Mobile GPU Accelerated Digital Predistortion on a Software-defined Mobile Transmitter

Kaipeng Li*, Amanullah Ghazi†, Jani Boutellier†, Mahmoud Abdelaziz‡, Lauri Anttila‡,
Markku Juntti†, Mikko Valkama‡, Joseph R. Cavallaro*
*Rice University, Dept. Electrical and Computer Engineering, Houston, TX, USA
†University of Oulu, Dept. Computer Science and Engineering, Finland
‡Tampere University of Technology, Dept. Electronics and Communications Engineering, Finland

***Abstract*—We present the design exploration and the performance evaluation of a mobile transmitter digital predistortion (DPD) module on a mobile GPU. Digital predistortion is a widely used technique for suppressing the spurious spectrum emission caused by the imperfection of power amplifier and radio frequency (RF) circuits in a real wireless transmitter. Considering the parallel architecture, numerous computing cores and programmability of GPU, in this work, a DPD design based on augmented parallel Hammerstein structure is implemented on a mobile GPU integrated in an Nvidia Jetson TK1 mobile development board, targeting at a mobile transmitter. The algorithm level and data level parallelism are carefully explored for efficient mapping of the DPD algorithm and full utilization of the mobile GPU resources. We analyze the throughput and timing performance of our implementation and verify the functionality of DPD experimentally on a novel software-defined mobile terminal. The results show that our proposed mobile GPU driven digital predistortion design not only achieves real-time high performance, but also offers programmability and reconfigurability for design upgrading and extension.**

***Index Terms*—mobile GPU, digital predistortion, software-defined radio, cognitive radio, reconfigurability**

I. Introduction

Current wireless transceivers usually utilize the direct conversion radio architecture, which can perform the up-conversion and down-conversion of complex in-phase and quadrature (I/Q) signals [1], to achieve low cost and high performance. However, impairments may exist in such transceivers, such as nonlinearity of power amplifier (PA), I/Q imbalance, local oscillator (LO) leakage, etc, resulting from the imperfection of analog RF and digital baseband circuits. At the transmitter (TX) side, when driving a PA to its saturation region for better signal coverage and efficiency, the nonlinear PA effects will cause spurious spectrum emission and intermodulation distortion (IMD), which are especially harmful in non-contiguous carrier aggregation (CA) scenarios in LTE-Advanced systems [2]. The promising cognitive radio (CR) system also requires proper control of spurious spectrum components to avoid the violation of the interference constraints between secondary and primary user [3].

Recently, adaptive digital predistortion (DPD) [4] [5] is shown to be an effective method for suppressing the unwanted spectrum at the PA output by predistorting the signals at digital baseband before passing through the PA. However, inserting a DPD module in a transmitter baseband will introduce extra baseband processing complexity, and sacrifice the throughput and latency performance at a transmitter. In addition, the digital predistorter parameters, which have a significant effect on the DPD suppression functionality, are expected to be reconfigurable, so that they can be easily updated to adapt to the various PA conditions and wireless communication standards. Therefore, DPD implementations with both high performance and high flexibility are needed in the context of software-defined radio (SDR) systems and CR systems.

General-purpose computing on graphics processing units (GPGPU) [6] has been seen as an alternative for accelerating baseband processing in SDR systems. Compared to conventional baseband components implemented in FPGA or ASIC, GPU can not only achieve high throughput on computationally intensive workloads, such as LDPC decoding [7], MIMO detection [8], etc, but also offer high-level programmability to simplify the baseband design and upgrade.

Currently, the existing GPU accelerated baseband components are all based on desktop GPU devices, targeting at a software-defined basestation [9] [10]. With the development of embedded computing platforms, we are motivated to investigate the capability of mobile GPU on accelerating a parallel DPD design for a software-defined mobile terminal. In this paper, we present the first–to the best of the authors' knowledge–published **mobile GPU** driven application in the wireless communication area. The proposed DPD design on mobile GPU is able to achieve real-time performance on predistorting streaming baseband samples for a mobile transmitter. We also show its high reconfigurability for supporting various transmission signal types, such as single LTE carrier or non-contiguous LTE component carriers (CC) in CA scenarios, and for easily updating DPD parameters adapted to various transmission environments and radio hardware. Furthermore, the functionality of the DPD design is experimentally verified on our novel software-defined mobile terminal platform, which is constructed by an Nvidia Jetson TK1 board [11] integrated with a mobile GPU and a WARP [12] radio board integrated with PA and radio interfaces, to show the DPD suppression effect incorporating a real PA. In addition, our design can be extended to a desktop GPU for a software-defined basestation.

The paper is organized as follows. Section II introduces the DPD algorithm to be implemented. Section III illustrates the GPU implementation details and the experimental verification of DPD functionality on our software-defined mobile transmitter. Section IV discusses the major results and Section V draws the conclusion.

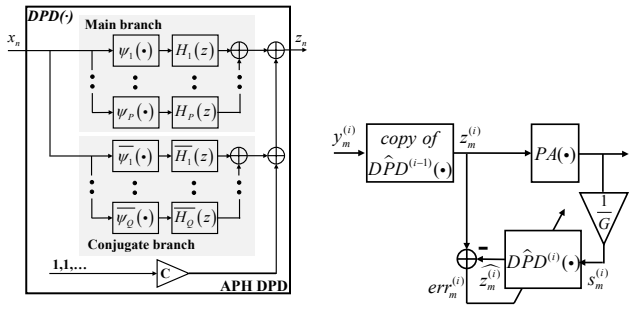
II. DPD algorithm

We adopt the DPD algorithm developed in our previous work [4], which can jointly estimate the major analog impairments, i.e., the PA distortion, I/Q imbalance, and LO leakage, in a direct-conversion transmitter. This algorithm achieves good DPD suppression effect according to simulation results [4], and has inherent parallelism and manageable complexity for efficient implementation. The whole DPD process includes two stages: an iterative training stage for DPD parameter estimation and a predistortion stage for predistorting new samples with finalized DPD parameters.

The predistorter is based on an augmented parallel Hammerstein (APH) structure [4], as shown in Figure 1(a). Assume we have N modulated I/Q samples x_0, x_1, x_2, $\cdots$, x_{N-1} to transmit, instead of sending them directly to the PA and radio, we can first pass them through the DPD module to generate the predistorted samples z_0, z_1, z_2, $\cdots$, z_{N-1} at baseband. Then the predistorted samples are sent to the PA and radio to jointly compensate for the major impairments at TX. Here, the relationship between a certain DPD input x_n (n=0,1,2,$\cdots$,N-1) and DPD output $z_n = DPD(x_n)$ is:

$$z_n = \sum_{\substack{p=1 \\ p\ odd}}^{P} \sum_{k=0}^{L_p} h_{p,k}\psi_p(x_{n-k}) + \sum_{\substack{q=1 \\ q\ odd}}^{Q} \sum_{k=0}^{L_q} \bar{h}_{q,k}\bar{\psi}_q(x_{n-k}) + c$$

This work was supported by the US NSF under grants ECCS-1408370, CNS-1265332, ECCS-1232274, and the Finnish Agency of Innovation, Tekes



(a) APH DPD architecture (b) Indirect learning architecture

Figure 1. DPD architecture

Here, $\psi_p(x_n) = |x_n|^{p-1}x_n$ and $\bar{\psi}_q(x_n) = \psi_q(x_n^*) = |x_n|^{q-1}x_n^*$ are the p^{th} order polynomial of direct signal x_n and the q^{th} order polynomial of conjugate signal x_n^*, respectively, and only odd order polynomials are considered in this model, that is, $p \in \{1,3,5,\cdots,P\}$, $q \in \{1,3,5,\cdots,Q\}$. P and Q are the highest polynomial orders of the main and conjugate branches respectively. Generally, the conjugate signals stemming from I/Q imbalance are clearly weaker than direct signals, so we can set, for example, $P = 5$ and $Q = 3$, without losing much predistorting effect. $h_{p,k}$ and $\overline{h}_{q,k}$ indicate the k^{th} impulse response of L_p tap FIR filter $H_p(z)$ and L_q tap FIR filter $\overline{H}_q(z)$, respectively, and c is the compensator coefficient considering the LO leakage. We define APH filter coefficient vectors $\mathbf{h_p}$=$[h_{p,0}\ h_{p,1}\cdots h_{p,L_p-1}]^T$ and $\bar{\mathbf{h}}_\mathbf{q}$=$[\bar{h}_{q,0}\ \bar{h}_{q,1}\cdots \bar{h}_{q,L_q-1}]^T$, and stack all $\mathbf{h_p}$ and $\bar{\mathbf{h}}_\mathbf{q}$ as well as c together to form a single coefficient vector $\mathbf{h}$=$[\mathbf{h_1^T}, \mathbf{h_3^T}\cdots \mathbf{h_P^T}\ \bar{\mathbf{h}}_\mathbf{1}^\mathbf{T}\ \bar{\mathbf{h}}_\mathbf{3}^\mathbf{T}\cdots \bar{\mathbf{h}}_\mathbf{Q}^\mathbf{T}\ c]^T$. The coefficient vector $\mathbf{h}$ needs to be estimated during the iterative training stage until convergence to finalize the APH DPD, which can be used to predistort all the following samples in the predistortion stage until another retraining is needed in another PA condition or environment.

The DPD parameter estimation stage is based on an indirect learning architecture (ILA) [13], which is shown in Figure 1(b). A feedback loop is established for iterative training. In the i^{th} iteration, we stream M training samples $y_0^{(i)}, y_1^{(i)}, y_2^{(i)}\cdots y_{M-1}^{(i)}$ through the DPD function $D\hat{P}D^{(i-1)}(\cdot)$ estimated from the $(i\text{-}1)^{th}$ iteration. For each sample $y_m^{(i)}$ (m=1,2,3 $\cdots$ M-1), we calculate the predistorted samples $z_m^{(i)} = D\hat{P}D^{(i-1)}(y_m^{(i)})$ and send $z_m^{(i)}$ to the PA (in the first iteration, $z_m^{(1)} = y_m^{(1)}$). Then we measure the samples $s_m^{(i)}$ scaled by the PA gain G in the feedback loop and update the filter coefficient vector $\mathbf{h}$ of $D\hat{P}D^{(i)}(\cdot)$ based on the least squares (LS) estimation [4]. Finally we insert the estimated $D\hat{P}D^{(i)}(\cdot)$ in the TX chain for the $(i\text{+}1)^{th}$ iteration. 1-3 iterations are usually needed for $\mathbf{h}$ to converge and for finalizing the DPD parameters used in the predistortion stage.

For a certain PA in an environment without many fluctuations, it is not necessary to retrain the DPD frequently, so the training stage can be performed offline. Once the DPD parameters are finalized, real-time predistortion of the new streaming samples is required. Therefore, in the following section, we focus on the parallel GPU implementation of the finalized APH DPD in the predistortion stage. Considering the high flexibility of our implementation, we can easily update the DPD parameters once a retraining does happen.

III. DPD IMPLEMENTATION AND EXPERIMENTAL VERIFICATION

In this section, we first discuss the implementation details of the APH DPD on a CUDA enabled mobile GPU. Then we experimentally verify the functionality of DPD on a novel software-defined mobile transmitter which integrates such a mobile GPU and also includes a radio platform. It is easy to transfer and extend our mobile GPU driven DPD design to a desktop GPU for supporting real-time DPD processing targeting at a potential 5G software-defined basestation, which requires much higher baseband performance for realizing wideband [14] techniques, so a supplementary implementation on a desktop GPU is also discussed here and profiled in Section IV for completeness.

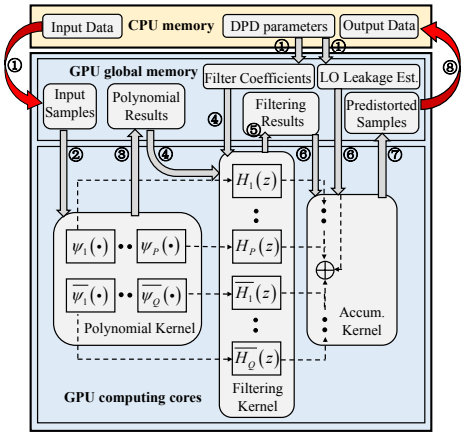


Figure 2. High-level dataflow diagram

A. Experimental hardware

We use an Nvidia Jetson TK1, a mobile development board, for our mobile GPU implementation. The Jetson TK1 board is integrated with an Nvidia Tegra K1 28nm SOC including a Kepler mobile GPU GK20A (192 CUDA cores) and 4-Plus-1 quad-core ARM Cortex A15 CPU. Nsight Eclipse edition 6.5 and CUDA tookit 6.5 are used for design implementation, cross compilation, remote debugging and performance profiling with a host Ubuntu 12.04 operating system (OS) running on a desktop and a remote Linux For Tegra (L4T) R21.2 OS running on Jetson. A supplementary DPD implementation on a GTX TITAN desktop GPU with 2688 CUDA cores is discussed for comparison. For the experimental verification of DPD, we use a WARP version3 board with MAX2829 transceiver and Anadigics AWL6951 PA as the radio platform and an Agilent E4404B spectrum analyzer to monitor the radio output in real-time.

B. DPD implementation on GPU

1) High-level dataflow: Figure 2 shows the high-level dataflow diagram for the DPD implementation on GPU, and the numbers beside the arrows illustrate the order in which the original input samples propagate between memory and computing cores to finally get the predistorted output samples. As shown in Figure 2, we have three major computation kernels in the trained and finalized DPD design for the run-time predistortion stage: polynomial kernel, filtering kernel and accumulation kernel. Each input sample x_n passes through those kernels to generate the DPD output z_n: polynomial kernel calculates different order polynomials of x_n in the main branch and conjugate branch; filtering kernel computes the polynomial results based on the estimated filter coefficient $\mathbf{h}$; accumulation kernel adds the filtering results of all the p and q as well as the LO leakage estimation c to generate the predistorted sample z_n.

2) Multithreaded kernel execution: In the polynomial kernel, assume we have N modulated I/Q samples $x_0, x_1, x_2, \cdots, x_{N-1}$ as input, for each sample x_n (n=0,1,2,$\cdots$,N-1), our polynomial kernel will generate $R = (P+1)/2 + (Q+1)/2$ polynomials, i.e, $\psi_p(x_n)$ and $\bar{\psi}_q(x_n)$ for all $p \in \{1,3,5,\cdots,P\}$, $q \in \{1,3,5,\cdots,Q\}$. Since the polynomial computation between each sample x_n has no data dependency, we launch our polynomial kernel with N threads to calculate the R polynomials for each of the N samples x_n in parallel. Specifically, we set the block size to 192 threads per block and thus $N/192$ blocks, considering we have 192 CUDA cores on the mobile GPU. The NR polynomials at the output of the polynomial kernel will be streamed to the following filtering kernel. In the filtering kernel, for each sample x_n, we pass its R polynomial results through their corresponding filters, for example, $\psi_p(x_n)$ will pass through

Table I
APH DPD CONFIGURATION

Parameter	Main branch	Conjugate branch
Max polynomial order	P=5	Q=3
Number of filters	3	2
Taps per filter	L_p=5 (for each p)	L_q=5 (for each q)

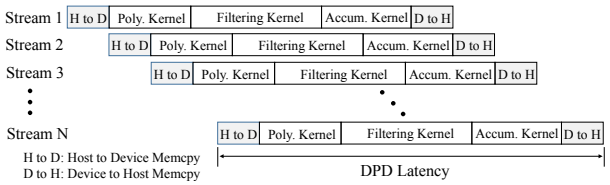


Figure 3. Multi-stream scheduling on desktop GPU

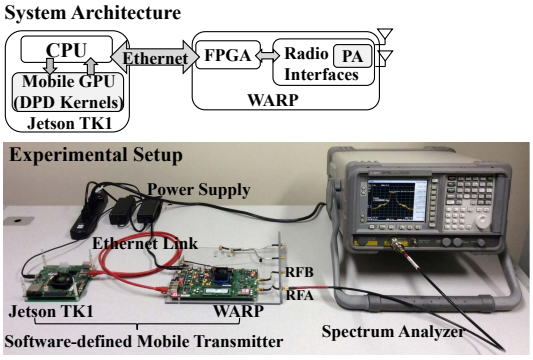


Figure 4. System architecture and experimental setup

$H_p(z)$, which is configured by the estimated filter coefficient $\mathbf{h_p}$. The filtering computation between each of the R filters also has no data dependency stemming from the parallel APH architecture, so in the filtering kernel, we can have even higher NR degree parallelism for calculating the filtering results of each filter for each sample simultaneously. Therefore, the filtering kernel can be launched with $NR/192$ thread blocks with 192 threads in each block and then the NR filtering results will be sent to the next accumulation kernel. In the accumulation kernel, for each sample x_n, we add up its corresponding R filtering results as well as the LO leakage estimation c to generate its corresponding predistorted result z_n. Similarly, we can launch $N/192$ thread blocks with 192 threads in each block for the processing of each sample in parallel.

In above kernels, the value of R is decided by the APH structure, indicating the total number of parallel filters in a DPD, and it should be tuned in the training stage for better DPD suppression effect. Table I shows a typical configuration of the APH DPD architecture, and the configuration is also used for performance profiling in Section VI. The value of N can scale the computational workload and kernel parallelism degree, and usually we set a large N, eg, $10^5 - 10^6$, for our mobile GPU to achieve and maintain high occupancy ratio for the parallel cores, and set an even larger N for a desktop GPU to fully utilize its computational resources. For real-time data streaming, we can provide a batch of N-sample packets and send those packets to the DPD continuously.

3) Memory access optimization: To efficiently access the data for kernel computation, careful utilization of the GPU memory hierarchy is required. Given that the memory copy between CPU and GPU will introduce significant latency overhead, we only copy the DPD input from the CPU at the beginning of the kernel execution and copy back the final predistorted samples back to CPU at the end. We utilize the GPU global memory for the necessary data sharing between the concatenated kernels, such as sharing the polynomial computation results and filtering computation results, as shown in Figure 2. Accessing the GPU global memory will still cost hundreds of GPU clock cycles, so we carefully assign the limited register resources to store the frequently used temporary local variables for fast numerical computation inside the multithreaded kernels. We also take advantage of some special GPU memory modules, such as constant memory, to store the estimated filter coefficients for efficient broadcasting during filtering computation.

A GPU provides powerful floating-point (FP) computation capability, so in our implementation, the DPD input, predistorted output, as well as intermediate results shared among kernels, are all represented by a 32 bit FP data type. In addition, a GPU can group the global memory requests within a 32-threaded $warp$ to a single memory transaction for memory access overhead reduction [16], so alignment of those FP data in the GPU global memory is necessary for coalesced data access to achieve better performance. Here, the N input samples of DPD are stored with consecutive addresses in the global memory and thus originally aligned. In the polynomial kernel, there will be R polynomial results for each sample x_n. To write the total NR polynomial results to global memory in a coalesced way, we need R writing iterations inside the N-threaded polynomial kernel. In a certain r^{th} (r=1,2,$\cdots$,R) iteration, we pack the r^{th} polynomial result of each x_n (we generate the R polynomial results of x_n in the order of $\psi_1(x_n), \psi_3(x_n), \cdots, \psi_P(x_n), \bar{\psi}_1(x_n), \bar{\psi}_3(x_n), \cdots, \bar{\psi}_Q(x_n)$) and write them to global memory by N threads in parallel. In this way, those N results will occupy consecutive addresses within an N-sample length memory segment. In the later accumulation kernel, we also read the filtering results in a coalesced way with similar data alignment. Such coalesced memory access is shown to significantly reduce the memory transaction overhead and improve the throughput.

4) Multi-stream scheduling: The discussed memory optimization strategies are able to enhance the throughput and timing performance on both mobile GPU and desktop GPU. In this part, we illustrate an advanced optimization strategy, multi-stream scheduling, for further performance optimization. Note that such a strategy is only realized in our desktop GPU-based DPD reference design, since the current mobile GPU driver on the Jetson TK1 board does not fully support the concurrent kernel execution mode. Figure 3 illustrates the multi-stream scheduling strategy.

The major goal of multi-stream scheduling is to overlap the memory copy latency between CPU and GPU with the kernel execution latency on GPU. To realize this, we first allocate the host CPU memories as page-locked memories, which can support the direct memory access (DMA) upon GPU memory requests without involving the control of the CPU process [16], enabling asynchronous memory copy between host and device. Then we generate multiple streams in GPU, with each stream controlling the asynchronous memory copy and kernel execution on a segment of samples. In different streams, memory copy and kernel execution can occur concurrently and be overlapped. By performing multi-stream scheduling on the desktop GPU, we can achieve 11%-16% reduction of the total latency on predistorting streaming samples according to profiling results.

C. DPD functionality verification

We first introduce a novel mobile GPU driven software-defined mobile terminal platform. Figure 4 shows the system architecture and experimental setup of our mobile platform. We use WARPLab [12], a MATLAB and FPGA based software-defined radio framework, as the starting point and customize it for the prototype implementation of our mobile terminal. In our previous work [9], we were able to implement a high performance GPU-based software-defined basestation based on our customized WARPLab. Here, benefiting from the powerful and flexible mobile GPU integrated on the Jetson board, we develop a software-defined mobile terminal platform with similar system architecture using the Jetson board and WARP radio board.

The key idea of such a system is to implement the digital baseband processing components, such as DPD, on the mobile GPU integrated on Jetson for both high performance and high flexibility, since

Table II
PERFORMANCE COMPARISON

	90nm CMOS TTA [17]	45nm CMOS TTA [17]	Mobile GPU
Poly. throughput	44.6Msample/s	114.3Msample/s	252.2Msample/s
Filter throughput	39.1Msample/s	100.0Msample/s	156.0Msample/s
Sample precision	16-bit float	16-bit float	32-bit float

Table III
LATENCY COMPARISON ON PREDISTORTING 2×10^6 SAMPLES

	Poly. Kernel	Filtering Kernel	Accum. Kernel
Mobile GPU (@852MHz)	785.59μs	1.28ms	747.43μs
Desktop GPU(@876MHz)	39.26μs	76.96μs	42.43μs

conventional FPGA or ASIC accelerated baseband may suffer from limited flexibility and require significant design period. Such platform can serve as a transmitter (TX) or a receiver (RX) when TX or RX baseband processing kernels for a certain wireless standard are implemented on the mobile GPU. Note that we have intensively customized the original WARPLab in C and CUDA to replace the MATLAB environment to improve both the baseband processing performance and Ethernet throughput for data streaming. Please refer to [9] for more customization details.

Here, we use our mobile terminal platform as a mobile transmitter and the finalized DPD module with estimated parameters is implemented on the Jetson mobile GPU. The streaming samples are created from Jetson CPU and then predistorted on the mobile GPU. The predistorted samples are copied back to Jetson CPU and then streamed to WARP board via Ethernet based on socket APIs. Those samples will be passed through the FPGA-based radio control modules and the real PA and radio interfaces on WARP to transmit out. As shown in Figure 4, the final radio output can be monitored by a spectrum analyzer in real-time so that the DPD suppression effect on spurious spectrum components is verified experimentally in this system. To describe the properties of the PA on WARP, we gather the PA input and output data and generate a memoryless PA model:

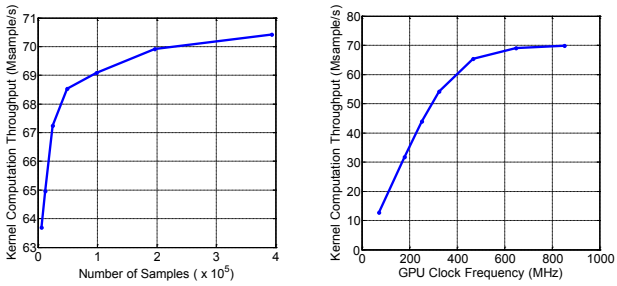
$$PA_{out} = \beta_1 \cdot PA_{in} + \beta_3 |PA_{in}|^2 PA_{in} + \beta_5 |PA_{in}|^4 PA_{in},$$

where $\beta_1 = 0.9490 - 0.0197i$, $\beta_3 = 0.4885 + 0.1071i$, $\beta_5 = -1.0156 - 0.0474i$ are the 1^{st}, 3^{rd}, 5^{th} polynomial coefficients of a 5 order WARP PA model. Note that we also perform the offline training stage experimentally on WARP for the DPD parameter estimation before the real-time DPD experiments. The training feedback loop can be established by connecting RF antenna connector A (RFA, as TX) and RF antenna connector B (RFB, as RX) of WARP so that we can gather the signals in the feedback path based on WARPLab and then do the offline training to finalize the DPD parameters.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

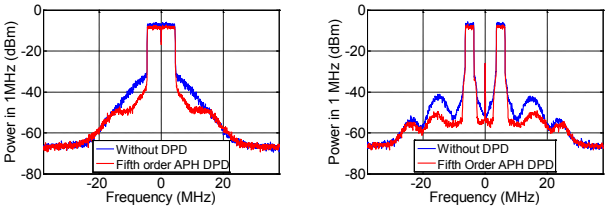
A. GPU performance evaluation

1) Throughput: We profile the kernel computation throughput at different numbers of DPD input samples, i.e., different configuration of N, on the mobile GPU, and show the results in Figure 5(a). With the increase in the number of samples, more parallel threads will be generated to maintain a higher occupancy ratio of GPU cores and finally drive the mobile GPU to achieve its peak performance. Over 70 Msample/s peak kernel computation throughput for the whole predistortion stage can be achieved on the mobile GPU after a large $N \approx 2 \times 10^6$. It is worth mentioning that the mobile GPU clock frequency also has significant effect on the throughput performance and GPU power consumption and can actually be modified on Jetson [18]. In Figure 5(a), the results are measured under the default GPU clock rate of Jetson, i.e., 852MHz. In Figure 5(b), we show the kernel throughput (at a large enough $N \approx 2 \times 10^6$) under different configurations of the mobile GPU clock frequency. There is also a performance upper bound when increasing the frequency, which is due to the overhead of GPU thread deployment and scheduling on computing cores and the limitation of memory resources and bandwidth. In



(a) Different N (b) Different clk. freq.

Figure 5. Throughput performance



(a) Single carrier (b) Non-contiguous CA

Figure 6. DPD effect on spurious spectrum suppression

Table II, we compare the peak performance of our mobile GPU implementation with a previous implementation on Transport Trigger Architecture (TTA) multiprocessor [17], a programmable application-specific multiprocessor. A significant performance improvement is achieved in our mobile GPU based design. We claim that FPGA and ASIC can be other alternatives of DPD accelerators [19] [20], but previous FPGA designs are based on different DPD algorithms, so we omit the performance comparison with those work here for fairness.

2) Latency: In Table III, We profile and compare the kernel computing latency performance between the mobile GPU and desktop GPU. The desktop GPU can significantly outperform the mobile GPU resulting from its more streaming multiprocessors and higher memory bandwidth, which is promising for supporting real-time DPD on a potential wideband software-defined $basestation$ in the 5G context, while our mobile GPU based DPD design can be specifically applied to a high performance software-defined $mobile\ terminal$.

B. DPD effect on a real radio platform

On our Jetson-WARP based software-defined mobile platform, we experimentally verify the DPD suppression effect on spurious spectrum components using various baseband signals. A single 10MHz LTE carrier and two non-contiguous 3MHz LTE carriers with 10MHz spacing are used as input of DPD under single-carrier and non-contiguous CA scenarios, which are shown in Figure 6(a) and Figure 6(b), respectively. Around 10 dB suppression on spurious spectrum components can be experimentally achieved under both scenarios.

V. CONCLUSION

In this paper, we implement a digital predistortion module on a mobile GPU, presenting the feasibility and performance of using mobile GPU for reconfigurable baseband processing in a mobile terminal. By taking advantage of powerful computing features of mobile GPU, our GPU-accelerated DPD design can achieve real-time high performance on predistorting streaming samples. The experimental verification of DPD functionality on our novel Jetson-WARP based software-defined mobile transmitter shows evident suppression effects on spurious spectrum components when transmitting various types of LTE signals through a real radio platform. Benefiting from the programmablility of GPU, our DPD implementation also provides high flexibility on adaptive design reconfiguration and extension.

REFERENCES

[1] P.-I. Mak, S.-P. U, and R. Martins, "Transceiver architecture selection: Review, state-of-the-art survey and case study," *IEEE Circuits and Systems Magazine*, vol. 7, no. 2, pp. 6–25, Second 2007.

[2] E. Dahlman, S. Parkvall, and J. Skold, *4G LTE/LTE-Advanced for Mobile Broadband*, 2011.

[3] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201–220, Feb 2005.

[4] L. Anttila, P. Handel, and M. Valkama, "Joint mitigation of power amplifier and I/Q modulator impairments in broadband direct-conversion transmitters," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, no. 4, pp. 730–739, April 2010.

[5] M. Abdelaziz, L. Anttila, J. Cavallaro, S. Bhattacharyya, A. Mohammadi, F. Ghannouchi, M. Juntti, and M. Valkama, "Low-complexity digital predistortion for reducing power amplifier spurious emissions in spectrally-agile flexible radio," in *9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM)*, June 2014, pp. 323–328.

[6] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[7] G. Wang, M. Wu, B. Yin, and J. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec 2013, pp. 1258–1261.

[8] M. Wu, B. Yin, and J. Cavallaro, "Flexible N-way MIMO detector on GPU," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2012, pp. 318–323.

[9] K. Li, W. Michael, G. Wang, and J. Cavallaro, "A high performance GPU-based software-defined basestation," in *48th IEEE Asilomar Conference on Signals, Systems, and Computers (ASILOMAR)*, 2014.

[10] S. Bang, C. Ahn, Y. Jin, S. Choi, J. Glossner, and S. Ahn, "Implementation of LTE system on an SDR platform using CUDA and UHD," *Analog Integr. Circuits Signal Process.*, vol. 78, no. 3, pp. 599–610, Mar. 2014.

[11] *Nvidia Jetson TK1*. [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[12] *WARP Project*. [Online]. Available: http://warpproject.org/trac/

[13] C. Eun and E. Powers, "A new Volterra predistorter based on the indirect learning architecture," *IEEE Transactions on Signal Processing*, vol. 45, no. 1, pp. 223–227, Jan 1997.

[14] S. Chen and J. Zhao, "The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 36–43, May 2014.

[15] E. Larsson, O. Edfors, F. Tufvesson, and T. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, February 2014.

[16] *Nvidia CUDA tookit documentation*. [Online]. Available: http://docs.nvidia.com/

[17] A. Ghazi, J. Boutellier, M. Abdelaziz, X. Lu, L. Anttila, J. Cavallaro, S. Bhattacharyya, M. Valkama, and M. Juntti, "Low power implementation of digital predistortion filter on a heterogeneous application specific multiprocessor," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 8336–8340.

[18] *Jetson performance tuning*. [Online]. Available: http://elinux.org/Jetson/Performance/

[19] G. Cunha, S. Farsi, B. Nauwelaers, and D. Schreurs, "An FPGA-based digital predistorter for RF power amplifier linearization using cross-memory polynomial model," in *Integrated Nonlinear Microwave and Millimetre-wave Circuits (INMMiC), 2014 International Workshop on*, April 2014, pp. 1–3.

[20] C. Quindroit, N. Naraharisetti, P. Roblin, S. Gheitanchi, V. Mauer, and M. Fitton, "FPGA implementation of orthogonal 2D digital predistortion system for concurrent Dual-Band power amplifiers based on Time-Division multiplexing," *IEEE Transactions on Microwave Theory and Techniques*, vol. 61, no. 12, pp. 4591–4599, Dec 2013.