

Compressive Sampling Hardware Reconstruction

Avi Septimus and Raphael Steinberg

VLSI Laboratory - Department of Electrical Engineering
Technion - Israel Institute of Technology, Haifa, Israel 32000
{septman,rafi}@siglab.technion.ac.il

Abstract—Compressive Sampling reconstruction techniques require computationally intensive algorithms, often using L^1 optimization to reconstruct a signal that was originally sampled at a sub-Nyquist rate. In this work we present a VLSI implementation of a computationally efficient algorithm named Orthogonal Matching Pursuit. We further optimize the algorithm to meet typical hardware constraints and describe the different block units of our design. We synthesize our design for the Xilinx Virtex 5 FPGA and give timing and area results. We summarize our work with a short discussion of the possible uses for our system.

I. INTRODUCTION

Signal processing applications often sample much more data than is actually required by the end user. For example, a digital color camera samples a raw color image only to compress it at a later stage by (roughly) a 100:1 ratio. Such high compression ratios are possible because the original image contained sparse information, i.e. only a small number of its values were non-zero in some transform domain. Compressive Sampling (CS) allows for the recovery of sparse signals sampled at a sub-Nyquist rate directly in the sparse domain or after undergoing an invertible linear transformation. The field has gained much interest since work done by Candès on Magnetic Resonance Imaging in 2004. A good review of CS by Candès can be found in [1].

Sampling is often done in continuous time/space by multiplying the signal by a pseudo-random rectangular matrix Φ . Usually, the columns of Φ are chosen from a random Gaussian or Bernoulli distribution or specified deterministically according to some application requirements. For instance, a non-sparse signal can often be brought to a domain in which it is sparse by having it undergo a linear transformation. In this case, Φ could be the product of a Gaussian induced matrix and the linear transform correcting matrix. Using the Bernoulli distribution to determine Φ , we obtain a binary sampling matrix that can be used for a very fast, memory efficient, hardware implementation. The more general case, as demonstrated in this work, requires using rational numbers for Φ . In any case, the CS reconstruction requires Φ to be determined and given in advance.

In the following equation we see that multiplying the input signal $x \in \mathbb{R}^N$ by the rectangular matrix $\Phi \in \mathbb{R}^{K \times N}$, $K \ll N$, yields a measurement vector $y \in \mathbb{R}^K$ that is significantly smaller than x :

$$y = \Phi x \quad (1)$$

The advantages of CS are compact signal representation and

the ability to use lower rate A/D. These advantages translate to lower power and memory requirements from the A/D sampling side. However, these advantages are incurred at the cost of a computationally intensive reconstruction algorithm. Optimal reconstruction is NP-hard [2] and involves searching over all of the possible support sets for x . Therefore, most attention has been focused on finding approximate solutions to the problem.

A common approach to sparse reconstruction is known as Basis Pursuit (BP) [3]. This method uses convex optimization to find a signal representation in an over-complete dictionary that minimizes the L^1 norm of the coefficients in the representation. While known to achieve accurate signal reconstruction, BP is more computationally intensive and has been shown to be significantly slower than other methods [4].

Orthogonal Matching Pursuit (OMP) was recently shown by Tropp and Gilbert [5] to be an efficient and reliable reconstruction algorithm. OMP is a greedy method which identifies the location of one nonzero component of x at a time. In order to converge, it requires a minimum number of samples in the order of $O(m \cdot \log N)$, where m is the signal's sparsity and N is the original dimension of the problem. Tropp and Gilbert show that by performing enough measurements, signal recovery is possible with high probability. We chose to implement OMP due to its speed and relative simplicity.

II. PRIOR WORK

We could not find any ASIC or FPGA implementation of a reconstruction algorithm for CS. We thus believe that our system is the first VLSI implementation of a CS reconstruction algorithm. We did find several implementations that accelerate the reconstruction using parallel computing and GPGPU [6]–[8]. Borghi *et al.* [6] compare the performance of a reconstruction algorithm on an IBM Cell processor, an Nvidia GPU and an Intel Dual Core. In a different work, Andrecut [7] uses a general matching pursuit algorithm which is less computationally demanding than OMP. GPU algorithms achieve decompression results in the order of tens of milliseconds for vectors of 512 elements. Their major bottleneck is memory bandwidth from the main memory to the GPU and back.

III. IMPLEMENTATION

A. Overview

In this section, we review the OMP algorithm for signal recovery as described in [5]. The reconstruction system requires two inputs, the sampling matrix Φ and the measurement vector y . The output of the system \hat{x} is an approximation of the



Fig. 1. OMP Reconstruction

original signal x . Fig. 1 shows the basic OMP reconstruction block.

As stated earlier, if x is an m -sparse signal, then we need to find the m columns of Φ which contribute to y . In each iteration, we choose the column of Φ which is best correlated with the remaining part of y . We then determine its contribution, subtract it from y , and perform the next iteration on the residual vector. After finding the relevant columns of Φ , the values of the signal are found solving an over-determined least squares equation. The procedure for OMP reconstruction is as follows:

- a. Initialize the residual $R_0 = y$, the index set $S = \emptyset$, and the iteration counter $t = 1$.
- b. Find the index s_t which solves the simple optimization problem

$$s_t = \arg \max_{j=1 \dots N} |\langle R_{t-1}, \phi_j \rangle| \quad (2)$$

where ϕ_j is the j column of Φ .

- c. Update the index set

$$S_t = S_{t-1} \cup \{s_t\}. \quad (3)$$

- d. Perform modified Gram-Schmidt [5] using the s_t column of Φ and $q_{1 \dots t}$ in order to determine q_{t+1} .
- e. Calculate the new residual according to

$$R_t = R_{t-1} - q_{t+1} \cdot q_{t+1}' \cdot R_{t-1}. \quad (4)$$

- f. Increment t and return to step b if t is less than the assumed sparsity m of the signal.
- g. Find the signal values at the indices in S by solving the least squares problem

$$\hat{x} = \arg \min_x \|\tilde{\Phi}x - y\| \quad (5)$$

where $\tilde{\Phi} \in \mathbb{R}^{m \cdot K}$ consists of the m relevant columns of Φ .

A flowchart of the algorithm is shown in Fig. 2. We note that OMP reconstruction can be divided into two main stages. The first deals with finding the m indices of the non-zero entries of x which corresponds to steps a-f in the above algorithm, and the second deals with finding the values of x at those indices which corresponds to step g.

In our implementation, we assume that $\Phi \in \mathbb{R}^{K \cdot N}$ where $K = 32$, $N = 128$, and that the original signal is (at most) m -sparse with $m = 5$. We use fixed point arithmetic with a word length of 32 bits of which 9 bits are reserved for the integer part and 22 bits are reserved for the fractional part.

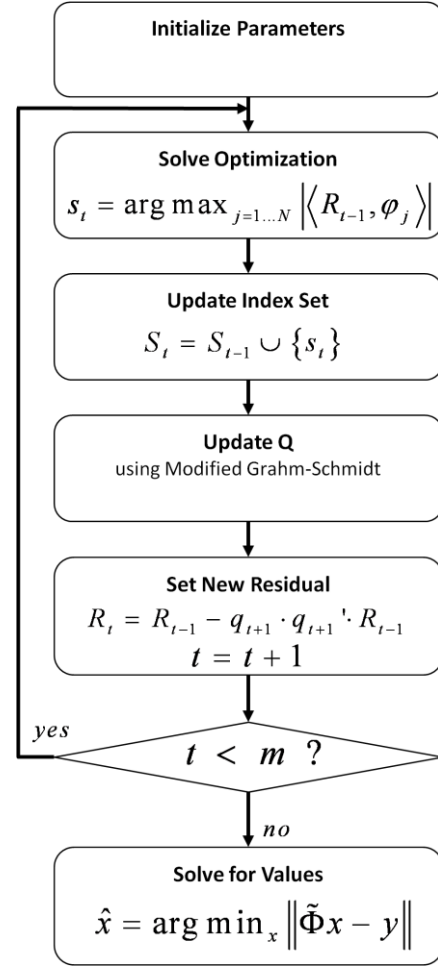


Fig. 2. Flow Chart for OMP

B. Finding the Relevant Indices

The first stage in OMP signal recovery is to find the columns of the sampling matrix Φ which contributed to the measurement vector y . These are the indices at which the original signal x has nonzero entries. This stage requires many matrix-vector, vector-vector, and vector-scalar multiplications. Since each vector and matrix column is of length 32, we designed a chain of 32 multipliers to work in parallel, where each multiplier multiplies two 32-bit numbers. This allows for vector-vector and scalar-vector multiplication in one clock cycle. The matrix-vector multiplication is performed when multiplying the columns of Φ by R in (2). Therefore, each column of Φ is multiplied by R in a separate clock cycle, making the matrix-vector multiplication take 128 clock cycles. In addition, step d generally requires use of the square root operation in the modified Gram-Schmidt normalization. However, we note that both in steps d and e, use of $q_{1 \dots m}$ is always squared. Therefore, in order to save accuracy and area, our implementation avoids the square root by simply storing a multiplicative variable for each $q_{1 \dots m}$.

C. Finding the Relevant Values

As stated earlier, finding the values of the reconstructed signal involves solving the minimization problem given by (5). This equation is generally solved using the Moore-Penrose pseudo-inverse which is defined as:

$$\tilde{\Phi}^\dagger = (\tilde{\Phi}^T \tilde{\Phi})^{-1} \tilde{\Phi}^T \quad (6)$$

If we let $C = \tilde{\Phi}^T \tilde{\Phi}$, then solving (6) involves finding the inverse of C . Particularly, C is a 5x5 matrix which is symmetric and positive definite (SPD). A common way of inverting SPD matrices is to use the Cholesky Decomposition (CD) [9]. However, this involves finding square roots, which wastes time and area. Thus, we chose to use an Alternative Cholesky Decomposition (ACD) which does not require square root calculations. The ACD decomposes a matrix C into the product of three matrices, LDL^T , where L is a lower triangular matrix with ones on its diagonal and D is a diagonal matrix. The procedure for performing the ACD is described by

$$L_{i,j} = \frac{1}{D_{j,j}} (C_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} D_{k,k}), i > j \quad (7)$$

$$D_{i,i} = C_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2 D_{k,k} \quad (8)$$

Fig. 3 shows the dependency graph for (7) and (8). This allows us to see that the most efficient order of operations involves solving for a value of D and then the corresponding column in L , alternatingly. In order to solve for entire columns of L at once, we require four 32-bit multipliers with multiplexors controlling their inputs. At this point, we can solve for C^{-1} using

$$C^{-1} = (L^T)^{-1} D^{-1} L^{-1} \quad (9)$$

The steps involved in inverting C are now inverting the diagonal matrix D which is the inverse of each of its elements on the diagonal, and inverting a lower triangular matrix L . However D^{-1} is already available to us from the division performed in (7). Thus, we must only find L^{-1} , and the procedure for doing so is:

```

for  $j = 1$  to  $n$  do
   $L_{inv}[j, j] = 1/L[j, j]$ 
  for  $i = j + 1$  to  $n$  do
     $L_{inv}[i, j] = -L[i, j : (i-1)]L_{inv}[j : (i-1), j]/L[j, j]$ 
  end for
end for

```

where L_{inv} is the inverse of L and n is the order of the matrix. We note that the inverse of a triangular matrix is itself a triangular matrix, and the diagonal remains ones. In our case where $n = 5$, we again require exactly four 32-bit multipliers. Therefore, we chose to reuse the same multipliers used in the ACD and simply add more inputs to the controlling multiplexors. We define $L_{inv,k}$ to be the diagonal k places from the main diagonal. We observe that the optimal order of

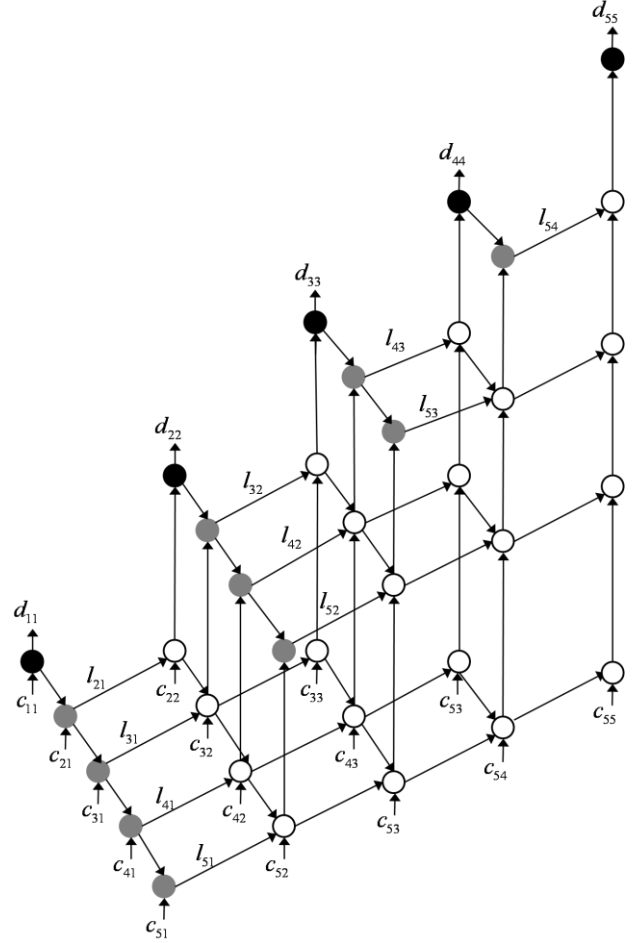


Fig. 3. Dependency Graph for ACD

operations for inverting L involves finding subsequent super-diagonals of L_{inv} . Since L_{inv0} consists of only ones, we find $L_{inv-1}, L_{inv-2}, L_{inv-3}$ and L_{inv-4} in that order.

Finally, we are required to perform the multiplication in (9) in order to find C^{-1} . Since each of the matrices are of size 5x5, it is a simple matter to multiply them. However, we prefer to take advantage of the large number of zeros and ones in these matrices, and perform the minimum amount of MADS needed. We choose to do four multiplications at one time, and use the same four multipliers used in the ACD and inversion of L . The result is a unit which performs efficient inversion of an SPD matrix. A block diagram for the unit is shown in Fig. 4. This matrix inversion unit is a stand-alone unit which receives a start signal from the main controller telling it that the input matrix C is ready, and sends a finish signal in return stating that the matrix inversion is complete and that the output is ready. The inverse of C is then used by the main data path in order to reconstruct the original signal. Fig. 5 shows a block diagram for the entire OMP design.

IV. RESULTS

We implemented our code in VHDL and tested our design with results of a Matlab fixed-point simulation. We tested

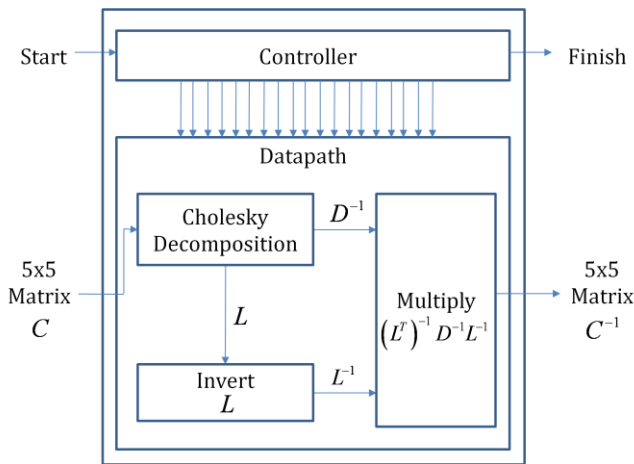


Fig. 4. Matrix Inverse Unit

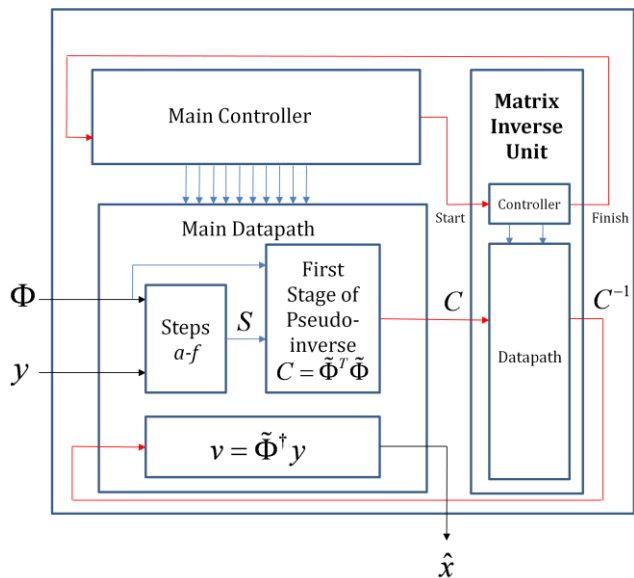


Fig. 5. OMP Unit

sparsity modes using random input vectors, of up to and including a sparsity of $m = 5$. We obtained synthesis and layout results for the Xilinx Virtex-5 FPGA. For a clock frequency of 39 MHz we obtained a usage below 40% of all FPGA units (flip-flops/DSP48e/FIFO/...). For sparsity $m = 5$ we are required to run for 930 clock iterations which is a total run-time of about $24\mu s$. The multiplication of the sampling matrix Φ by the residual vector R in (2) is the bottleneck of the algorithm, as mentioned in [5], and takes $128 \cdot 5 = 640$ clock cycles. The pseudo-inverse unit takes 80 clock cycles and other logic takes 210 clock cycles.

Comparing our results with those of [6], [7] we see that the GPU implementations focused on very large input vectors. Since our basic unit can process a vector of length 128 we need to perform some kind of interpolation to compare our results. Neglecting I/O overhead, and assuming a linear interpolation,

TABLE I
COMPARISON OF HARDWARE RECONSTRUCTION (SIGNAL LENGTH = 128)

	Time [msec]
OMP Hardware	0.024
Intel Core i7, 920 [6]	68
CUBLAS (GPU) [7]	37.5

we obtain a comparison of our system with [6], [7] shown in Table I. We see that our OMP hardware implementation significantly speeds up the signal reconstruction process.

V. CONCLUSION

We have presented a hardware implementation of the OMP algorithm that is capable of reconstructing sparse signals that were sampled using CS techniques. Our design supports vector inputs up to 128 elements with vector sparsity up to 5. Results show that our OMP implementation outperforms other approaches for CS signal recovery. The fast hardware implementation of OMP can be used in different ways. For example, consumer applications can integrate the design within their systems and obtain fast low-power signal reconstruction. We intend to set up a server containing a Virtex 5 FPGA board and offer the research community a means to speed up their CS related experimental results by accessing the server online using a web interface. We look forward to seeing CS based applications integrated in real life products.

ACKNOWLEDGMENT

We would like to thank Goel Samuel for his help with the synthesis, Inna Rivkin for her advice on Xilinx FPGA's and Moshe Mishali for his helpful comments and for providing his Matlab implementation of OMP.

REFERENCES

- [1] E. Candès, "Compressive sampling," in *Proceedings of the International Congress of Mathematicians*, 2006, pp. 1433–1452.
- [2] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Computing*, vol. 24, no. 2, pp. 137–147, 1995.
- [3] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Rev.*, vol. 43, no. 1, pp. 129–159, 2001.
- [4] S. Kunis and H. Rauhut, "Random sampling of sparse trigonometric polynomials, II. orthogonal matching pursuit versus basis pursuit," *Journal Foundations of Computational Mathematics*, vol. 8, no. 6, pp. 737–763, 2007.
- [5] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.
- [6] A. Borghi, J. Darbon, S. Peyronnet, T. Chan, and S. Osher, "Compressive sensing algorithm for parallel many-core architectures," in *CAMS Report*, 2008.
- [7] M. Andreucut, "Fast GPU implementation of sparse signal recovery from random projections," 2008, [http://arxiv.org/PSS/_\\$cache/arxiv/pdf/0809/0809.1833v1.pdf](http://arxiv.org/PSS/_$cache/arxiv/pdf/0809/0809.1833v1.pdf).
- [8] S. Lee and S. J. Wright, "Implementing algorithms for signal and image reconstruction on graphical processing units," 2008, [http://www.optimization-online.org/DBS/_\\$FILE/2008/11/2131.pdf](http://www.optimization-online.org/DBS/_$FILE/2008/11/2131.pdf).
- [9] O. Maslennikov, V. Lepekha, A. Sergiyenko, A. Tomas, and R. Wyrzykowski, "Parallel implementation of Cholesky LL^T - algorithm in FPGA-based processor," *Parallel Processing and Applied Mathematics*, pp. 137–147, 2008.