Hardware Decompression for Compressed Sensing Applications

Keith Dronson   Frank Zovko   Samuel Subbarao
Federico Garcia

Columbia University

May 14, 2009

Outline

Motivation

- Compressed sensing is a relatively new approach to collecting and storing images
- Trade-off between image storage space and decompression time
- Decompression can take a very long time
- Increase in speed of decompression by using dedicated hardware

Compressive Sensing

- Conventional Sampling: Shannon's Sampling Theorem / Nyquist rate
- Images are not bandlimited
- Desired resolution determines bandwidth
- Compressive sensing - fewer samples can represent almost the same image
- Compressive sensing - Dependant on:
  - Sparsity
  - Incoherence

Sparsity and Incoherence

- Sparsity
  - Bandwidth may be larger than actual number of “information” samples
  - Signal represented in the right basis, Ψ , would be more compressed
- Incoherence
  - Something compressed in Ψ will be spread out in the original basis

Mathematical Background

- Typical approach to sensing: $y_k = \langle f, \phi_k \rangle$
  - f is image to be sampled
  - ϕ_k is sensing waveform
  - y_k is sampled data
- Assuming: ϕ_k's are indicator functions of pixels, then y_k's are typical image data
- Dimension of y is n... Perhaps we could take less than that (say m) and still get a good image.
- Create an $m \times n$ sensing matrix, A, composed of n rows of the ϕ_k's: $\phi_1^*, \phi_2^*, \ldots, \phi_m^*$
  * denotes complex transpose
- Problem: f is n dimensional, y is of dimension m and $y = Af$
  Infinite number of possibilities for f!

Sparsity

If $f \in \mathbf{R}^n$ and sampled in an n dimensional basis $(\phi_1, \phi_2, \dots, \phi_n)$, then we have:

$$f = \sum_{1}^{n} x_i \phi_i \tag{1}$$

Some x_i's are small, toss out the related ϕ_i's and you could still almost add up to f: $f = \sum_1^s x_i \phi_i$, or

$$f = \Phi x_s \tag{2}$$

Φ is $n \times n$ matrix of $\phi_1 - \phi_n$ as columns. x_s are the s largest coefficients of the x_i's.

Sparsity

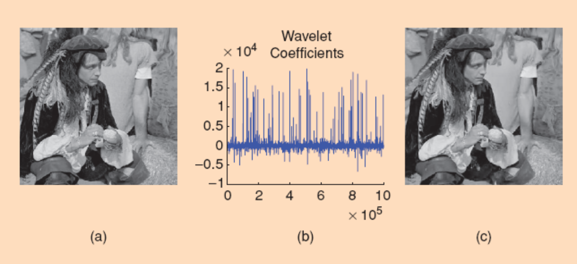


Figure: Part a shows the initial image. Part b is the image in the ϕ basis. Note that there are only a few discrete ϕ_i's that have x_i's with large coefficients. Part c is the reconstruction of the image using the phi_i's linked to the largest 25,000 coefficients. This means that 97.5% of the sampled data was thrown away and the picture still looks pretty good.

Incoherence

- Since $f \in \mathbf{R}^n$, we can find two basis sets Φ (represents f) and Ψ (used as the sensing basis) for the space.
- Compressive sensing looks for low coherence pairs (maximum incoherence) between any two elements of Φ and Ψ.
- Coherence measures the largest correlation.
- Φ will be some fixed basis. The best basis for Ψ is a random basis (white Gaussian noise).

Recovery

$y = \Psi f$ or $y_k = \langle f, \psi_k \rangle$ (dot product of f with each basis vector in Ψ). In order to recover the image we look at the following:

$$y_k = \langle \phi_k, \Psi f \rangle \tag{3}$$

f is the signal to be recovered... of course this is impossible, given the number of unknowns and equations. But f is sparse, so we can try to solve:

$$\min \left(\|x\|_0, \Psi x = y \right) \tag{4}$$

Essentially looking for an x with the least number of non-zero coefficients that will satisfy $\Psi x = y$. But this is also intractable, so we'll solve a similar problem (L1 minimization):

$$\min \left(\|x\|_1, \Psi x = y \right) \tag{5}$$

Finally $f_{\mathrm{rec}} = \Phi x$.

Overall Architecture

- Compression done on a computer using Matlab.
- Image (just black and white, can be extended to color) is first made sparse using the Daubechie Wavelet Transform (described in the next section).
- N largest elements are preserved while the rest is set to zero.
- Sparse image is then multiplied by the random matrix A, resulting in a smaller data set.
- This smaller data set is sent to the FPGA to be decompressed

Architecture

- Implements the decompression side of a CS system on the Altera Cyclone II FPGA board.
- CPU runs C program that decompresses a compressed image stored in the SDRAM.
- Computationally intensive operations are built in hardware to increase the speed
- Decompressed image is then displayed on the VGA display (in addition to a few others to show how the process works)
- We use a 128x128 pixel image.

Overall Hardware Design

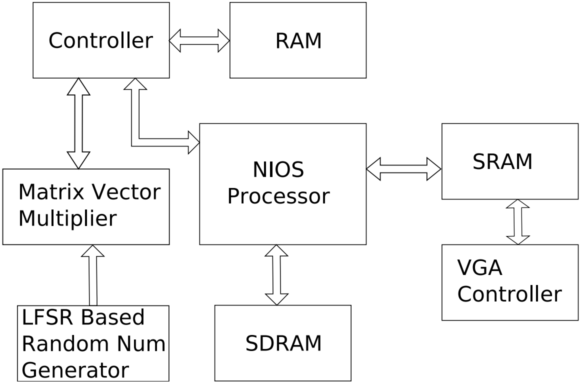


Figure: Overall Design Architecture

Accumulator Design

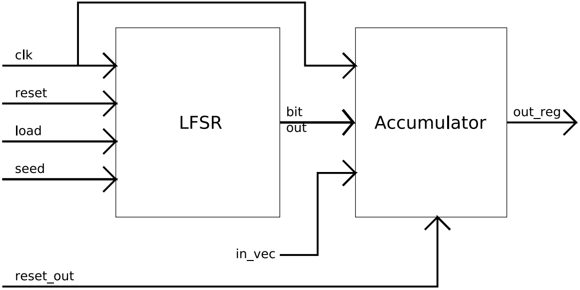
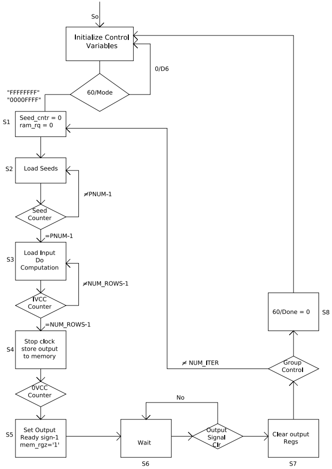


Figure: Accumulator Design

State Diagram



Software Architecture

- C code mimics the matlab code until it is time to do one of the 3 mat-vec mults
- For matrix-vector multiplication the CPU loads data into the memory of our hardware unit which then performs the computation.

Uncompressed Image

Figure: Uncompressed Image

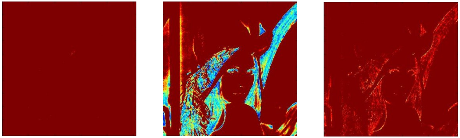


Figure: Uncompressed Image Parts: From left to right - Red, Green and Blue

Transformed Image

- Each single row of the image is transformed using the Daubechie Wavelet Transform (to make it sparse).
- Entire matrix transposed, and then each row is transformed again (getting both rows and columns of the origional).
- This process is repeated on each subset of the image until the image left to be transformed is of size 2×2.

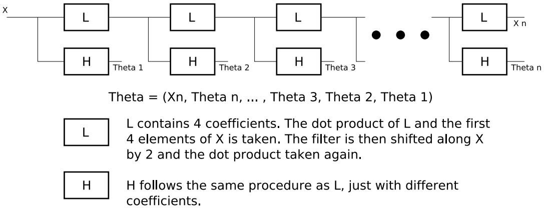


Figure: Daubechie Filter Bank

Transformed Image

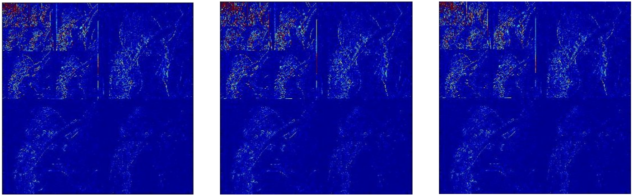


Figure: Transformed Image: Red, Green, Blue

Inverse Transform

- The inverse transform follows the same procedure as the transform except in reverse.
- The coefficients of the inverse transform are obviously different.
- Just like the transform, the inverse must be done to both rows and columns.
- This is done in C, on the processor.

Figure: Inverse Transformed Image

Summary

- Implemented the compression and transform in Matlab.
- Implemented the decompression algorithm and inverse transform in C, with hardware to assist in the computation
- Couldn't get the hardware block to work
- Couldn't use the output of the C decompression algorithm. Implemented the decompression algorithm in Matlab and used that to create the displayed output.
- Decompression algorithm running purely in C w/o hardware support works fine for small images

Lessons Learned

- Primary problems were with understanding the algorithms behind compressive imaging.
- Couldn't run the decomp code on the processor w/o hardware support for any decent-sized images due to inability to store the A matrix
- Needed to rely on the hardware fully working to get a meaningful result (but it failed)
- Insufficient information about what was going on in the hardware, which made it a lot harder to debug