Nonlinear Equalization for RF Receivers

Brandon Kam, Benjamin Miller, and Joel Goodman
MIT Lincoln Laboratory, Lexington MA
{bkam, bamiller, jgoodman}@ll.mit.edu

Gil Raz
GMR Research & Technology, Concord MA
raz@gmrtech.com

Abstract

This paper describes the need for High Performance Computing (HPC) to facilitate the development and implementation of a nonlinear equalizer that is capable of mitigating and/or eliminating nonlinear distortion to extend the dynamic range of radar front-end receivers decades beyond the analog state-of-the-art. The search space for the optimal nonlinear equalization (NLEQ) solution is computationally intractable using only a single desktop computer. However, we have been able to leverage a combination of an efficient greedy search with the high performance computing technologies of LLGrid and MatlabMPI to construct an NLEQ architecture that is capable of extending the dynamic range of Radar front-end receivers by over 25dB.

1. Introduction

The linear dynamic range of RF receivers is a limiting factor in the performance of many high-end military radar receivers. The nonlinear distortion generated in the analog-to-digital converter (ADC) and the analog receiver limits the capability of the backend radar signal processor to unambiguously detecting targets with weak signatures. By reducing and/or eliminating nonlinear induced distortion after the ADC, NLEQ enables the radar signal processor to detect weak target signals that would otherwise be masked by unwanted nonlinear artifacts.

Because the nonlinear equalizer operates on high-rate (e.g., > 1GB/sec) data directly from the ADC, the goal is to select a robust and computationally parsimonious NLEQ architecture that may be implemented in digital hardware. However, the process of identifying NLEQ architecture is computational onerous due to the vast size of the search space of candidate architectures and the computational complexity of evaluating each candidate.

The potentially astronomical size of the search space for a robust and computationally efficient NLEQ architecture is due to the over-parameterization of the polynomial representation of the nonlinear mapping describing the receiver and ADC transfer functions. Historically such representations were implemented using architectures such as the Volterra filter (see e.g., Reference 1). To overcome limitations in the over-parameterized Volterra representation, we have devised an alternate structure (Partitioned Horizontal Coordinate System - PHoCS) that enables us to efficiently search for and identify a robust and computationally efficient NLEQ architecture. In this paper we describe PHoCS and the methods to identify computationally efficient equalization architectures by using a specialized greedy algorithm running on a large grid computer (LLgrid) using MatlabMPI[6] that is designed to run in parallel on host hardware.

This paper is organized as follows. Section 2 describes the methodology for identifying a robust and computationally efficient nonlinear equalizer using PHoCS; in Section 3 we quantify the computational costs of NLEQ identification. Section 4 presents equalization results after identification as well as benchmarks showing the improved productivity using HPC and in Section 5 we conclude with a brief summary.

2. Methodology

The computational complexity of identifying an optimal NLEQ architecture is quite challenging. Typically the nonlinear behavior of the receive chain is modeled and equalized using a canonical Volterra filter.

Using a Volterra filter model for our equalization problem, however, becomes prohibitively expensive in terms of computational complexity; a \( p \)th order Volterra kernel with a memory depth of \( L \) requires \( \binom{L+p-1}{p} \) coefficients. For example a single 5th-order Volterra kernel with a memory depth of eight requires 792 coefficients. This makes the physical realization of a Volterra-based polynomial filter impractical for real-time implementation at high sample rates.

As an alternative to the Volterra filter method we have formulated the PHoCS architecture[2,8] which is a...
successor to the Diagonal Coordinate System (DCS) architecture. The PHoCS architecture has several advantages over competing structures such as Volterra; in particular we note that despite its nonlinear input/output behavior the equalization coefficients are linear with respect to both the input signal and the output signal thus allowing the use of linear or modified linear techniques for identification and optimization.

The PHoCS architecture is represented by the relationship

\[ y(n) = \sum_{j=1}^{J} y_j^p(n) \]  

\[ y_j^p(n) = u_j^p(n) \sum_{m=0}^{p_j-1} x(n-m) h_j \{ m, \alpha_1^j(j), \ldots, \alpha_{p_j-1}^j(j) \} \]  

where \( x(n) \) is the input signal, \( y(n) \) is the output signal, \( y_j^p(n) \) is an output of a PHoCS Processing Element (PE), \( p_j \) is the kernel order of the \( j \)th processing element, \( g_{l,j}^p(q) \) is the \( l \)th partitioning filter in the \( j \)th PE of a kernel of \( p_j \)th order, and \( \alpha_l^j(n) \) are integer delay values. For an in depth description of the partitioning filters’ \( g_{l,j}^p(q) \) function we refer the reader to Reference 2.

A block diagram for the PHoCS architecture is depicted in Figure 1, and an individual PHoCS PE is depicted in Figure 2. In Figures 1 and 2, and the organic nonlinear equalizer is built by combining the outputs of \( J \) PEs, where \( J \) is the total number of PHoCS processing elements in an NLEQ architecture. The processing elements as shown in Figure 2 are comprised of linear delays, linear filters and multipliers. The filters which are denoted by \( h_j \) are identified and optimized to best mitigate nonlinear distortion of the target receiver.

Software which we developed in Matlab is used to determine the appropriate PHoCS PEs that comprise an NLEQ processor. For a given NLEQ processor, each PE is chosen based on its composite equalization performance from a pool of candidate PHoCS PEs. The total number of PEs selected are limited to fit in target hardware, where variables such as bit width, tap lengths, and filter orders are chosen to achieve a balance between equalization performance and hardware complexity. Because the processor needs to be implemented in hardware, we would like to get the most equalization performance from the least number of PHoCS PEs. So, the objective is to choose \( N \) PHoCS elements from a pool of \( M \) which have the very best performance. However, this search space can be very large; testing all permutations would require deriving coefficients for and testing \( M \) \( N \). For typical values of \( M \) and \( N \) this number can become very large, as Table 1 shows.

<table>
<thead>
<tr>
<th>Typical Values</th>
<th>( \binom{M}{N} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M = 1,200 ), ( N = 20 )</td>
<td>( 10^{42} )</td>
</tr>
<tr>
<td>( M = 11,400 ), ( N = 80 )</td>
<td>( 10^{205} )</td>
</tr>
</tbody>
</table>

To help reduce the problem of searching through an astronomical number of candidate architectures, we have implemented a greedy algorithm that uses an incremental approach to search for a locally optimal solution. In the beginning of the search the algorithm selects from the pool of candidate PEs the PE that best mitigates distortion using a mean square error criterion. Once this PE is determined it is saved, and then the algorithm searches the space again to determine the next PE that works best in conjunction with the previously chosen PE. This process continues until \( N \) PEs is selected. The greedy algorithm...
reduces our search space by potentially many orders of magnitude, since the greedy algorithm requires the examination of approximately $M \times N$ cases. Table 2 highlights the differences in the size of the search space between trying all possible combinations versus implementing the greedy algorithm.

### Table 2. Greedy algorithm provides computational tractability for NLEQ

<table>
<thead>
<tr>
<th>Typical Values</th>
<th>$M \times N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 1,200, N = 20$</td>
<td>$10^{43}$ 24,000</td>
</tr>
<tr>
<td>$M = 11,400, N = 80$</td>
<td>$10^{305}$ 912,000</td>
</tr>
</tbody>
</table>

The greedy algorithm is a logical choice to select PEs to construct NLEQ architecture for two major reasons. The first reason is the drastic reduction in computational complexity described above (see Table 2). We also need to determine how many PEs to use in the NLEQ processor, and the greedy algorithm’s other benefit is the insight it gives us by effectively tackling the “worst” nonlinear distortions first. As a result, we may obtain a good idea of how NLEQ will perform on $N$-1 PEs as opposed to $N$ PEs and weight the performance improvements in distortion mitigation of adding an additional PE against the added hardware complexity needed for its realization.

Because our objective is to specify an NLEQ architecture that will ultimately end up being implemented in a VLSI chip, hardware constraints are central to identify the architecture. The decision was made to use a block floating-point implementation after balancing the trade-off in performance vs. complexity between a floating-point implementation (best performance but high power consumption and large VLSI size) vs. a fixed-point implementation (data varies in magnitude too much for a practical realization). As a compromise, we chose a block floating-point implementation that uses prior knowledge of the input data’s scale to avoid overflows. Further, the software that identifies PEs was designed to determine the appropriate scaling at each stage of computation in the nonlinear equalizer. Finally, the identification software takes into account the effects of quantization when selecting component PEs.

### 3. The Need for High Performance Computing Technologies

While the real-time operation of the equalizer is parsimonious, the complexity of identifying new architectures based on PHoCS remains challenging. Each step in the construction and evaluation of NLEQ architecture requires many floating point operations (FLOPs). We describe these steps as follows.

The first step is that of database construction, modifying the measured signals to be used in identification. Here the major computational bottleneck is the translation of the time domain data to the frequency domain using the FFT. The computational cost of the FFT is given by $5N \log_2(N)$, where $N$ is the number of samples in the input signal. This computation is done repeatedly throughout the first step, where the repetitions are based on $(\#\text{taps}) \times (\#\text{signals}) \times (\#\text{PEs})$. For this example, $N = 258048$, $\#\text{taps} = 12$, $\#\text{signals} = 263$, and $\#\text{PEs} = 5938$, hence the number of FLOPs required for this stage is $\approx 456$ trillion.

In the second stage the software determines the coefficients for each choice of PE. Here we make use of a QR decomposition to efficiently solve the identification equations. While there are other operations as well, the QR-decomposition dominates the complexity. The computation uses modified Gram-Schmidt which requires $2mn^2$ FLOPs. This computation is done multiple times, since the greedy algorithm calculates optimal coefficients for each PE, selects the appropriate one, and then repeats the process. There are approximately $(\#\text{chosen PEs}) \times (\#\text{candidate PEs})$ QR-decompositions completed. For this example, $m = 224170$, $n = 12$, $\#\text{chosen PEs} = 60$ and, $\#\text{candidate PEs} = 5000$, hence the computational complexity for the QR-decomposition is 19 trillion FLOPs.

The third stage checks the performance of the partially identified NLEQ processor by sending all signals through it. The computational cost for this process is given by $\frac{\#\text{FLOPs}}{\text{sample} \times \text{PE}} \times (\#\text{samples}) \times (\#\text{PEs}) \times (\#\text{signals})$. In our example, $\frac{\#\text{FLOPs}}{\text{sample} \times \text{PE}} = 43$, $\#\text{samples} = 258048$, $\#\text{PEs} = \text{sum}(1:60) = 1830$, $\#\text{signals} = 2698$, thus the computational complexity is 55 trillion FLOPs.

Table 3 summarizes the computational cost as described above. Note that these results are a lower bound of computational cost as they only take into account the major operation in each step. We note that other factors besides the pure computational cost affect the run time (e.g., the time spent reading and writing to disk).

### Table 3. Approximate computational cost of a NLEQ simulation

<table>
<thead>
<tr>
<th>Step</th>
<th>1 FFT</th>
<th>Step 2 QRD</th>
<th>Step 3 Filtering</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOPs (trillions)</td>
<td>456</td>
<td>19</td>
<td>55</td>
<td>530</td>
</tr>
</tbody>
</table>
As may be seen, each step requires significant computational power and the simulation would be intractable if not for such enabling HPC technologies as MatlabMPI\(^6\) and the LLGrid\(^7\) supercomputing system.

4. Results

MatlabMPI is a HPC technology that leverages the Message Passing Interface (MPI) standard for parallel computing within the Matlab environment. It was developed to be an easy to use technology that incorporates itself almost seamlessly with Matlab. The source code may be found on the MatlabMPI website.\(^6\) LLGrid is the grid computing system used at MIT Lincoln Laboratory that enables individual users easy access to a powerful computing cluster. The LLGrid and MatlabMPI platforms in conjunction are particularly suitable to our computational needs, as they provide the power of parallel computing in a scientist friendly way.

Paralleling the NLEQ software over the LLGrid cluster significantly reduces the run times. Figure 3 depicts the achieved speedup compared to running on a single processor. For reference we show here the ideal linear speedup. Compared to the run time of 35.4 hours on a single processor for a small NLEQ simulation run, on 4, 16, and 64 processors we may achieve run times of 10.9, 3.9, and 1.3 hours, respectively.

![Figure 3. Speedup from parallelization vs. the “ideal” linear speedup](image)

Figure 3. Speedup from parallelization vs. the “ideal” linear speedup

Figure 4 illustrates the difference in the time to identify an NLEQ architecture using MatlabMPI and 64-processors on LLgrid vs. a single desktop computer. The goal of identification was to construct NLEQ architecture capable of improving the dynamic range of the Radar receiver under test by 27 dB over a 500 MHz bandwidth. Since dynamic range for state-of-the-art ADCs typically improves ~1 dB/year, it would take ~27 years for devices to evolve to match the dynamic range performance that we may achieve today. Note that the identification process is iterative; different receiver/ADC combinations require different NLEQ architectures. The time required to identify an NLEQ architecture on a single desktop computer is well over two months, while it takes only a couple of days on the LLgrid using just 64 processors. Therefore LLgrid and MatlabMPI are enabling technologies for timely identification of NLEQ architectures.

![Figure 4. Approximate time required to identify an NLEQ architecture using 64-processors vs. a single desktop computer](image)

Figure 4. Approximate time required to identify an NLEQ architecture using 64-processors vs. a single desktop computer

5. Summary

In this paper we discussed the issue of linear dynamic range in RF receivers, methods for improving them and the commensurate computational challenges involved. NLEQ has proven to be an effective way of dramatically improving this critical figure of merit, especially in wideband systems. NLEQ’s real-time operation is parsimonious and lends itself readily to cost effective small footprint hardware implementation. However the non real-time computational requirements for identifying the NLEQ structure are large and may overwhelm a stand-alone workstation. Thus, the NLEQ program has benefited significantly from an efficient search algorithm which is parallelizable taking advantage of the HPC capabilities of MatlabMPI and LLGrid.

Acknowledgements

This work is sponsored by the Defense Advanced Research Projects Administration under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.
References