GPU-Based Linearization of MIMO Arrays

Chance Tarver*, Arav Singhal†, and Joseph R. Cavallaro†
*†Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA
*Standard and Mobility Innovation Lab, Samsung Research America, Plano, TX, USA

***Abstract*—In this paper, we present a graphics processing unit (GPU)-based implementation for linearizing the power amplifiers (PAs) in massive multiple-input multiple-output (MIMO) arrays leading to lower error vector magnitude for the users and lower adjacent channel leakage ratio at the output of each antenna. In wireless transmitters, the nonlinearities of PAs can cause undesired spectral regrowth into the adjacent channels. For single antenna communications, this is corrected by digitally predistorting the transmit signal with the inverse nonlinearities of the power amplifier. However, in 5G and beyond, MIMO systems may have over one-hundred antennas and PAs that need to be linearized. Scaling up digital predistortion so that it can be performed on every transmit chain in large antenna arrays creates a significant computational burden for the base station. The parallel processing structure of GPUs provides a commercially available off-the-shelf solution that can be used to efficiently implement digital predistortion across many PAs in a massive MIMO basestation. Such a software-based solution is particularly attractive in virtual radio access networks or other software-defined radio scenarios. In this paper, we examine how the widely used memory polynomial scales on a GPU as the number of antennas scales up to 128, the number of memory taps scales up to four, and the polynomial degree scales up to nine. We find that a mid-range GPU can support predistortion for a sample rate of 100 MSps for up to thirty-two antennas while using a seventh-order polynomial with two memory taps.**

***Index Terms*—DPD, GPGPU, Massive MIMO**

I. Introduction

Massive multiple-input multiple-output (MIMO), where basestations may have more than 32 transmit antennas, is the most exciting physical layer (PHY) technology trend in 5G and beyond. With massive MIMO comes lofty promises and expectations of improved spectral efficiency, network capacity, and even energy efficiency [1]. However, the energy efficiency of the components that may consume the most power in the basestation, power amplifiers (PAs), is often overlooked in these discussions.

When operating in a linear region, many PAs will have efficiencies near 20%. While in saturation, many PAs may have energy efficiency as high as 60% [2]. However, the nonlinearities experienced in this region of operation may be severe, which leads to decreased error vector magnitude (EVM) and increased adjacent channel leakage ratio (ACLR), which can violate standards' requirements [3]. The power overhead when accounting for the PA efficiency can quickly become a problem as the number of PAs scales up, creating further design challenges such as heat management.

To better achieve the desired energy efficiency claims for massive MIMO, it is necessary to operate each PA near saturation. Hence, it is also necessary to perform digital predistortion (DPD) to linearize each PA and maintain spectral requirements. However, the computation to perform DPD scales poorly for a massive MIMO basestation making computational efficient implementations a necessity in that one basestation has to linearize all transmit chains and potentially compensate for crosstalk in the array [4]. Moreover, bandwidths as wide as 100 MHz may be utilized in sub-6 GHz 5G, leading to more memory effects to account for in DPD processing and hence more complexity.

PAs and their linearization in massive MIMO have recently received attention in the literature. [4] showed the effects of crosstalk and presented an algorithm to compensate for it though that analysis, being early in the field, was done for 4x4 MIMO. The presented crossover DPD algorithm does not scale well for many antennas. In [2], Doherty PAs are presented as a good candidate for MIMO basestations due to their high energy efficiency. However, they also have poor nonlinearity, and the author discusses the need for power-efficient linearization schemes. [5] discusses linearizing a virtual PA along the main beam. However, it is unclear how this method scales for orthogonal frequency-division multiplexing (OFDM) data where there may be separate precoding for each subcarrier. Moreover, it is unclear how it scales for multiple users.

Although there are a few works with algorithmic contributions, there is currently no, to the best of the authors' knowledge, implementation results. In this paper, we present a novel graphics processing unit (GPU)-based implementation, where we linearize each transmit chain using memory polynomials (MPs) [6]. This implementation is intended to provide a baseline for complexity and achievable rates for MP-based DPDs in a massive MIMO scenario.

GPUs are attractive for implementation due to their high degree of parallelism and ease of programming when compared to other high-performance computing devices like field-programmable gate arrays (FPGAs). GPUs are also considered in other areas of software-defined radio (SDR) physical layers, including for MIMO processing. For example, in [7], GPUs are used for detection and beamforming in a multi-user (MU) MIMO base station. In [8], GPUs are used for LDPC decoding. They have also previously been used for DPD [9] [10]. By porting more functionality into GPUs, the benefits of the GPU can be further realized as the data can stay on the GPU longer, avoiding time-consuming memory transfers into and out of the device.

978-1-7281-8099-1/20/$31.00 ©2020 IEEE

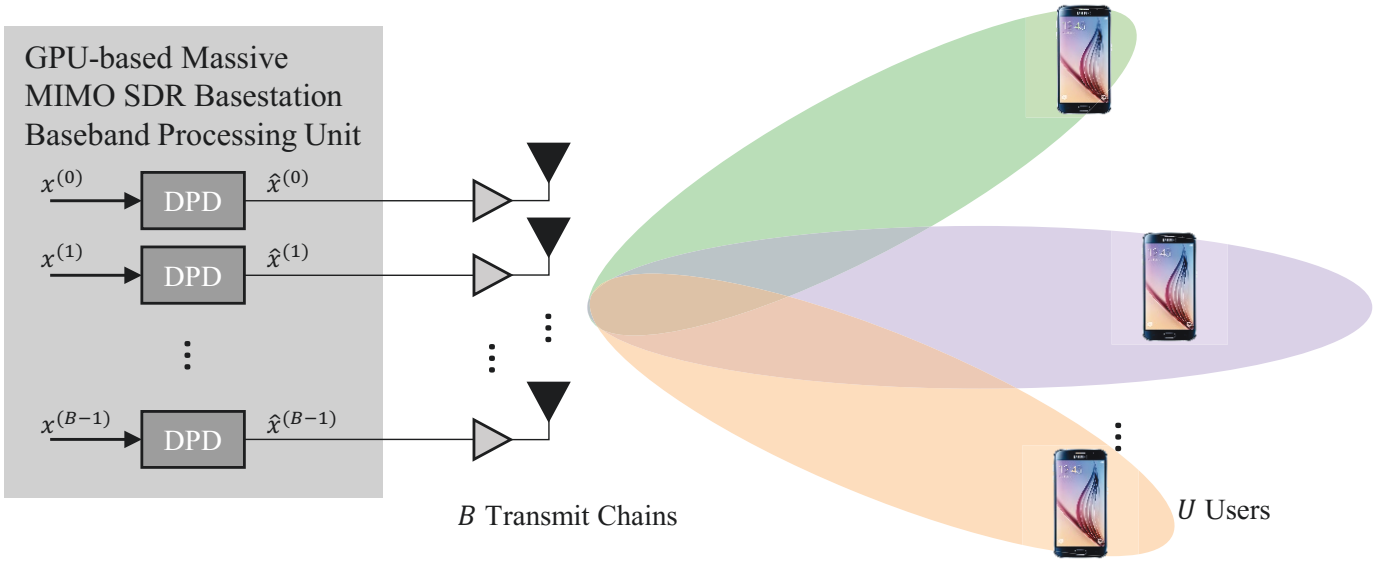

Figure 1. System Diagram. Within the GPU-based massive MIMO SDR baseband processing unit, computation for each TX chain is performed in parallel on the GPUs, allowing for predistortion and improved spectral containment for wide bandwidths and large antenna arrays.

II. GPU Computation

A GPU consists of an array of streaming multiprocessors (SMs), which each often contain 32 or 64 vector processing lanes. NVIDIA provides C++ language extensions, known as CUDA, to write highly parallel applications that target these devices. The developer writes kernels that will utilize a certain number of blocks and threads, which are roughly a programming abstraction of the SMs and compute unified device architecture (CUDA) cores. There are multiple tiers of memory on the GPU. Global memory is typically a GDDR based memory that is available to all kernels over the lifetime of the application. Threads in a block all execute on the same SM and can have a shared memory for the lifetime of the kernel. Each thread also has local registers for intermediate results. There is also a constant memory that is globally available to all threads, but it is read-only. The global memory is typically the slowest, so it is desirable to put data in constant and shared memory whenever possible.

III. Memory Polynomial Implementation Overview

We implement a memory polynomial for each PA [6],

$$\hat{x}^{(i)}(n) = \sum_{p=1}^{P} \sum_{m=0}^{M} \beta_{p,m}^{(i)} x^{(i)}(n-m) \left| x^{(i)}(n-m) \right|^{p-1}, \quad (1)$$

where $x^{(i)}(n)$ is the original baseband input signal for the ith PA, $\hat{x}^{(i)}(n)$ is its predistorted input calculated by the MP, P is the highest polynomial order considered, M is the highest memory depth considered, and $\beta_{p,m}^{(i)} \in \mathbb{C}$ represents the scalar coefficients that are used to tune the DPD for the ith PA.

(1) can be expanded into a matrix-vector product for a block of N samples, as shown in (2) – (5). When formulated as many independent matrix-vector products, the mapping to the GPU becomes clear, as will be discussed in the next section.

A. Architecture of the massive MIMO GPU-DPD

An overview of our architecture for the massive MIMO GPU-DPD is shown in Fig. 1. This figure shows how input samples are supplied into the GPU kernel, where each thread works on one output element in parallel while threads within a block work on data for the same PA.

1) Memory System: Effective memory management is critical for GPU software design to reduce unnecessary data movement. In our architecture, we place the DPD coefficients, $\boldsymbol{\beta}^{(i)}$, in a 2D array in constant memory. This use of constant memory allows for fast access to the coefficients during computation.

The input and output data is placed in global memory. To improve any transfers into global memory from the host, we pin any of the corresponding arrays in host memory.

2) Computation: The primary strategy taken in our GPU architecture is that each thread computes one element of the output. We use one kernel, outlined in Kernel 1, to perform all computation. When a thread is created, it will get its thread and block index and use this to identify which element, n, in the output data it will compute and which PA, i, it will be predistorting. Using these indexes it will access multiple elements of the input array depending on the memory depth, M.

For each input sample into a thread, the thread will perform the polynomial expansion through an unrolled *for* loop. It begins by computing the magnitude of the complex envelope, $|x(n-m)|^2$. As it moves onto higher polynomial orders, it reuses previous calculations from lower orders to fill out an array corresponding to a row of $\mathbf{X}^{(i)}$ in (5).

The primary advantage of this method is that by using a single kernel, we do not need to store intermediate data inside the slower global memory. Each intermediate computation exists in a thread's fast, local memory. This method does

$$\mathbf{x}_{p,m}^{(i)} = \begin{bmatrix} \mathbf{0}_{1\times m} & x^{(i)}(0)|x^{(i)}(0)|^{p-1} & x^{(i)}(1)|x^{(i)}(1)|^{p-1} & \dots & x^{(i)}(N-1)|x^{(i)}(N-1)|^{p-1} & \mathbf{0}_{1\times M-m} \end{bmatrix}^T \quad (2)$$

$$\boldsymbol{\beta}^{(i)} = \begin{bmatrix} \beta_{1,0} & \dots & \beta_{1,M-1}^{(i)} & \beta_{3,0}^{(i)} & \dots & \beta_{P,M-1}^{(i)} \end{bmatrix}^T \quad (3)$$

$$\mathbf{X}^{(i)} = \begin{bmatrix} \mathbf{x}_{1,0}^{(i)} & \dots & \mathbf{x}_{1,M-1}^{(i)} & \mathbf{x}_{3,0}^{(i)} & \dots & \mathbf{x}_{P,M-1}^{(i)} \end{bmatrix} \quad (4)$$

$$\hat{\mathbf{x}}^{(i)} = \mathbf{X}^{(i)}\boldsymbol{\beta}^{(i)} \quad (5)$$

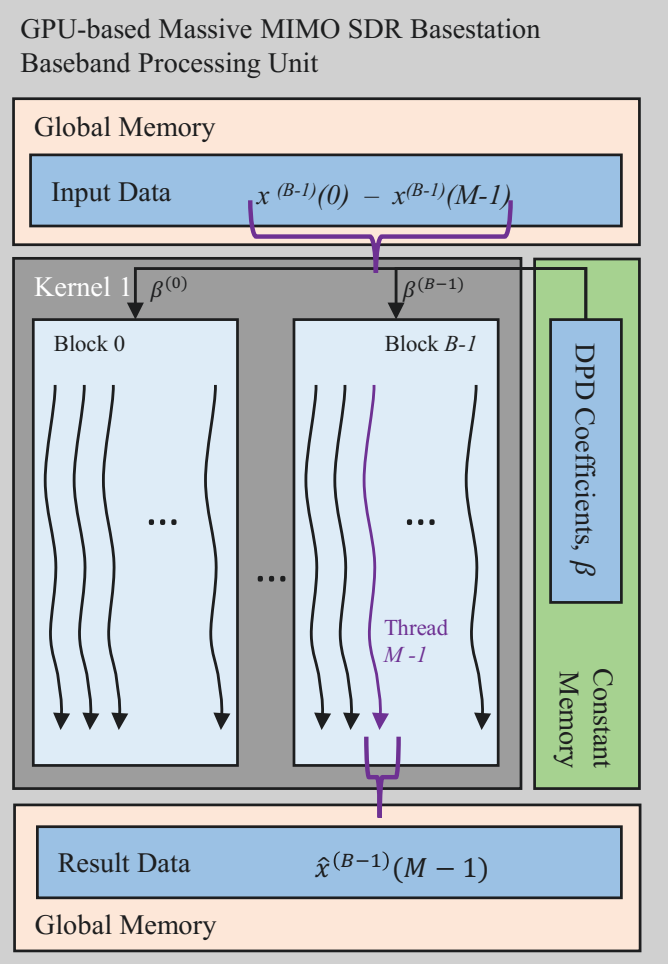

Figure 2. GPU Architecture Diagram. Each thread computes one output sample. Each block computes the result for one PA in the array.

result in adjacent threads within a window of M computing the same intermediate steps. However, by not sharing results, the threads do not need to synchronize which results in higher performance for polynomials with a short memory depth.

IV. Results

A. Memory transfer

A critical part of any GPU system is getting data to and from the device. Most commercial off-the-shelf GPUs use PCIe3, which is limited to 16 GBps in each direction. This constraint can be alleviated by using multiple CUDA "streams" to perform memory transfers at the same time as computation. However, this technique can only help to keep the bus fully utilized over time and does not change the fundamental IO limit of PCIe. Nonconsumer GPUs may use a higher speed bus such as Infiniband, which increase the IO throughput. In the case of DPD, we ideally would immediately stream the upsampled, predistorted signals through a digital-to-analog converter (DAC) for each PA. To the best of the authors' knowledge, such a platform does not currently exist. However, a system could use the GPU-direct RDMA for a direct network interface controller (NIC) connection streaming to an FPGA to handle sending data to a DAC. With the appropriate hardware, such RDMA systems have IO capabilities on the order of 100 Gbps[11].

In this work, we omit the data-transfer time to-and-from the GPU in our results as the PCIe bus available on our GPUs is not sufficient for moving upsampled IQ data for multiple PAs at large sampling rates. Instead we provide an examination of the on-device time. Alternatively, if the number of antennas is limited to be fewer than 16, predistortion could be done for reasonable bandwidths with limited IO buses such as PCIe. This option could be viable for decentralized massive MIMO, such as in [7].

Kernel 1 DPD Processing Thread

```
1:  i ← Block Index
2:  n ← Thread Index
3:  load β^(i) from constant memory
4:  m = 0; p = 1;          // Memory and polynomial indexes
5:  for m < M do           // Loop over memory depth
6:      y ← x^(i)(n − m)   // Loaded from global memory
7:      z[m] = y
8:      a = |y|^2
9:      b = y · a
10:     for p <= P do
11:         z[pM + m] = b
12:         b = b · a
13:     end for
14: end for
15: x̂^(i)(n) = z · β^(i)
16: return x̂^(i)(n)
```

B. DPD Coefficients

To collect DPD coefficients, $\boldsymbol{\beta}^{(i)}$, we use a standard indirect learning architecture and least-squares fit to solve for a variety

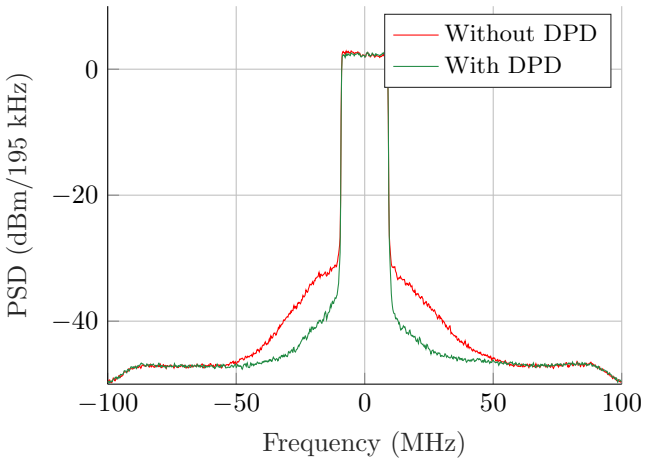

Figure 3. Example DPD result from RFWebLab. By applying DPD, the spectral regrowth around the main carrier can be reduced.

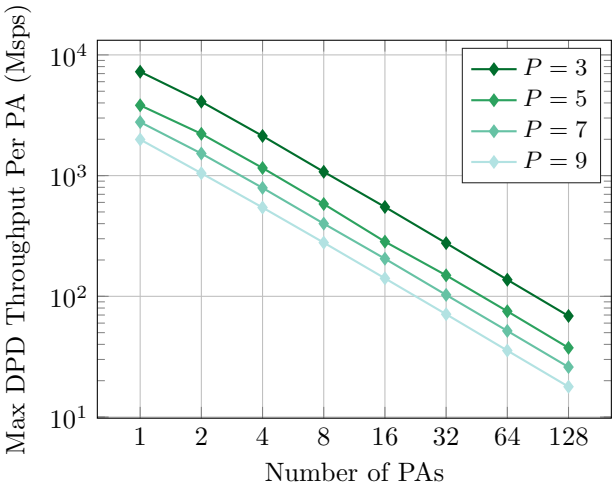

Figure 4. Throughput versus the number of PAs for varying DPD nonlinearity order. For this test, $M = 2$, and we use floating-point data types. The GPU under test was an Nvidia 1080.

of polynomials using RFWeblab [12]. An example DPD result is shown in Fig. 3. For the trained DPD, we then made N copies and added random variability into the coefficients to mimic the variations that could occur between PAs in a massive MIMO array. This set of coefficients was stored as a binary to be read at the start of the GPU testbed experiment and loaded into constant memory.

a) Throughput vs number of antennas: In Fig. 4, we show the performance as a function of the number of antennas on a log-log plot. The results show that as we double the number of PAs that we are performing DPD computations for, we get a halving in the maximum sample rate that can be linearized per PA. This trend is due to the memory bandwidth constraint between the GDDR5 memory and the computational units. For this test, an Nvidia 1080 GPU was used, which has a memory bandwidth of 320 GB/s. Newer GPUs such as the Nvidia 2060 and Titan RTX have memory bandwidths as high as 672 GB/s and could likely be used to nearly double these measurements. Moreover, by using smaller width data types such as half-precision, the amount of data moving could be halved, potentially doubling performance.

For DPD with nonlinear order $P = 3$, the max throughput when computing only for 1 PA is 7252 MSps. For 5G signal bandwidths of 100 MHz, it may be necessary to linearize per PA at a rate of 500 MSps. This sample rate can be supported for up to $B = 16$ antennas. For the maximum LTE bandwidth of 20 MHz, it may be necessary to linearize per PA at a rate of 100 MSps. This rate can be supported for up to $B = 64$ antennas.

b) Throughput vs. degree of polynomial: In Fig. 4, we also show the performance as a function of the nonlinearity order P. This does not increase data movement to and from the SMs. Instead, it adds to the computation of the kernel. As we increase the degree of the polynomial, the increased computation in Kernel 1 is one additional real multiply when expanding the polynomial and an additional term when computing the dot product.

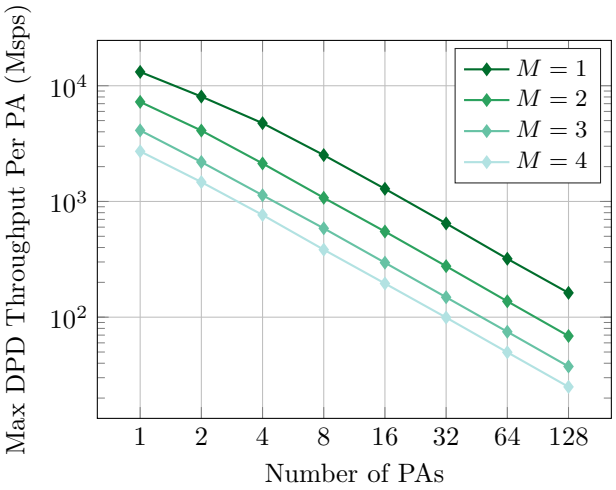

Figure 5. Throughput versus the number of memory taps, M on the Nvidia 1080. Here we fix $P = 3$ and use floating-point data types throughout.

For $P = 3$ and $B = 1$, we see a max throughput of 7252 MSps. By going to $P = 5$, we drop to 3822 MSps which is nearly a 50% reduction in lienarization rate. At $P = 7$ we see 2776 MSps throughput per PA, a 27% drop in linearization rate. This trend stays consistent for more PAs and higher polynomial orders. In most PA platforms, the third-order nonlinearity is dominant and linearizing it may be sufficient to reach ACLR requirements.

c) Throughput vs. number of memory taps: In Fig. 5, we present the measurements as the number of memory taps are varied from $M = 1$ to $M = 4$. More memory is used in practice to help linearize wider bandwidths. We fix $P = 3$ for this test and continue to use the Nvidia 1080 GPU.

For this test, we again see that the max possible linearization rate per PA is halved for every doubling in the number of PAs. As we increase M, each thread accesses one more data element and performs the corresponding polynomial expan-

Table I
GPU SPECIFICATION AND RESULT COMPARISON

GPU	Year Released	Architecture	Number of CUDA Cores	Memory Speed (Gbps)	$P=3, M=1$ 1 PA	$P=3, M=1$ 128 PA	$P=7, M=4$ 1 PA	$P=7, M=4$ 128 PA
GTX 1080	2016	Pascal	2560	10	13168	162	1066	8.6
RTX 2060	2019	Turing	1920	14	11860	286	1839	15.3

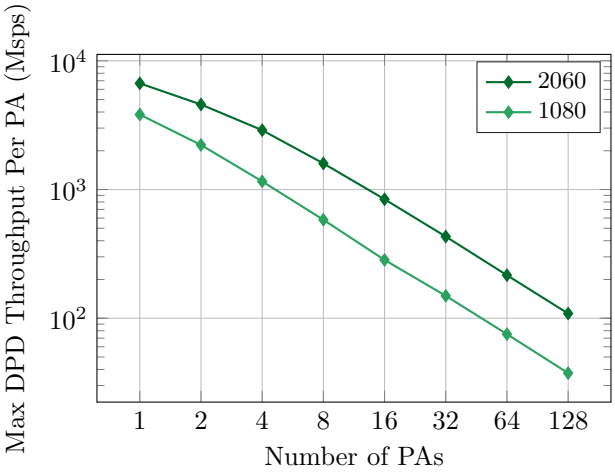

Figure 6. Throughput versus the GPU model. We fix $P = 5$, $M = 2$, and use floating point data types throughout.

sion. Hence, increasing M increases the computational load per thread and the amount of memory movement. For large number of M, it may be desirable to re-architect the kernel to avoid the duplication of computation, storing intermediates in shared memory.

For the memoryless case, $M = 1$, the max achievable linearization rate for 128 antennas was 168 MSps, which is more than sufficient for 20 MHz signal bandwidths. For each additional memory tap, there is a consistent 40% reduction in the maximum achievable linearization rate.

d) Throughput vs. GPU: In Fig. 6, we present the performance for various GPUs. We fix the MP to $M = 2$ and $P = 5$. We consider an Nvidia 2060 and 1080 whose key technical specifications are shown in Table I. In general, the newer series, 20xx, has faster memory while the higher model numbers, i.e xx80, have higher compute capability.

In the case of DPD, the system is largely memory bound. Hence, in Fig. 6 we see that the RTX 2060 outperforms the 1080 by a factor of 1.7 to 2.9. The gap increases as the number of PA increases due to the increase in memory transfers and better specifications regarding memory in the 2060.

Table 1 also shows the performance for each GPU at each extreme of DPD operation. We show the case of a $P = 3, M = 1$ and $P = 7, M = 4$ memory polynomials. The RTX 2060 outperforms in all tests except where for the smaller memory polynomial with only 1 PA. In that case, the ratio of memory movement to computation is not favorable, and the GPU with a higher number of CUDA cores outperforms.

V. CONCLUSIONS

In this paper, we present an MP-based predistorter for SDR implementations of massive MIMO. We show that GPU platforms can effectively perform the many parallel computations for each transmit chain and could make flexible solutions for MIMO basestations. However, this work also highlights the enormous IO challenges in massive MIMO and the need for algorithmic innovation for linearization of arrays. For cases where canceling the third-order nonlinearity is sufficient, memory effects are not a concern, and the expected signal bandwidths are around 20 MHz, the GPU solution can linearize for up to 128 antennas. For future work, we will consider reduced precision computation, crosstalk impairments, and larger memory polynomial structures such as the generalized memory polynomial.

REFERENCES

[1] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 186–195, 2014.

[2] W. Chen, G. Lv, X. Liu, D. Wang, and F. M. Ghannouchi, "Doherty PAs for 5G massive MIMO: Energy-efficient integrated DPA MMICs for sub-6-GHz and mm-wave 5G massive MIMO systems," *IEEE Microw. Mag.*, vol. 21, no. 5, pp. 78–93, 2020.

[3] 3GPP, "5G; NR; Base Station (BS) radio transmission and reception," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.104, 2018, version 15.2.

[4] S. A. Bassam, M. Helaoui, and F. M. Ghannouchi, "Crossover digital predistorter for the compensation of crosstalk and nonlinearity in MIMO transmitters," *IEEE Trans. on Microw. Theory and Techniques*, vol. 57, no. 5, pp. 1119–1128, 2009.

[5] A. Brihuega, L. Anttila, M. Abdelaziz, and M. Valkama, "Digital predistortion in large-array digital beamforming transmitters," in *Asilomar Conf. Signals, Syst., and Comput.*, 2018, pp. 611–618.

[6] L. Ding, G. T. Zhou, D. R. Morgan, Z. Ma, J. S. Kenney, J. Kim, and C. R. Giardina, "A robust digital baseband predistorter constructed using memory polynomials," *IEEE Trans. Commun*, vol. 52, no. 1, pp. 159–165, Jan 2004.

[7] K. Li, "Decentralized baseband processing for massive MU-MIMO systems," Ph.D. dissertation, Dept. of Elect. and Comput. Eng., Rice University, 2019.

[8] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 309–322, Feb 2011.

[9] K. Li, A. Ghazi, C. Tarver, J. Boutellier, M. Abdelaziz, L. Anttila, M. Juntti, M. Valkama, and J. R. Cavallaro, "Parallel digital predistortion design on mobile GPU and embedded multicore CPU for mobile transmitters," *J. Signal Process. Syst.*, vol. 89, no. 3, pp. 417–430, 2017.

[10] P. Campo, V. Lampu, A. Meirhaeghe, J. Boutellier, L. Anttila, and M. Valkama, "Digital predistortion for 5G small cell: GPU implementation and RF measurements," *J. Signal Process. Syst.*, vol. 92, pp. 475–486, 2020.

[11] "Mellanox OFED GPUDirect RDMA." [Online]. Available: https://www.mellanox.com/products/GPUDirect-RDMA

[12] "RF WebLab." [Online]. Available: http://dpdcompetition.com/rfweblab/