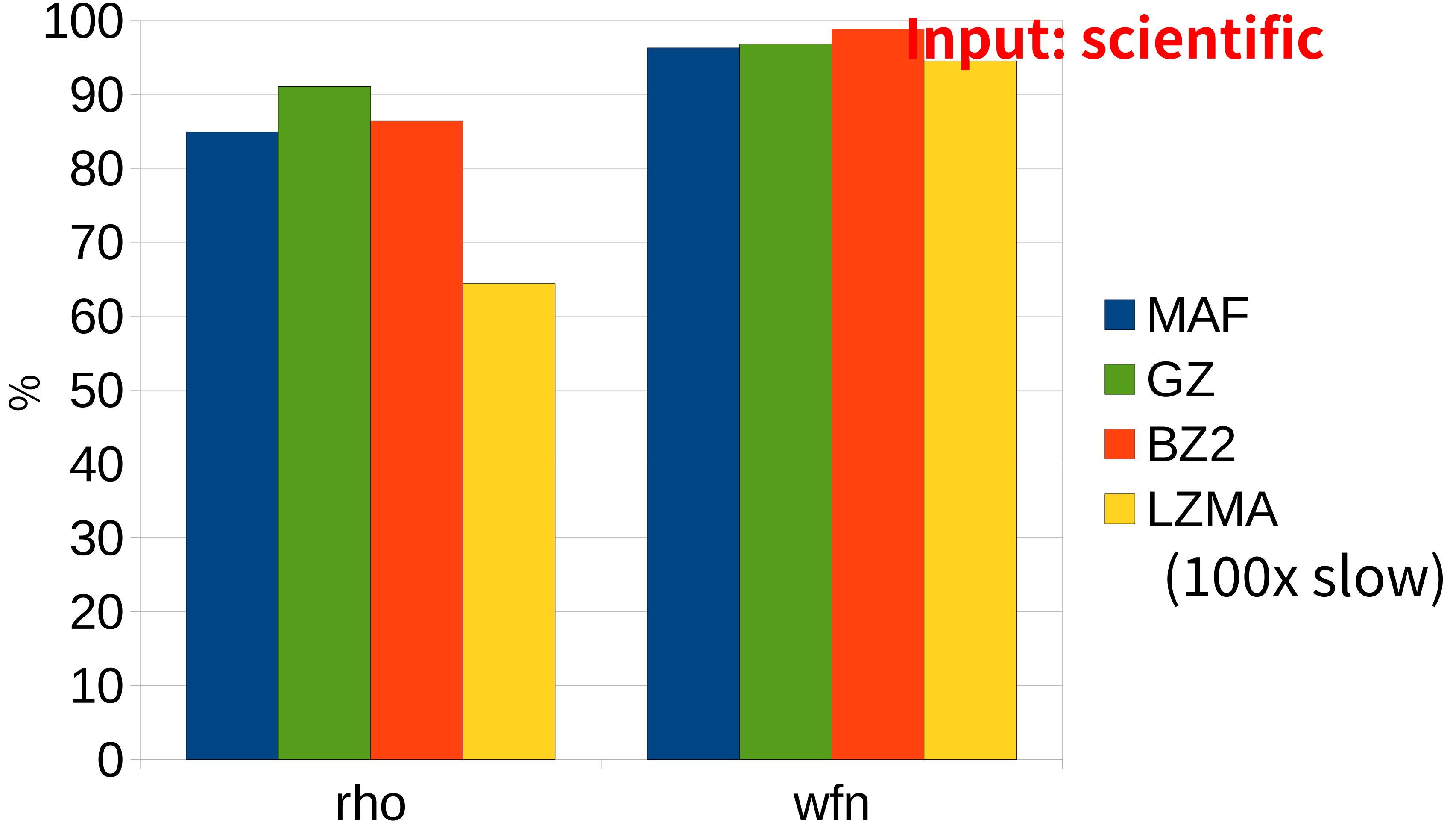
Software Compression speed

- **100x faster** than LZMA



Compression Ratio

- Better than GZ/BZ2



Evaluation: Summary

- Accelerates both accelerator and cluster
- Faster compression time compared to conventional algorithms
- Compression ratio better than GZ/BZ2

Related Work

- Compression of FP numbers:
  - For graphics [Jacob+, 08]
  - Single-precision [Lindstrom+, 06]
  - For disk and network [Burtscher+, 07]
- Ours is much faster, while retaining similar compression ratio and complete precision

Conclusion

- Floating point compression is effective
  - FFT, Matrix multiplication
- Exponent parts of FP numbers can be compressed
- Fast algorithm for FP compression is proposed
  - Both hardware and software implementations are feasible

Get your copy of MAF compression utility at:
http://www-hiraki.is.s.u-tokyo.ac.jp/members/tomari/maf/