

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

All-College Thesis Defense

**Neil Lindquist**

April 16th, 2019

# Motivation

- Solving sparse linear systems used in many computations
- Billions of variables or more
- Iterative solvers are used
- Spend most of the time fetching data

# Mathematics of Conjugate Gradient

- Solving  $\mathbf{A}\vec{x} = \vec{b}$

# Mathematics of Conjugate Gradient

- Solving  $\mathbf{A}\vec{x} = \vec{b}$
- Minimizing  $f(\vec{x}) = \frac{1}{2}\vec{x}^T \mathbf{A}\vec{x} - \vec{b} \cdot \vec{x}$

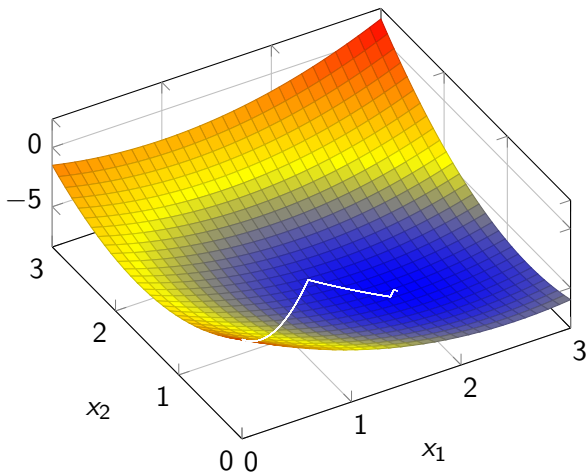
# Mathematics of Conjugate Gradient

- Solving  $\mathbf{A}\vec{x} = \vec{b}$
- Minimizing  $f(\vec{x}) = \frac{1}{2}\vec{x}^T \mathbf{A}\vec{x} - \vec{b} \cdot \vec{x}$
- If  $\mathbf{A}$  is symmetric, then  $\nabla f(\vec{x}) = \mathbf{A}\vec{x} - \vec{b}$

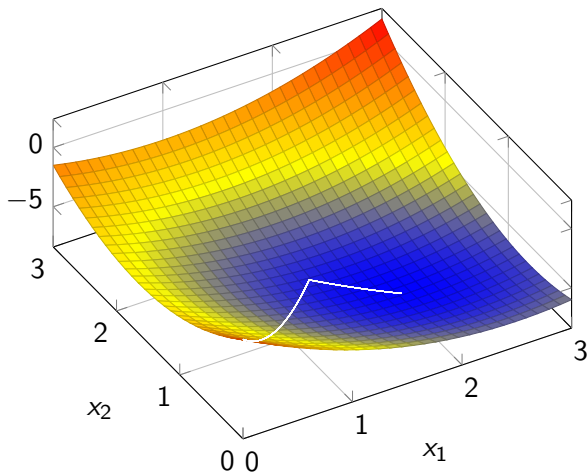
# Mathematics of Conjugate Gradient

- Solving  $\mathbf{A}\vec{x} = \vec{b}$
- Minimizing  $f(\vec{x}) = \frac{1}{2}\vec{x}^T \mathbf{A}\vec{x} - \vec{b} \cdot \vec{x}$
- If  $\mathbf{A}$  is symmetric, then  $\nabla f(\vec{x}) = \mathbf{A}\vec{x} - \vec{b}$
- $-\nabla f(\vec{x})$  is in direction of maximal decrease

# Mathematics of Conjugate Gradient



# Mathematics of Conjugate Gradient





## Solver Description

- Approximating the steady state heat equation in 3 dimensions
  - Discretized with a 27-point stencil

## Solver Description

- Approximating the steady state heat equation in 3 dimensions
  - Discretized with a 27-point stencil
- Preconditioned Conjugate Gradient was used
  - 3-level multigrid preconditioner with Symmetric Gauss-Seidel smoother

## Solver Description

- Approximating the steady state heat equation in 3 dimensions
  - Discretized with a 27-point stencil
- Preconditioned Conjugate Gradient was used
  - 3-level multigrid preconditioner with Symmetric Gauss-Seidel smoother
- Matrix store in CSR format
  - Stores the column index and value for each nonzero entry

## Solver Description

- Approximating the steady state heat equation in 3 dimensions
  - Discretized with a 27-point stencil
- Preconditioned Conjugate Gradient was used
  - 3-level multigrid preconditioner with Symmetric Gauss-Seidel 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
```

## 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

# Compression Methods

- Mixed Floating Point Precision
- SZ Compression
- Elias Gamma and Delta Codings
- ZFP Compression
- Huffman Coding
- Op Code Compression

# Compression Methods

- Mixed Floating Point Precision
- SZ Compression
- Elias Gamma and Delta Codings
- ZFP Compression
- Huffman Coding
- Op Code Compression



# Mixed Floating Point Precision

- Single Precision takes half the storage space
- Only 6-9 digits instead of 15-17 digits
- Certain vectors can be lower precision without slowing convergence
  - $\vec{b}$ ,  $\vec{x}$ ,  $\mathbf{A}\vec{d}$
  - Retains result accuracy

## Squeeze “SZ” Compression

- Predicts each value from the previous few
- Enforces minimum accuracy

## Squeeze “SZ” Compression

- Predicts each value from the previous few
- Enforces minimum accuracy
- Available prediction functions are based on the data

## Squeeze “SZ” Compression

- Predicts each value from the previous few
- Enforces minimum accuracy
- Available prediction functions are based on the data
- Compression rate is highly dependent on local patterns in the data

# Elias Gamma Coding

- Positive integers
- For each value, stores the size then the data
  - Very effective for small integers
  - Storing the difference from the previous index reduces the size of values

# Elias Gamma Coding

- Positive integers
- For each value, stores the size then the data
  - Very effective for small integers
  - Storing the difference from the previous index reduces the size of values
- Elias Delta Coding is similar, but uses Gamma coding for the length

# Elias Gamma Coding

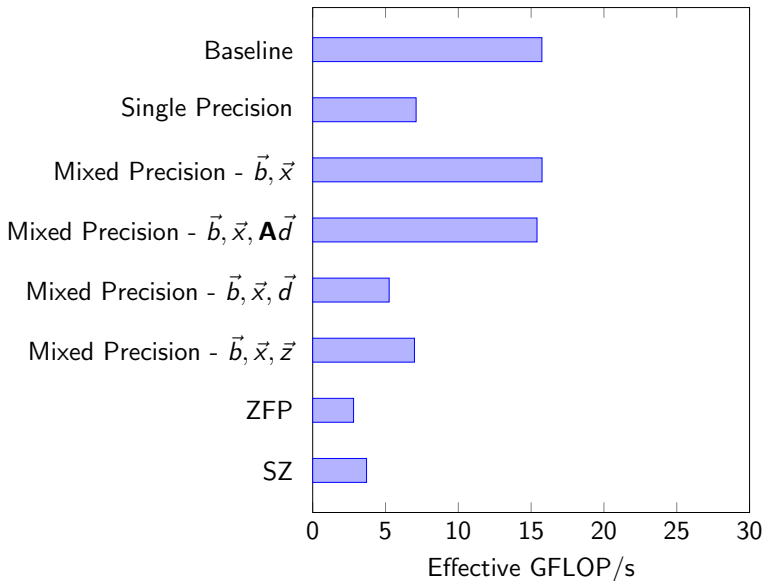
- Positive integers
- For each value, stores the size then the data
  - Very effective for small integers
  - Storing the difference from the previous index reduces the size of values
- Elias Delta Coding is similar, but uses Gamma coding for the length
- Compression rate is only dependent on the magnitude of the values

## Timing Results

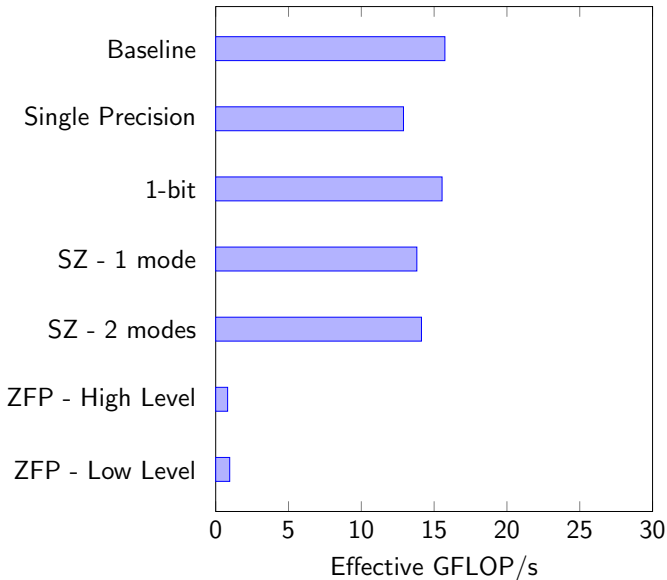
- 60 processes with  $96^3$  rows each
  - 53,084,160 total rows
- A 20-core, 2.2GHz, Intel Broadwell head node
- Plus five 8-core, 1.7GHz Intel Broadwell nodes
- MPI communication



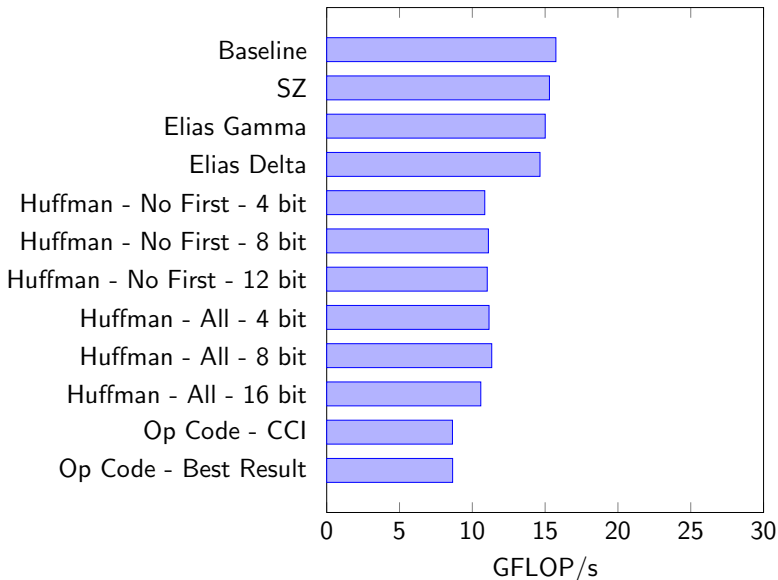
## Vector Compression



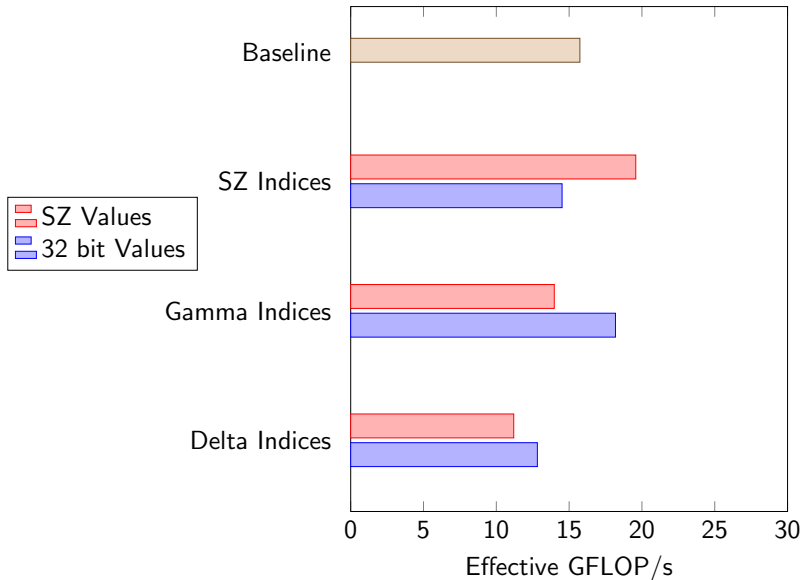
## Matrix Value Compression



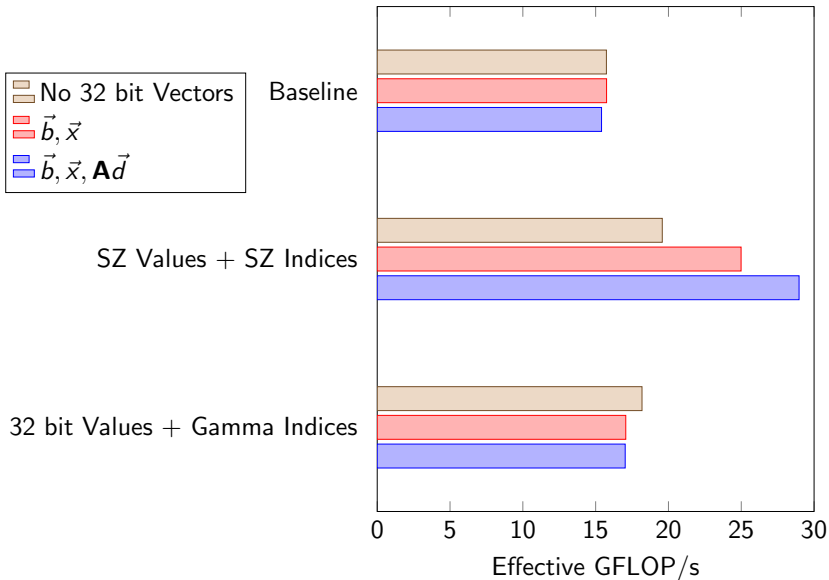
## Matrix Index Compression



## Matrix Value and Index Compression



# Vector and Matrix Compression



# Conclusion

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