Reducing Memory Access Latencies using Data Compression in Sparse, Iterative Linear Solvers

Neil Lindquist
All-College Thesis Defense

April 16th, 2019

Goal

- Sparse linear systems used by many scientific computations
- Problems can be large, with over a million variables
- Arithmetic is faster than fetching data from memory

Solver Description

- Preconditioned Conjugate Gradient was used for tests
  - Preconditioned with a 3 level multigrid
  - Symmetric Gauss-Seidel step smoother
- Matrix store in CSR format
  - Stores the column index and value for each nonzero entry

  3 compressible data structures
  - Vector Values
  - Matrix Indices
  - Matrix Values

Main Data Access Pattern

```
for row in rows do
    for nonzero entry in row do
        LOAD entry's value
        LOAD entry's column index
        LOAD vector value for column index
    end for
    WRITE vector value for row
end for
```

- need random vector reads
- need vector writes
- need both forward and backward iteration of matrix rows

Analytical Performance Model

- System of equations that assumes no processor-level parallelism
- Solving for what conditions outperform the baseline gives:

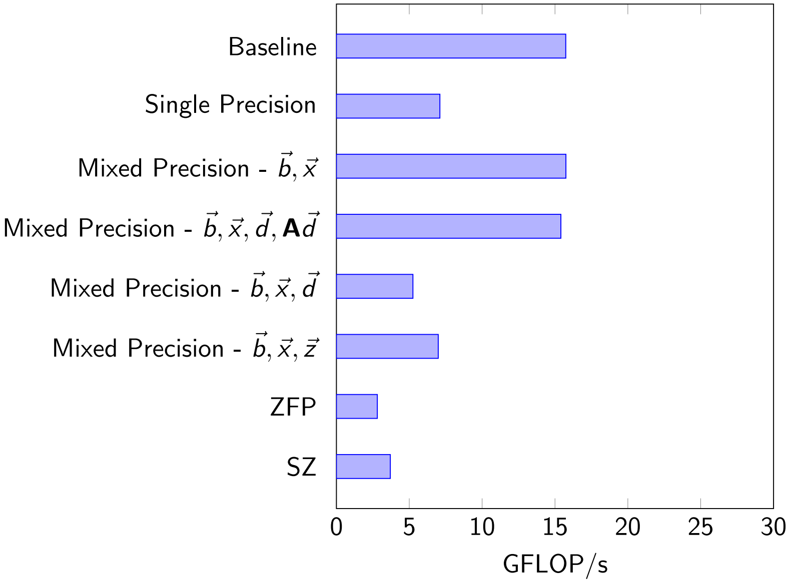
$$
\begin{aligned}
\mathrm{totalDecode} &< 32.5375 - 0.037037 \cdot \mathrm{vectEncode} \\
\mathrm{matBytes} &< 12.9664 - 0.398506 \cdot \mathrm{totalDecode} \\
&\quad - 0.0147595 \cdot \mathrm{vectEncode} \\
\mathrm{vectBytes} &< 7.9998 + 8.27835 \cdot (12 - \mathrm{matBytes}) \\
&\quad - 3.29897 \cdot \mathrm{totalDecode} - 0.122184 \cdot \mathrm{vectEncode}
\end{aligned}
$$

Simulation Performance Model

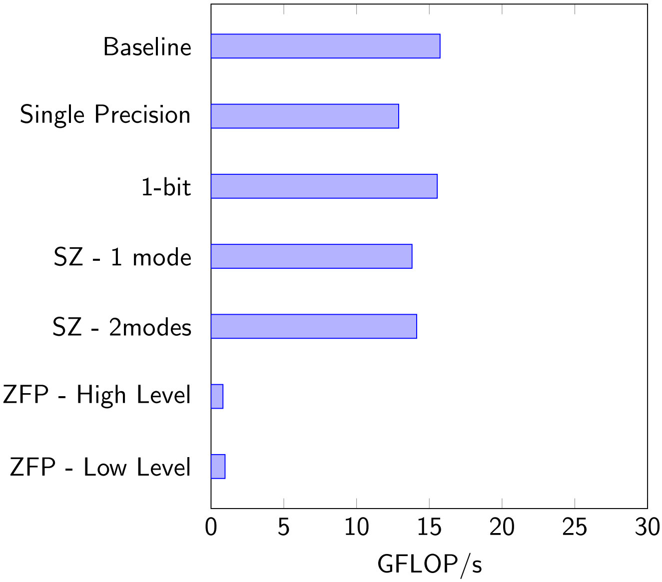
- Some processor level parallelism
- Conditions for outperforming the baseline:

Bytes	Matrix Index Decode	Matrix Value Decode	Vector Encode and Decode
1	66	41	$4.75 \geq 1.75 \cdot \text{decode} + \text{encode}$
2	51	14	$4.75 \geq 1.75 \cdot \text{decode} + \text{encode}$
3	66	40	$2 \geq 2 \cdot \text{decode} + \text{encode}$
4	0	0	$0 = \text{decode} = \text{encode}$
5	-	37	$4.75 \geq 1.75 \cdot \text{decode} + \text{encode}$
6	-	9	Not Possible
7	-	35	$0 = \text{decode} = \text{encode}$
8	-	0	$0 = \text{decode} = \text{encode}$

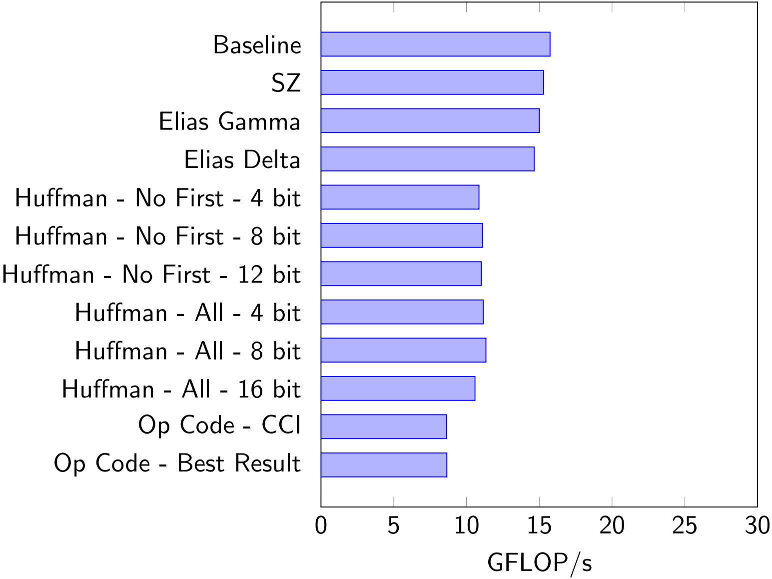
Vector Compression



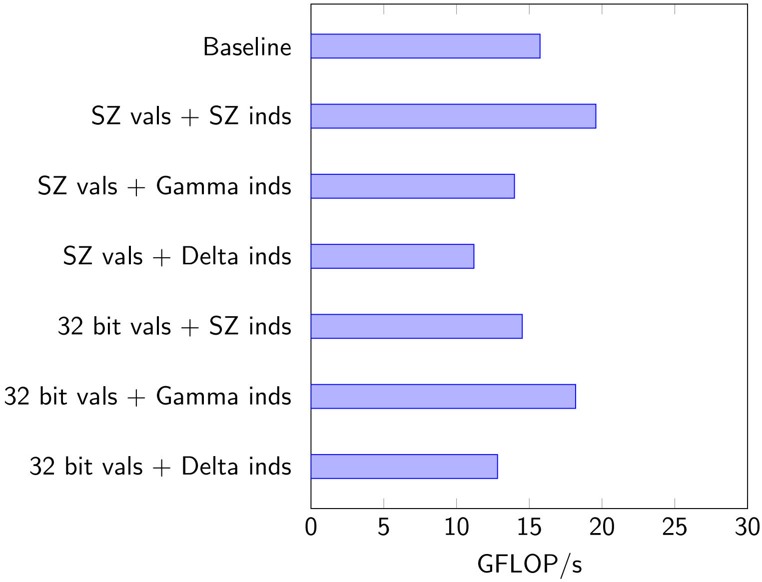
Matrix Value Compression



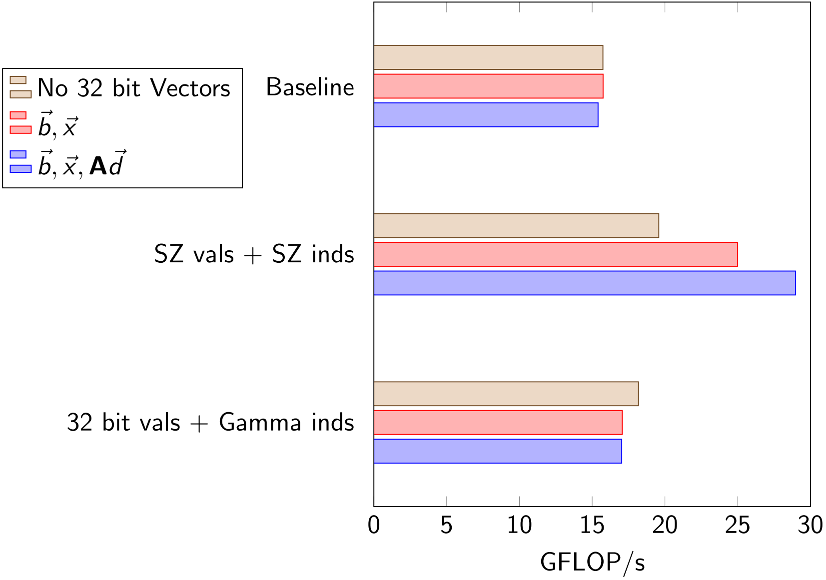
Matrix Index Compression



Matrix Value and Index Compression



Vector and Matrix Compression



Successful Compression Methods

- Mixed Floating Point Precision
- SZ Compression
- Elias Gamma Coding

Mixed Floating Point Precision

- Single Precision takes half the storage space
- But drops from 15-17 significant digits to 6-9 digits
- Certain vectors can be lower precision without slowing convergence

Squeeze "SZ" Compression

- Stores a key for the predictor function with the best accuracy
- If can't be predicted within a minimum accuracy, sorted explicitly in a second list
- Available prediction functions are chosen based on the type of data

Elias Gamma Coding

- Positive integers
- Stores number of bits needed then the data
- Very effective for small integers
  - "1" takes 1 bit
- Elias Delta Coding is similar, but Gamma codes the length

Conclusion

- Iterative linear solvers are memory access bound
- Compressing key data structures provided an 84% increase in performance