S5455, GTC 2015

High Capability Multidimensional Data Compression on GPUs

Sergio E. Zarantonello

szarantonello@scu.edu

Ed Karrels

ed.karrels@gmail.com

School of Engineering, Santa Clara University

Challenge

- Massive amounts of multidimensional data being generated by scientific simulations, monitoring devices, and high-end imaging applications.
- Growing inability of current networks, conventional computer hardware and software, to transmit, store, and analyze this data.

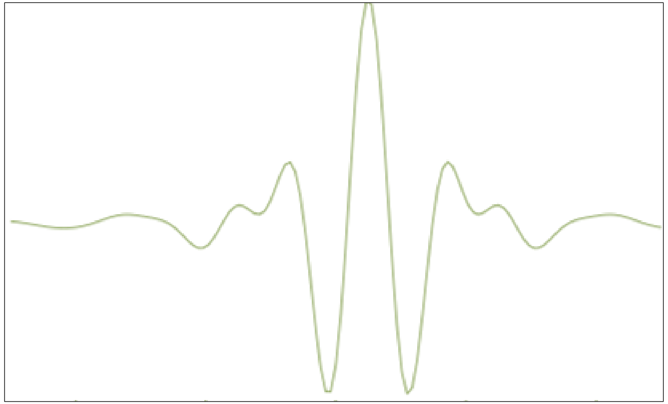
Solution

- Fast and effective lossy data compression.
- Optimized compression ratios subject to *a priori* set error bounds, requiring several iterations of compress/decompress cycle.
- GPUs to make the above feasible.

Our goal

- A multidimensional wavelet-based CODEC for large data.
- A discrete optimization procedure to provide best compression ratios subject to error tolerances and error metrics specified by user
- A high performance CUDA implementation for large data, exploiting parallelism at various levels.
- A flexible design allowing for future enhancements (redundant bases, adaptive dictionaries, compressive sensing, sparse representations, etc.)
- An initial focus on medical Computed Tomography, Seismic Imaging, and Non-Destructive Testing of Materials.

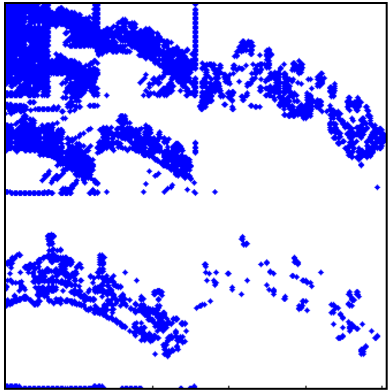
Why wavelets ?



- Wavelets are “short” waves “localized” in both, spatial and frequency domains.
- Can be used as basis functions for sparse representations of data.
- Give compact representations of *well-behaved* data and point singularities.
- Multidimensional wavelets take advantage of data correlation along all coordinate axes.
- Wavelet encoding/decoding can be implemented with *fast algorithms*.

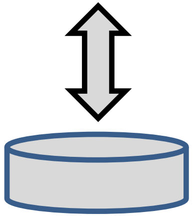
Conventional 2d procedure

Original



Non-Standard Wavelet Basis

Reconstructed



Compressed

Forward

- 2D FWT (NS)
- Thresholding
- Quantization
- Entropy Encoding

Inverse

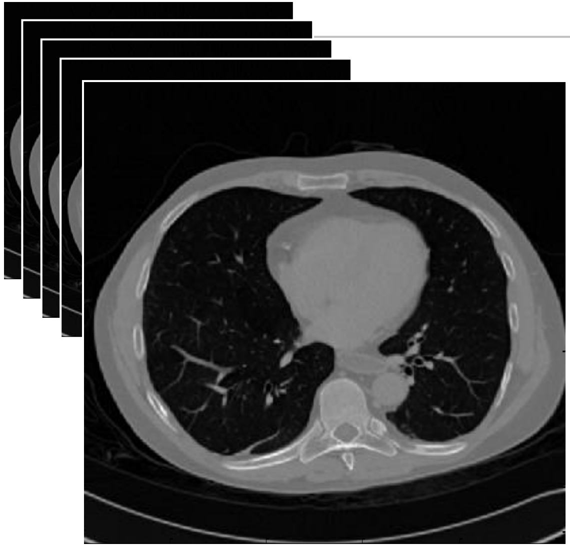
- Entropy Decoding
- Dequantization
- 2D IWT (NS)

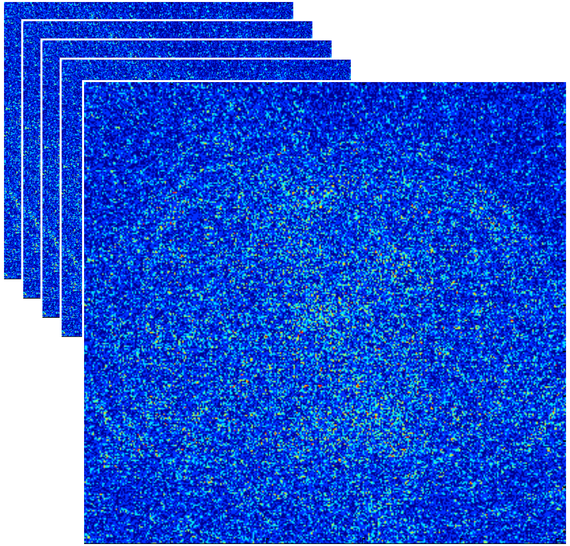
Design

- Data decomposed into overlapping cubelets.
- Cubelets encoded *via* biorthogonal wavelet filters along each coordinate axis.
- Wavelet coefficients are thresholded, then quantized.
- Quantized cubelets are Huffman encoded.
- Process is “reversed” to reconstruct the data.
- “Hill Climbing” algorithm is implemented to deliver highest compression possible subject to error constraint(s).

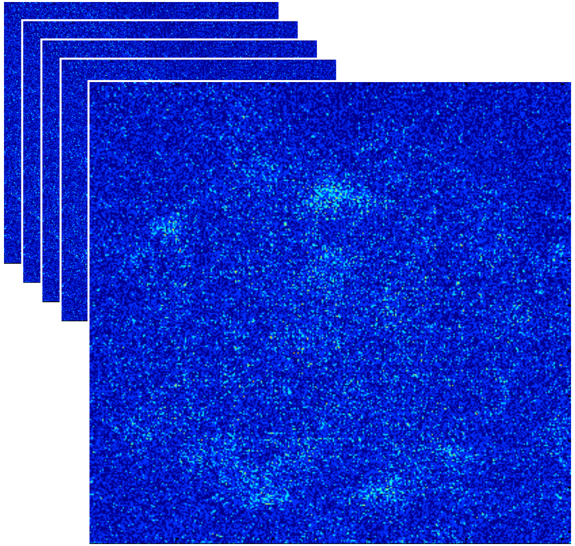
2d (frame-by-frame) versus 3d procedure

X-Ray CT scan
10 steps of the cardiac cycle
512 x 512 x 96 cube
http://www.osirix-viewer.com

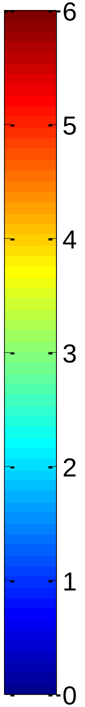




2d procedure



3d procedure



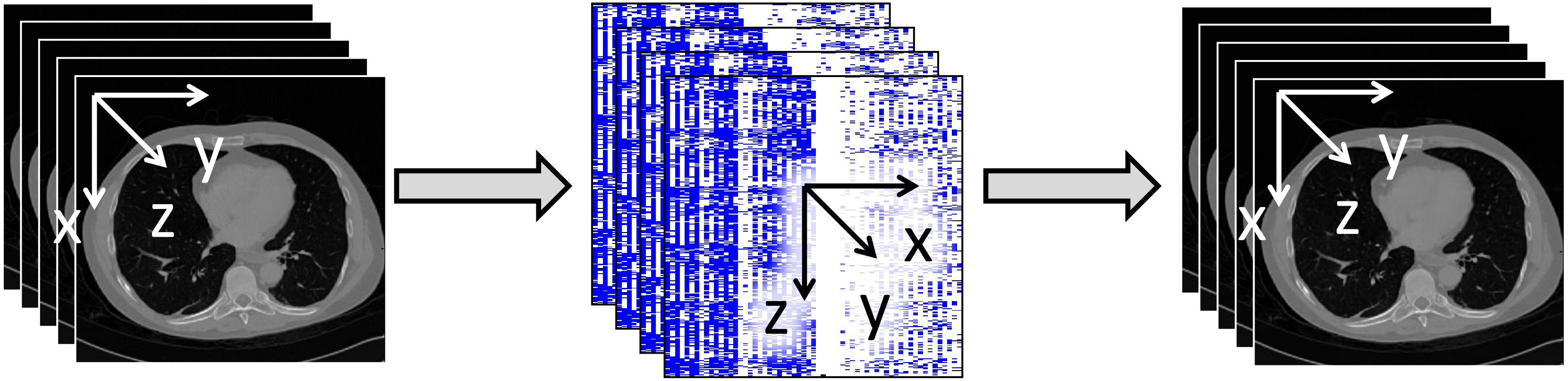
Same error rate: PSNR=46

Compression Ratio=6.6
Cutoff=88% Bins=1106
Max Error= 9.45

Compression Ratio=10.2
Cutoff=92% Bins=850
Max Error = 5.68

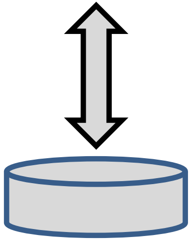
Outline of 3d procedure

Original **Reconstructed**



Standard Wavelet Basis

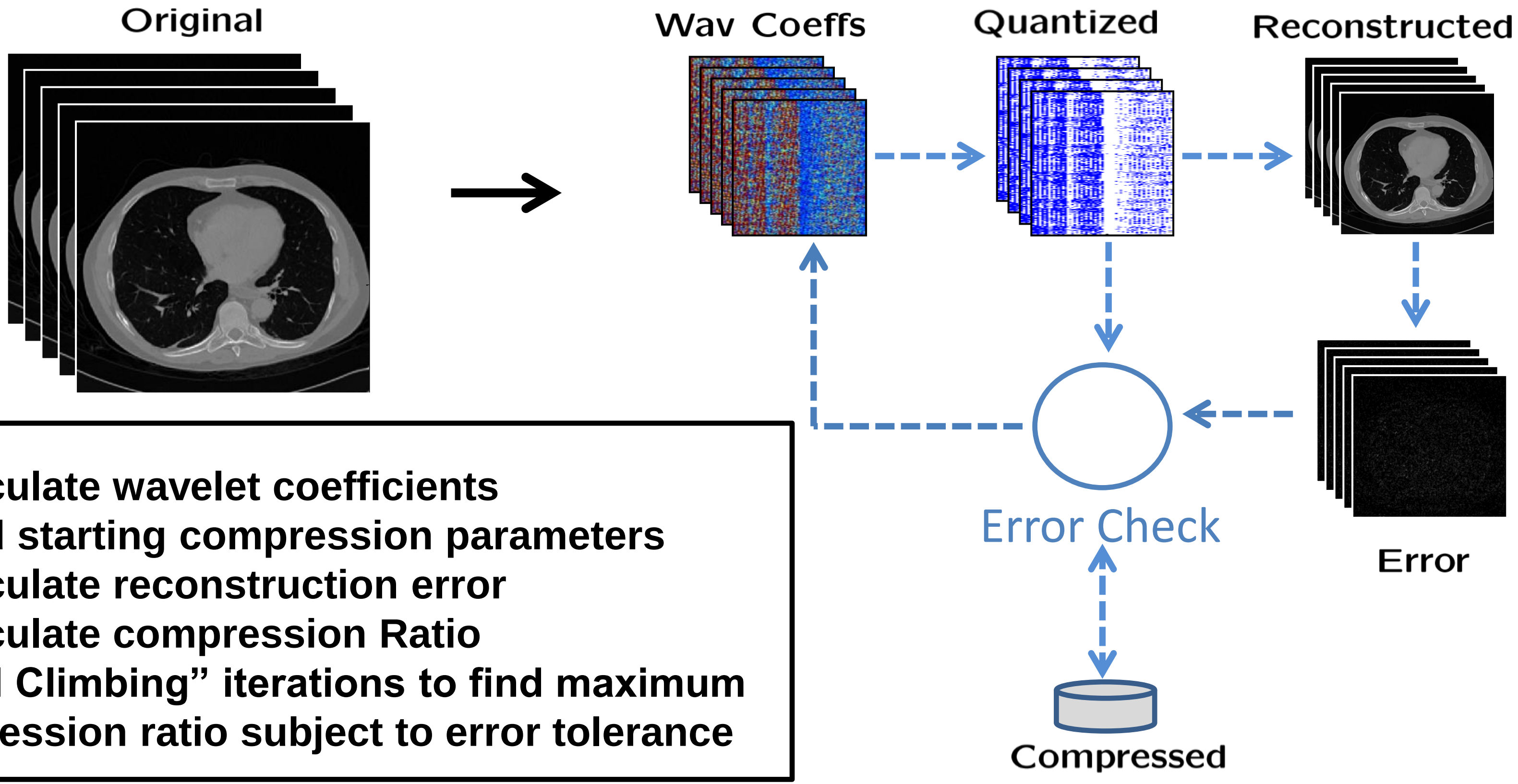
- 1D FWTs: Y × Z rows of length X
- Transpose X Y Z → Y Z X
- 1D FWTs: Z × X rows of length Y
- Transpose Y Z X → Z X Y
- 1D FWTs: X × Y rows of length Z
- Thresholding
- Quantization
- Huffman Encoding



Compressed

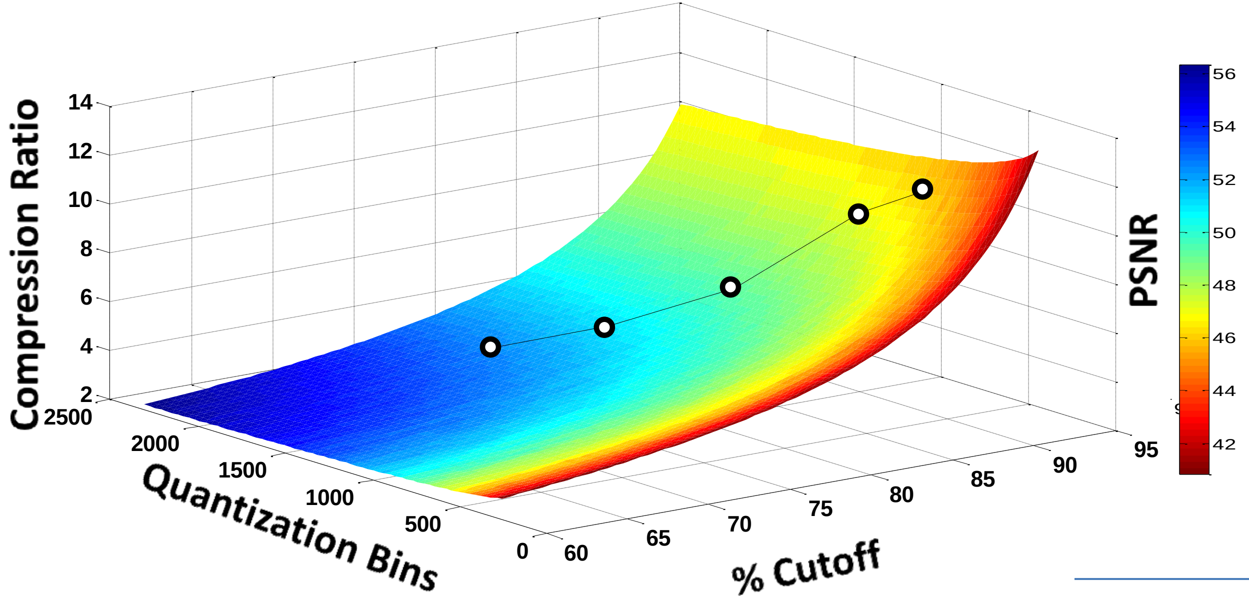
- Huffman Decoding
- Dequantization
- 1D IWTs: X × Y rows of length Z
- Transpose Z X Y → Y Z X
- 1D IWTs: Z × X rows of length Y
- Transpose Y Z X → X Y Z
- 1D IWTs: Y × Z rows of length X

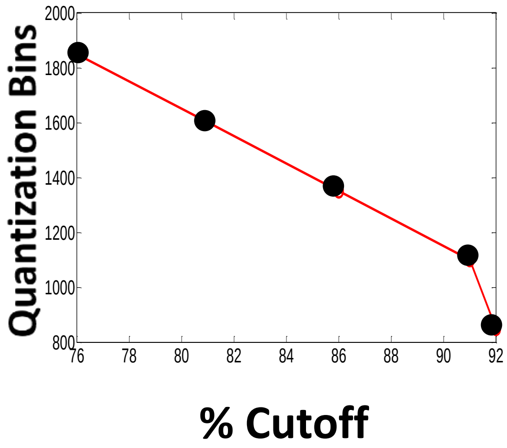
Optimized compression for given error tolerance



1. Calculate wavelet coefficients
2. Find starting compression parameters
3. Calculate reconstruction error
4. Calculate compression Ratio
5. “Hill Climbing” iterations to find maximum compression ratio subject to error tolerance

Optimized compression for given error tolerance





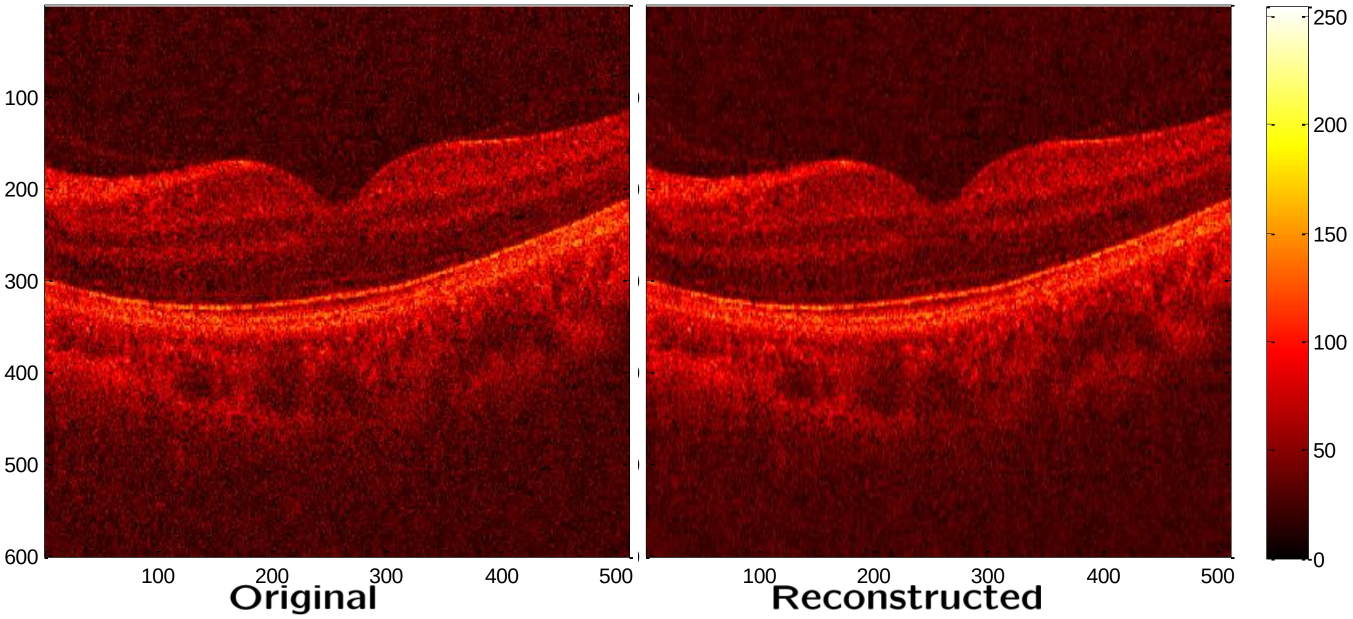
Constraint: PSNR ≥ 46

	Begin	End
Cutoff	76 %	92 %
Bins	1850	850
PSNR	52.4	46.2
Ratio	3.6 X	10.2 X

Applications: Optical Coherence Tomography

Objective: efficient transfer over the internet of high-resolution 3d images of retina for diagnosis.

Dataset courtesy of Quinglin Wang
Carl Zeiss Meditec Inc.



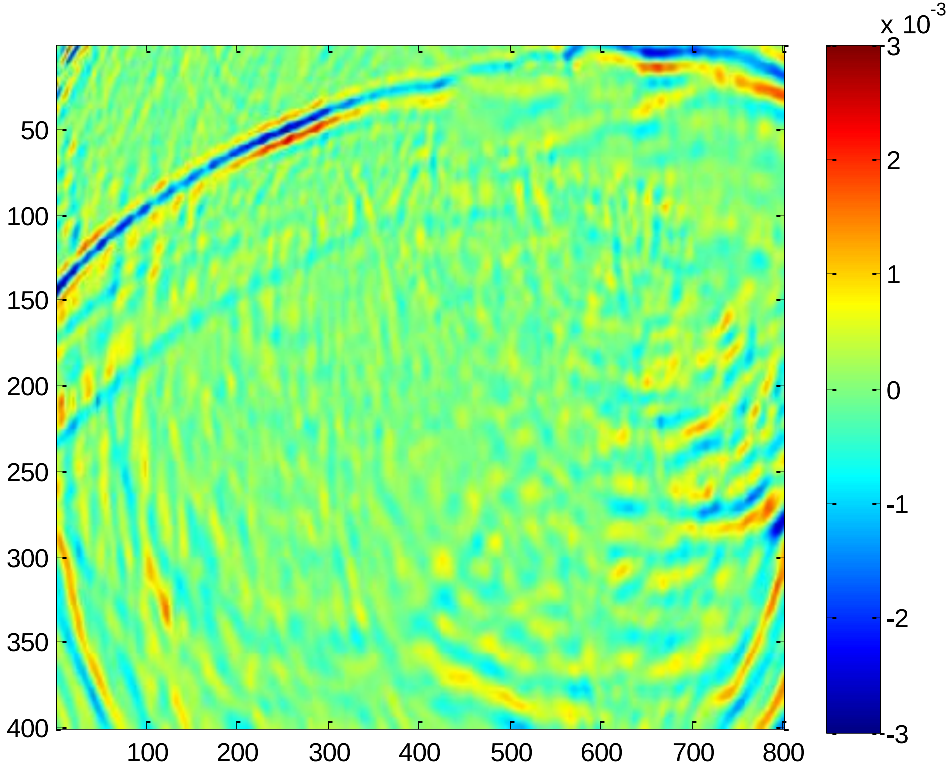
$960 \times 512 \times 128$ cube

$$\text{PSNR} = 27$$
$$\text{Comp.Ratio} = 21\text{X}$$
$$L^{\infty}\text{-Rerr} = 0.34$$
$$\text{Quality} = 0.66$$

$$\text{Rerr} = \frac{\|\text{Orig} - \text{Rec}\|_{\infty}}{\|\text{Orig}\|_{\infty}}$$

$$\text{Quality} = (1 - \text{Rerr})$$

Applications: Reverse Time Migration

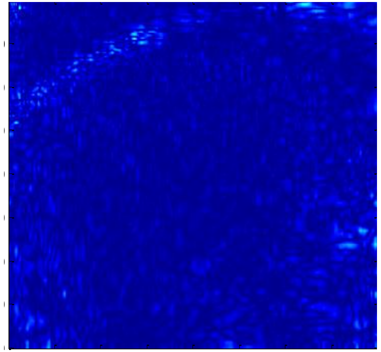


$400 \times 800 \times 400$ wavefield

- Step1. Time-slice records of an evolving 3D source wavefield are stored on disk.
- Step 2. Records are read back to memory in reverse order and cross correlated with a receiver wavefield to incrementally build a seismic image.
- Hundreds of terabytes of data can be involved.
- IO time for writing and reading this data is significant.

PSNR $= 59$
Comp.Ratio $= 22$X
L^{∞}-Rerr $= 0.02$

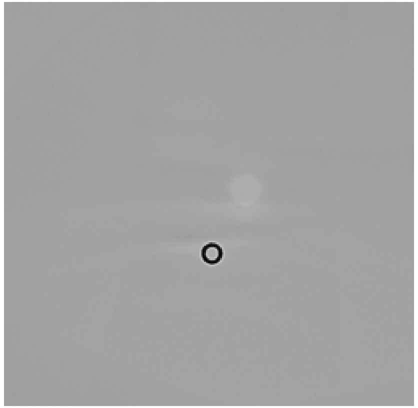
Error $\rightarrow$



Part 2: Implementation

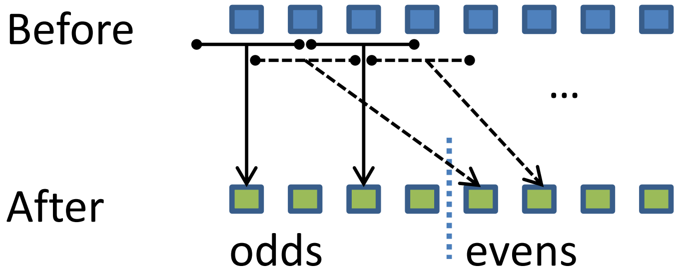
- Stages of compression
  - Wavelet transform
  - Threshold
  - Quantization
  - Huffman coding
- Overall speedup

Quality control X-ray scan from NSI



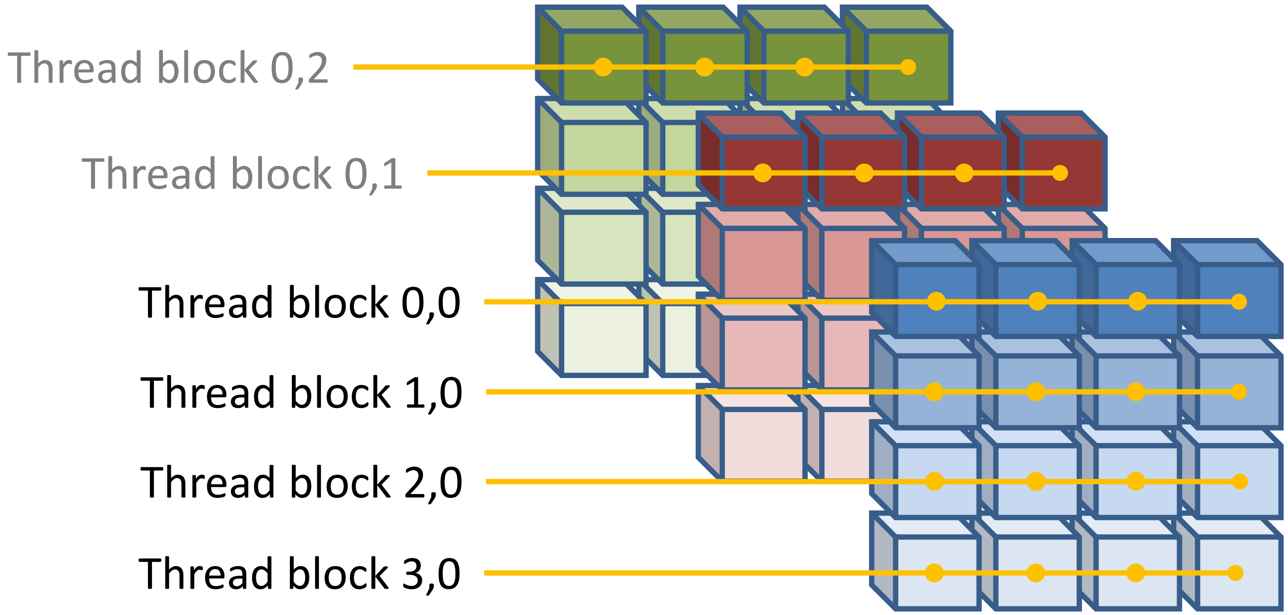
Wavelet Transform on GPU

- Apply convolution
- Each row is independent
- Within each row, multiple read / write passes



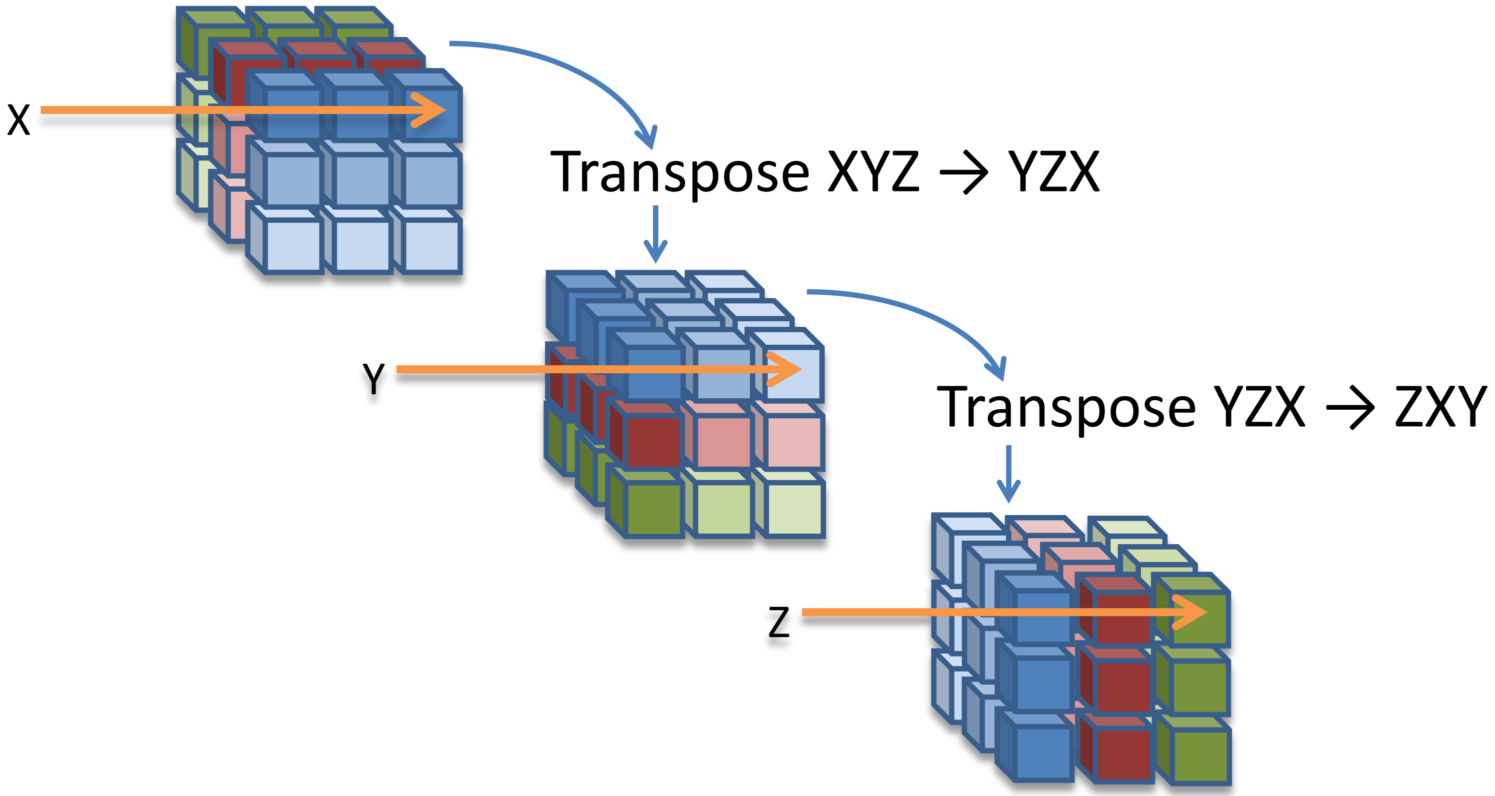
- 1 row == 1 thread block
- Synchronize between read & write

3d Wavelet Transform



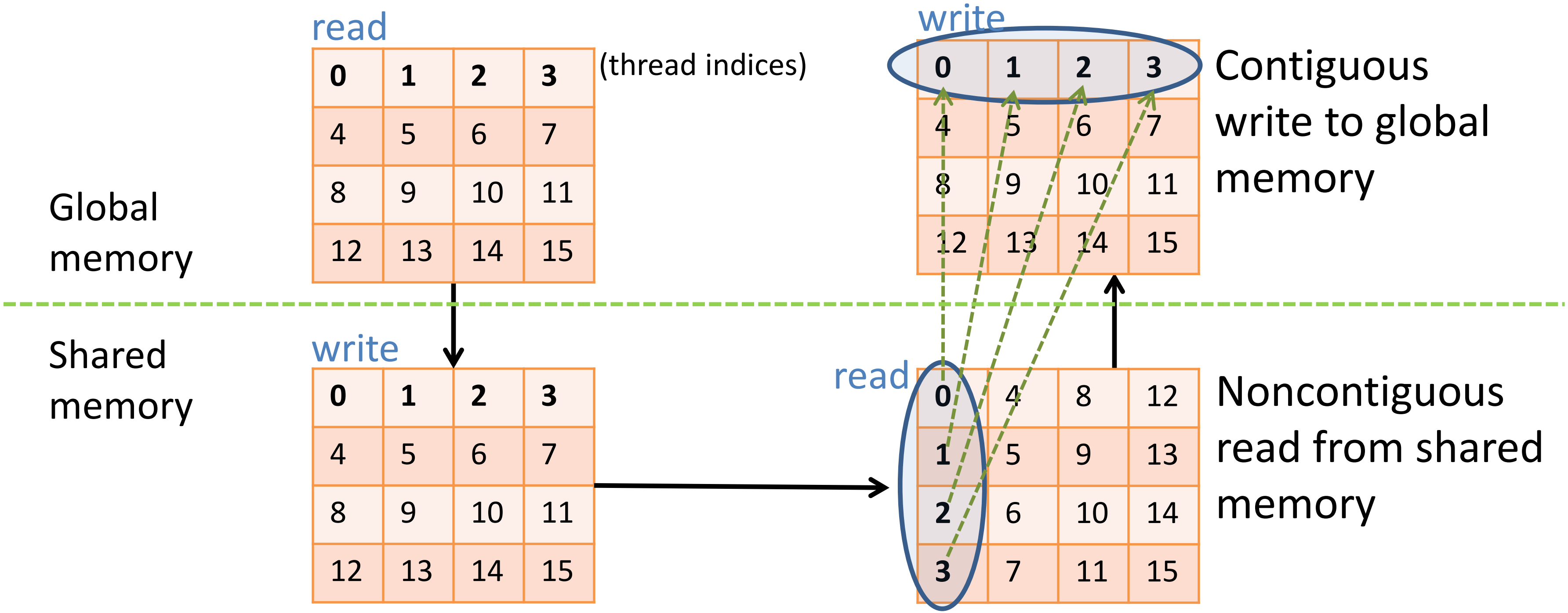
Height × depth rows, each one is an independent thread block.

Transform Along Each Axis



GPU Transpose

- Access global memory in contiguous order



Optimizations

Version 1: Global memory

Version 2: Shared memory

- 2.5× speedup

Version 3: Constant factors double → float

- 1.6× speedup

```
#define FILTER_0  .852
#define FILTER_1  .377
#define FILTER_2 -.110
```

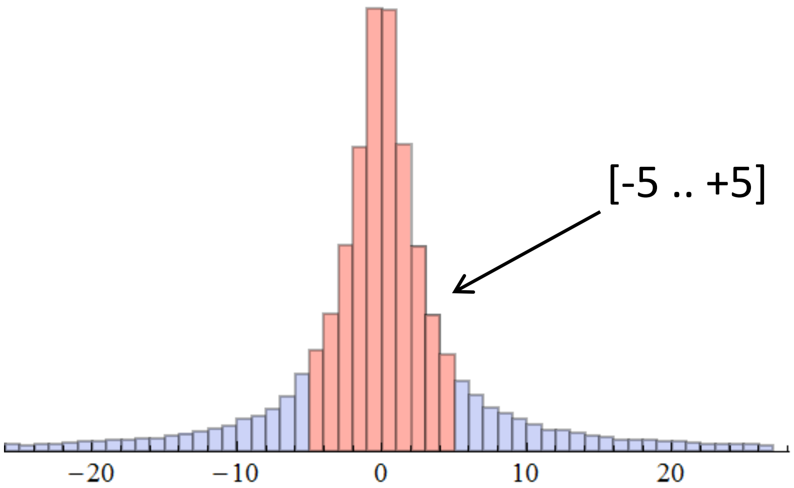
```
#define FILTER_0  .852f
#define FILTER_1  .377f
#define FILTER_2 -.110f
```

Speedup over CPU version: 105x

(860ms → 8.2ms for 256x256x256 cubelet,
8 levels of transforms along each axis)

Threshold

- Trim smallest x% of values – round to 0



- Just sort absolute values using Thrust library
- Speedup over CPU sort: 112× (7.0 toolkit is 35% faster than 6.5)

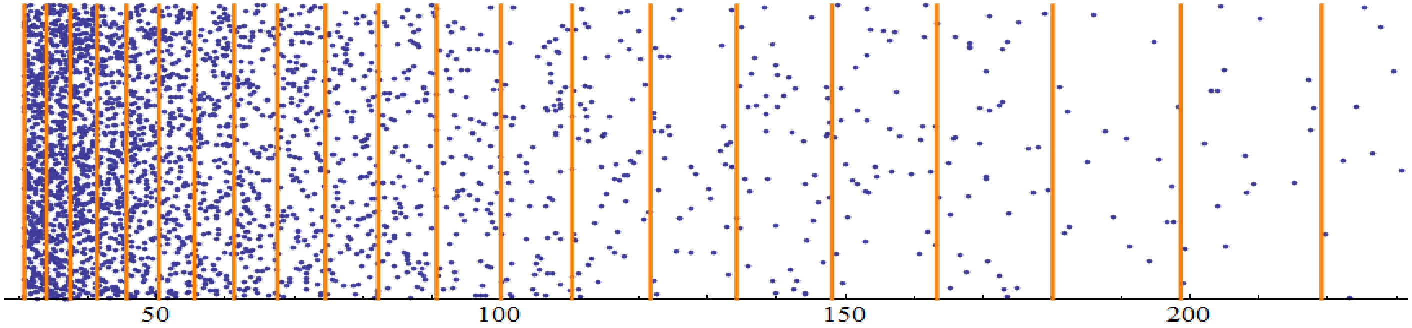
Quantization

Map floating point values to small set of integers

- Log : bin size near x proportional to x
  - Matches data distribution well
  - Simple function; fast

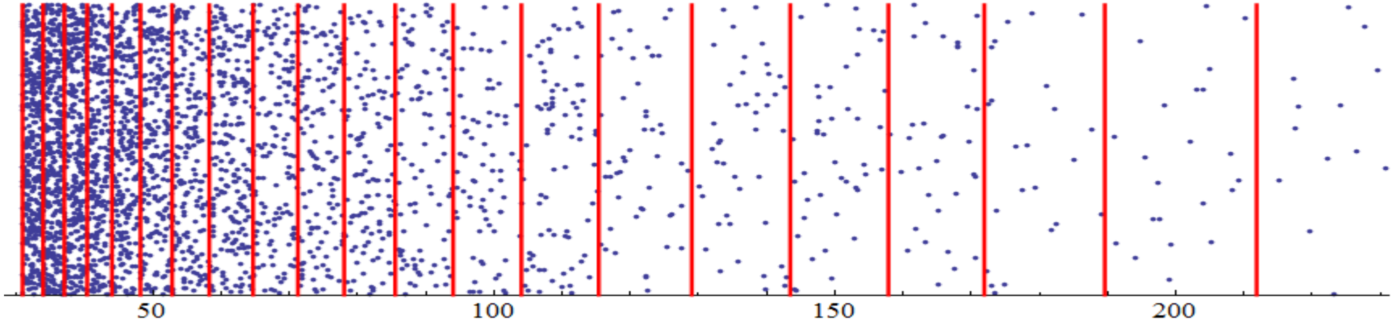
- Lloyd's algorithm
  - Given starting bins, fine-tune to minimize overall error
  - Start with log quantization bins
  - Multiple passes over full data set, time-consuming

Log / Lloyd Quantization



Log quantization, pSNR 38.269

GPU speedup: 97× (thrust::transform())



Lloyd quantization, pSNR 45.974

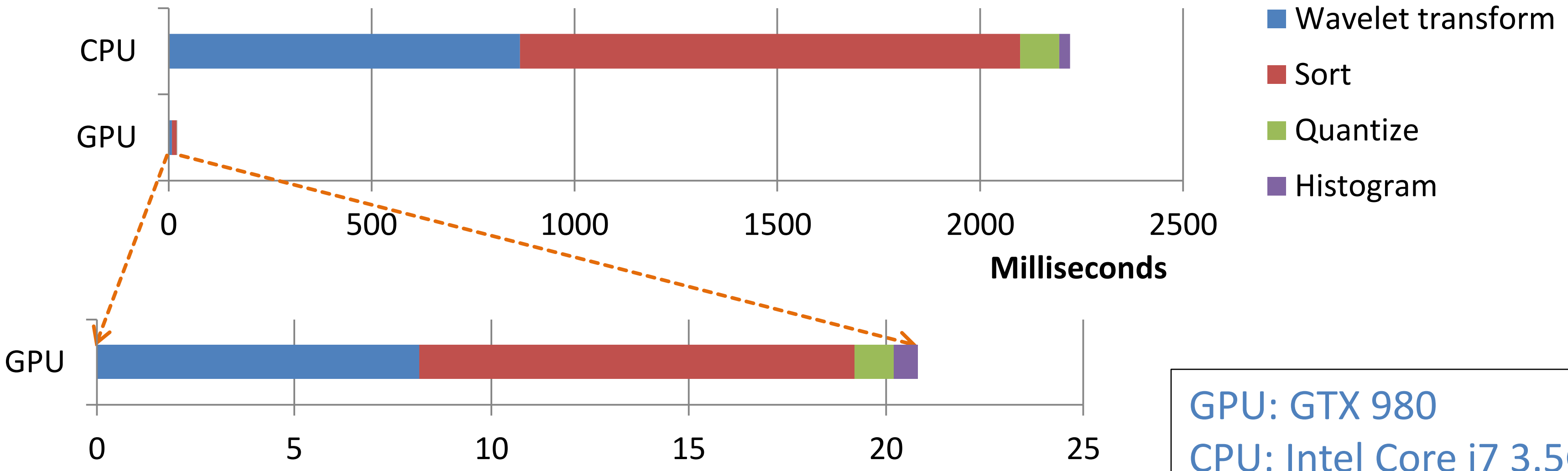
GPU speedup: create 13×, apply 48×

Huffman Encoding

- Optimal bit encoding based on value frequencies
- Compute histogram on CPU
  - Copy data GPU → CPU: 17ms
  - Compute on CPU: 27ms
- Compute histogram on GPU
  - No copy needed
  - Compute: .61ms
  - Optimization: per-thread counter for common value

Value	Count	Encoding
9	16609445	1
8	46198	011
10	42896	001
11	32594	000
7	30831	0101
12	6942	01000
6	5388	010011

Overall CPU → GPU speedup



GPU: GTX 980
CPU: Intel Core i7 3.5GHz

	MATLAB	CPU	GPU
Compress	43000	2300	21
Error control	39000	1400	18

time to process 256^3 cubelet, in milliseconds

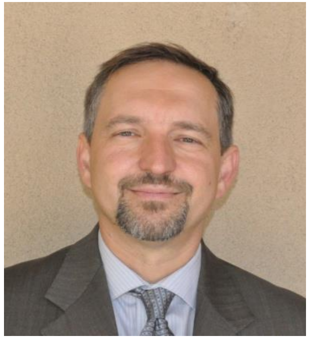
Future Directions

- Improve performance
  - Use subsample for training Lloyd's
  - Use Quickselect to find threshold value
  - Multiple GPUs
- Improve accuracy
  - Weighted values in Lloyd's algorithm
  - Normalize values in each quadrant

Our Team

Sergio Zarantonello
Santa Clara University
szarantonello@scu.edu

Ed Karrels
Santa Clara University,
ed.karrels@gmail.com



Drazen Fabris
Santa Clara University
dfabris@scu.edu

David Concha
Universidad Rey Juan Carlos, Spain
david.concha@urjc.es

Anupam Goyal
Algorithmica LLC
anupam@rithmica.com

Bonnie Smithson
Santa Clara University
Bonnie@DejaThoris.com