Leverage Mobile GPUs for Flexible High-Speed Wireless Communication

Qi Zheng Cao Gao Trevor Mudge Ronald Dreslinski
University of Michigan, Ann Arbor
{qizheng,caogao,tnm,rdreslin}@umich.edu

Abstract

Due to the fast emergence of new wireless technologies, traditional hardware accelerator based wireless baseband processors can no longer meet the requirement of a short time to market. On the other hand, general-purpose processors support good general programming models, which can greatly reduce the development cycle and effort for new mobile devices to support the most advanced wireless technologies. Among all the general-purpose platforms, graphics processing units (GPUs) are designed to achieve high throughputs and high energy efficiency by exploring data-level parallelism and thread-level parallelism. In addition, general-purpose programming support has been added to mobile GPUs recently. These make a mobile GPU a promising candidate for the flexible high-speed wireless communication in a user device.

In this work, we evaluate using a mobile GPU for the LTE baseband downlink receiver in a mobile device. We first develop highly parallel CUDA implementations of key signal processing kernels of an LTE downlink receiver. Then we evaluate achieved throughputs and latencies when running an LTE downlink receiver on an NVIDIA Jetson Tegra K1 board. Our results show that by deploying packet batching, we can meet LTE specification peak throughputs for all but one configuration for the physical layer kernels. We also profile the kernels, and point out that the demodulation kernel is the hotspot because of its high computation. Finally, we evaluate the energy efficiency of the mobile GPU under different frequencies, and show that a high frequency achieves better energy efficiency. Compared with other baseband processors, a mobile GPU still has relatively high energy consumption. We provide several implications for the future mobile GPU design to reduce the energy consumption, including having ISA supports for the Turbo decoder and reducing the energy waste from unused registers.

1. Introduction

Mobile applications, such as web browsing, navigation and real-time gaming, have been widely used over the last decade. To support these applications, advanced wireless technologies for high data rates have been deployed. 3GPP Long Term Evolution (LTE) is a standard for the 4^{th} generation (4G) wireless communication, which provides high-speed data for mobile phones and data terminals. The high data rate of LTE makes the wireless module one of the most computational intensive components in a mobile device. In addition, LTE needs to meet the real-time deadline and the quality-of-service (QoS) requirement. Therefore, in order to achieve high throughput and energy efficiency, mobile devices usually deploy hardware accelerators, such as ASICs, DSPs and FPGAs, as the wireless processors. Wireless kernels are hardcoded in an accelerator, and restricted to only a few specific wireless standards or configurations.

However, mobile devices need to support multiple wireless protocols for backward compatibility. Moreover, due to fast development of new wireless technologies, a short time to market of new devices supporting more advanced wireless standards becomes significantly important. Traditional wireless processors with hardware accelerators have fixed functionalities, thus require a long development cycle and tremendous design efforts, making them costly and ineffective. Consequently, we need programmable hardware platforms on which wireless system components can be implemented in software.

The ideal programmable platform for a software-defined wireless system is a general-purpose processor. However, traditional CPUs are designed to improve the latency of a single job, which does not match high throughput requirements of wireless communication. In this paper, we evaluate using mobile graphics processing units (GPUs) as a general-purpose programmable solution for wireless communication in a mobile device. GPUs can achieve high throughputs, and are able to exploit high degrees of data-level parallelism (DLP) and thread-level parallelism (TLP) in a program, with high compute horsepower per Joule [9]. Recently, general-purpose programming support, such as NVIDIA CUDA programming model, has been added to mobile GPUs to enable general-purpose computing on mobile platforms. Therefore, mobile GPUs are promising hardware platforms for flexible software-defined wireless systems where DLP and TLP are abundant.

In this work, we evaluate the effectiveness of using mobile GPUs for an LTE baseband downlink receiver. We first extract important signal processing kernels of LTE baseband from WiBench [18], and develop highly parallel CUDA implementations of these algorithms. We investigate techniques to improve achieved throughputs by batching multiple packets.

PRISM-3 with ISCA'15, June 13-17, 2015, Portland, OR USA

Then we measure the performance of the physical layer kernels and the Turbo decoder respectively, including throughputs and worst case latencies, under different system configurations on an NVIDIA Jetson Tegra K1 board [2]. Our results show that by deploying packet batching, a mobile GPU can meet LTE specification peak throughputs for all but one configuration, when running the physical layer kernels. In addition, we profile all physical layer kernels to identify the performance bottlenecks of mobile GPUs. The results demonstrate that the demodulation kernel is the hotspot in the physical layer, due to its heavy computation. Finally, we evaluate the energy efficiency of a mobile GPU at different frequencies. We find that both the throughput and the power consumption scale linearly with the GPU frequency, and a high frequency achieves higher Mega-bit per Joule. This indicates that we should run a mobile GPU at a high frequency as long as it is in the power envelope.

To the best of our knowledge, this is the first study of running LTE baseband system on mobile GPUs. In this study, we make the following contributions:

1. We develop highly parallel CUDA implementations of key kernels of an LTE baseband downlink receiver.
2. We evaluate a mobile GPU as a programmable platform for a software-defined LTE downlink receiver. We measure the performance and the power consumption of the LTE downlink receiver on a mobile GPU, and study the system performance bottleneck based on the profiling results.

In addition to the contributions mentioned above, we provide implications for future mobile GPU architecture design, indicating that a mobile GPU can benefit from architecture and instruction set support for the Turbo decoder, and reducing the energy waste from the unused GPU registers.

The rest of the paper is structured as follows:

Section 2 introduces the LTE baseband downlink receiver that we implement in this work. The performance, system bottleneck and energy efficiency of running an LTE downlink receiver on a mobile GPU are studied in Section 3. Section 4 shows our suggestions for future mobile GPU designs, and the comparison with other wireless baseband processors. Finally, Section 5 concludes the paper.

2. LTE Baseband Downlink Receiver

Before studying the performance of mobile GPUs, we first introduce the wireless system in a user-end device (UE). In the wireless system of a UE, a baseband unit, which processes baseband signals, does the most computation [12, 18, 19]. A multitude of the baseband unit configurations exist to support different wireless standards. In this work, we study the baseband unit for the LTE standard. LTE is the most widely deployed standard for 4G wireless communication of high-speed data. An LTE baseband unit in a UE contains the uplink transmitter and the downlink receiver. Since the receiver retrieves the data from a noisy communication channel, it must estimate the channel noise level and iteratively decode the information. This means that most of the computations in an LTE baseband unit occur on the receiver side. Therefore, we focus on the baseband downlink receiver.

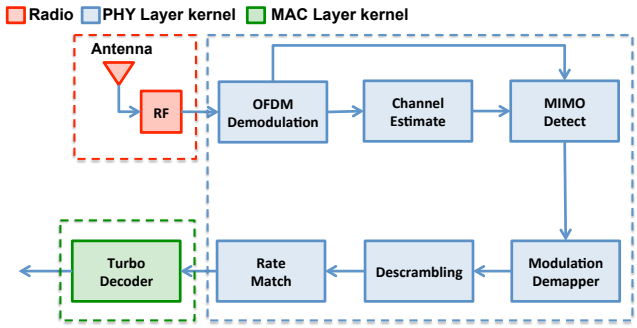

Figure 1: LTE baseband downlink receiver

We extract the most important signal processing kernels of an LTE downlink receiver from WiBench [18]. Figure 1 shows these kernels. The LTE downlink uses the Orthogonal Frequency Division Multiplexing (OFDM) scheme [7]. Data received from the communication channel is converted into digital signals by the RF module. Then the signals pass through the baseband kernels shown in Figure 1 on the physical (PHY) and media access control (MAC) layers, and finally are recovered into binary information bits. Table 1 gives a brief description of each kernel.

We implement these key kernels in CUDA. To maximize the mobile GPU utilization, we deploy several types of parallelism in each kernel. For the PHY layer kernels, we use antenna-level, symbol-level, subcarrier-level and algorithm-level parallelism [17]. For the Turbo decoder, we deploy codeword-level, trellis-level and subblock-level parallelism [16].

3. Running LTE Baseband Downlink Receiver on Mobile GPUs

In this section, we study the performance and energy efficiency of a mobile GPU when running an LTE downlink receiver.

3.1. Methodology

We perform our analyses on an NVIDIA Jetson Tegra K1 SoC platform [2]. It has an NVIDIA Kepler GPU with 192 CUDA cores at the maximum frequency of 852 MHz. The GPU has 256 KB register file and 64 KB L1 memory. The CPU on the Jetson Tegra K1 is a 4-Plus-1 Quad-Core ARM Cortex-A15 R3 processor with the frequency up to 2.3 GHz. The CPU and GPU share a 2 GB DDR3L memory. To profile the performance of the Tegra K1 GPU, we use a variety of tools to measure throughputs, latencies and performance metrics. Table 2 summarizes the tools used in this study.

Because the LTE baseband kernels that we implemented are on both the PHY and the MAC layers, we study the performance of kernels on two layers respectively.

Table 1: Key Kernels in an LTE Baseband Downlink Receiver

Kernel	Description
OFDM Demodulation	OFDM is a method of encoding digital data on multiple carrier frequencies. It is one of the key technologies of LTE. The OFDM demodulation is done through Fast Fourier Transform (FFT).
Channel Estimation	Channel estimation is to obtain the channel condition for reliable communication. LTE uses a pilot-based channel estimation: the transmitter inserts pilot signals at pre-defined positions, and the receiver uses the received signals at the same positions to estimate the channel conditions.
Multiple-Input Multiple-Output (MIMO) Detect	MIMO technology is the use of multiple antennae at both the transmitter and the receiver with the aim of increasing performance or data rate. Data sent by the multiple transmission antennae are combined and received by the receiving antennae, and the MIMO detection recovers the transmitted data by splitting it.
Modulation Demapping	Modulation is to represent a binary data stream with a signal that matches the characteristics of the channel. The binary sequences are mapped into complex-valued symbols in the transmitter. The modulation demapper in the downlink receiver is responsible for retrieving the binary stream from the signal. The most common modulation schemes include quadrature phase-shift keying (QPSK) and sixteen quadrature amplitude modulation (16QAM).
Descrambling	Descrambling recovers data that is scrambled by a wireless base station (BTS). The scrambling in a BTS is to interleave data so that bursts of consecutive bits in error will not contribute to the same symbol.
Rate Match	Rate match is used to provide different channel coding rates from a single "mother code" with a fixed rate, and increases the system flexibility for the performance tradeoff of channel coding. Rate match is performed by puncturing or repeating coded bits.
Turbo Decoder	Turbo coding is the channel coding used in LTE. Channel coding is the technique used to control errors in data transmission over noisy channels that enable reliable delivery of digital data. The Turbo decoder contains two Soft-Input-Soft-Output (SISO) decoders and one internal interleaver, and works in an iterative fashion. In each SISO decoder, a forward and backward trellis algorithm is performed.

Table 2: Profiling Methodology

Tools	Performance Metric
CUDA Event API	Throughput
	Latency
NVIDIA Profiler	Achieved occupancy
	Memory utilization
	Dynamic instruction count

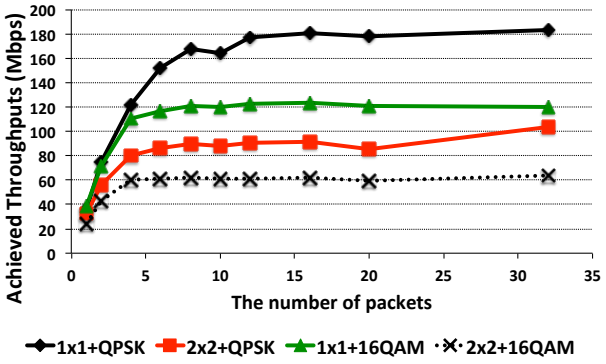

Figure 2: Achieved throughputs of PHY layer kernels with different packet batching sizes

3.2. The PHY Layer on a Mobile GPU

To evaluate the PHY layer, we implement four common LTE configurations: 1 antenna with QPSK (1×1 + QPSK), 1 antenna with 16QAM (1×1 + 16QAM), 2 antennae with QPSK (2×2 + QPSK), and 2 antennae with 16QAM (2×2 + 16QAM).

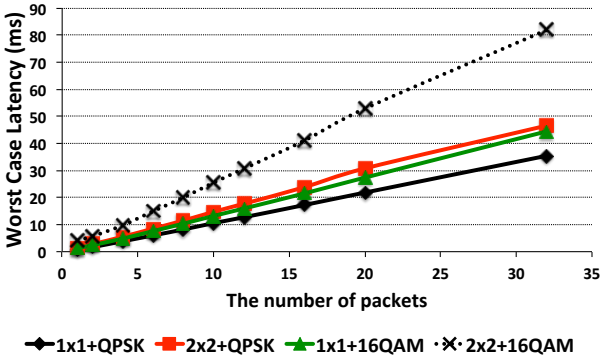

Figure 3: Worst case latencies of PHY layer kernels with different packet batching sizes

3.2.1. Achieved Throughput and Wost Case Latency First, we study the achieved throughputs of the PHY layer kernels. To improve the GPU performance, we try to batch multiple packets, which results in a longer processing latency, because the first packet needs to sit in the memory and wait for other packets. Figure 2 and 3 show the achieved throughputs and the corresponding worst case latencies for different batching sizes. As we can see in Figure 2, the throughputs start saturating at 8 packets for all four configurations. To study the reason of saturation, we measure the occupancy of the GPU, which is shown in Figure 4. The GPU occupancy correlates well with the achieved throughputs, indicating that the full utilization of GPU is the cause of throughput saturation.

However, batching leads to a longer processing latency. In Figure 3, the worst case latency, which is the latency of the first

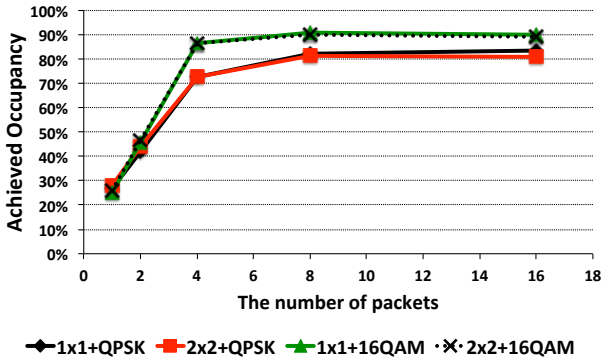

Figure 4: Achieved occupancy of a mobile GPU with different packet batching sizes

packet, is shown. For all four configurations, the latency increases almost linearly with the batching size. This is because that the buffering latency dominates the worst case latency. Based on the LTE quality of service requirement [14], a 30 ms latency in the radio access network (RAN) is required by latency-sensitive services, such as real-time gaming. Because the downlink receiver is only a part of the RAN, its worst case latency must be much smaller than 30 ms. Assuming a 20 ms latency budget in the downlink receiver, we can batch up to 8 packets.

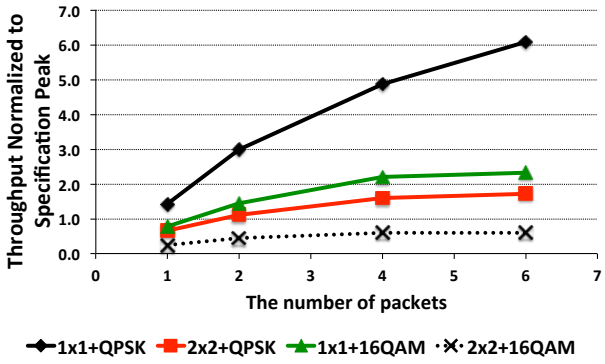

Figure 5: Achieved throughputs normalized to specification peak date rates

3.2.2. Compare with Specification Peak and Typical Data Rate We then compare the achieved throughputs with the specification peak data rates (shown in Figure 5). As we can see, by batching multiple packets, we can fulfill the specification peak data rates for all but one configuration. For 2×2 + 16QAM, we can achieve at most 60% of the specification peak, due to its high computation intensity.

There are usually multiple users in a cellular cell, and they share the available radio resources. This results in a typical data rate per user far lower than the specification peak. We use 10 Mbps as the typical user data rate [15], and compare it with the GPU achieved throughputs (shown in Figure 6). As we can see, all the achieved throughputs are higher than

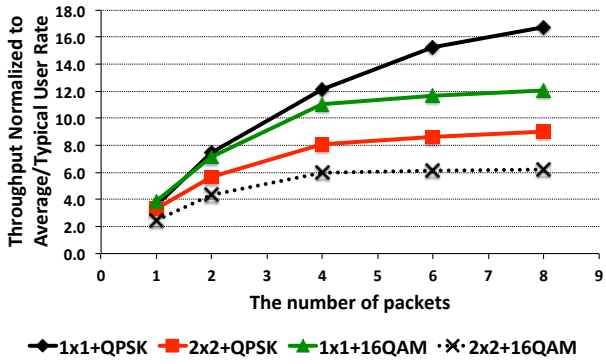

Figure 6: Achieved throughputs normalized to typical user date rates

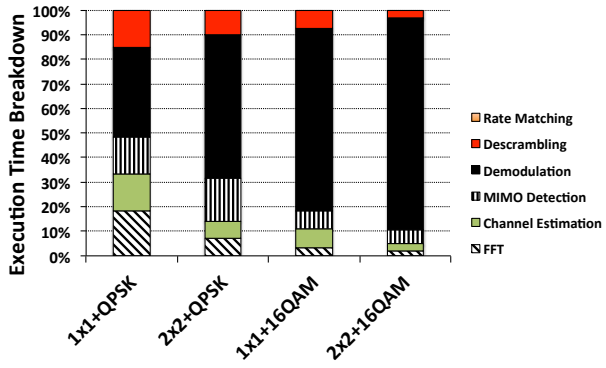

Figure 7: PHY layer kernel execution time breakdown

the typical user data rate, indicating that a mobile GPU can provide enough throughput for an average LTE use case.

3.2.3. Hotspot Analysis We further profile the downlink receiver to identify the hotspot kernel. Figure 7 shows the execution time breakdown among all PHY layer kernels. We can see that *Demodulation* dominates in all four configurations. As more complex modulation and antenna schemes are deployed, *Demodulation* becomes more important. To investigate why *Demodulation* dominates, we profile the executed instruction count of each kernel. Figure 8 presents the normalized instruction counts among all PHY layer kernels. It shows that *Demodulation* executes the most instructions, indicating it is

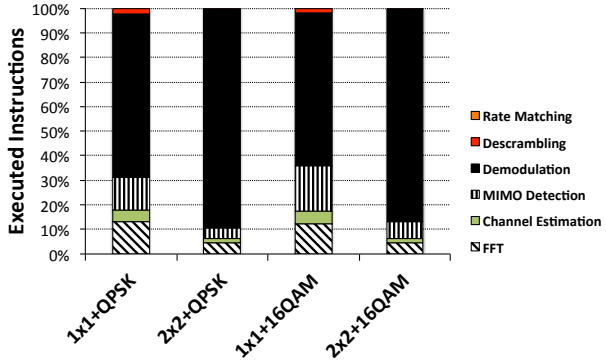

Figure 8: PHY layer kernel instruction count breakdown

the hotspot of the PHY layer due to the heavy computation.

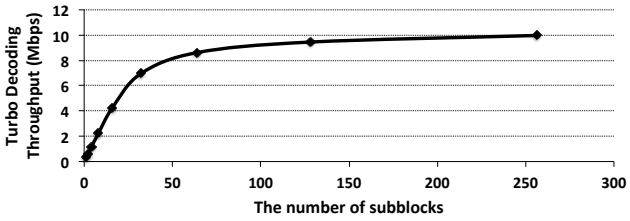

Figure 9: Decoding throughput with varying subblock numbers

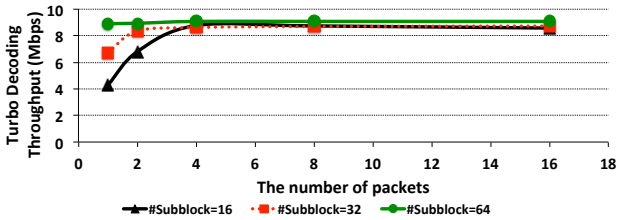

Figure 10: Decoding throughput of packet batching

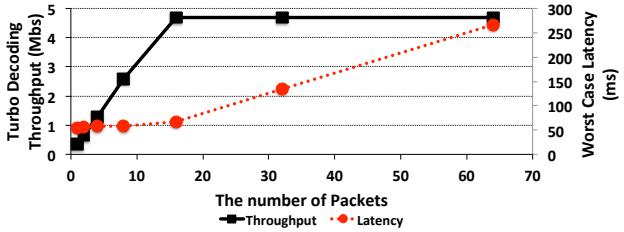

Figure 11: Decoding throughput and worst case latency for varying packet numbers with 1 subblock

3.3. The Turbo Decoder on a Mobile GPU

In this section, we study the throughput and latency of the Turbo decoder. To make full use of the GPU, we explore two different types of parallelism for the Turbo decoder. We first deploy the trellis-level parallelism, in which each state in a Turbo trellis stage is processed independently. This is fixed at eight in LTE, because there are eight states in each stage. Second, we deploy the subblock-level parallelism, where the dependent stages of a codeword are divided into several subblocks and processed in parallel, at the cost of more errors in the output. We combine the trellis-level and subblock-level parallelism, and Figure 9 shows the decoding throughput of this combination with different subblock numbers. We can see the throughput increases with the number of subblocks, ans starts saturating at 64 subblocks. This is because there are enough threads to maximum the GPU computing units.

Then we also try to improve the throughput by batching multiple packets, at the cost of a longer worst case latency. Figure 10 presents the improved throughputs. We can see that batching 4 packets achieves 2×, 1.3× and 1.02× improvement for 16, 32 and 64 subblocks respectively.

We additionally study the throughput and latency of packet batching. Figure 11 shows the results for different packet sizes without subblock-level parallelism. We can see the throughput saturates at 16 packets, and the latency starts increasing linearly at the same packet number. This is because the GPU performance saturates at 16 packets, and the buffering time dominates the worst case latency.

Overall, by batching 2 packets with 32 subblocks, we can achieve 8.4 Mbps throughput with 4.8 ms worst case latency. Though this is lower than any specification peak data rate, it is close to 10 Mbps typical user data rate.

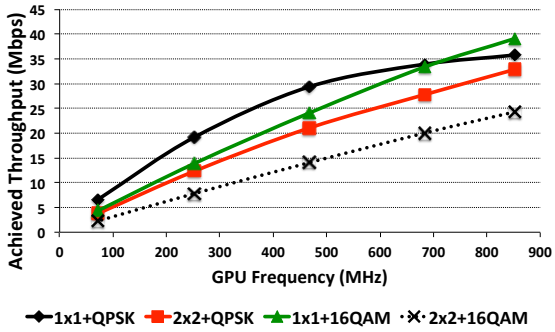

Figure 12: Achieved throughputs of the PHY layer with different frequencies

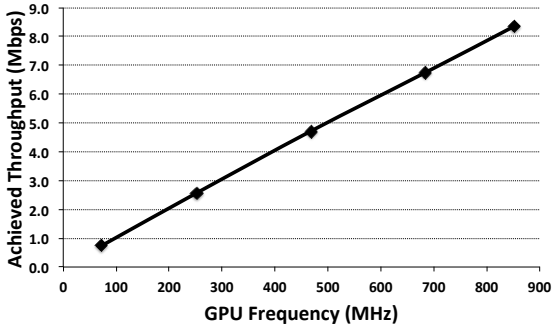

Figure 13: Achieved throughputs of the Turbo decoder with different frequencies

Table 3: Power Consumption of the Mobile GPU SoC

Frequency (MHz)	72	252	468	684	852
Power (W)	0.86	0.99	1.12	1.25	1.39

3.4. Mobile GPU Energy Efficiency

Energy efficiency is a key metric for a mobile hardware platform. In our study, we evaluate the mobile GPU energy efficiency by measuring the amount of data that can be processed

per energy unit. We use a Wattsup.NET [1] meter to measure the power supply of the Jetson board. This gives us the total power consumed by the entire board. We start collecting the power number after 10 warm-up runs for each configuration in order for the power number on the meter to become steady. We subtract the measured number with the power number when the board is idle, and use that as the board dynamic power consumption. Then we estimate the power consumed by the CPU+GPU SoC by deploying the power breakdown from [8], which shows that a CPU+GPU SoC consumes 33% of the power of the entire mobile platform. The Jetson Tegra K1 board supports 15 different GPU frequencies ranging from 72 to 852 MHz. We measure the performance and the power consumption of five different GPU frequencies, including 72 MHz, 252 MHz, 468 MHz, 684 MHz and 852 MHz.

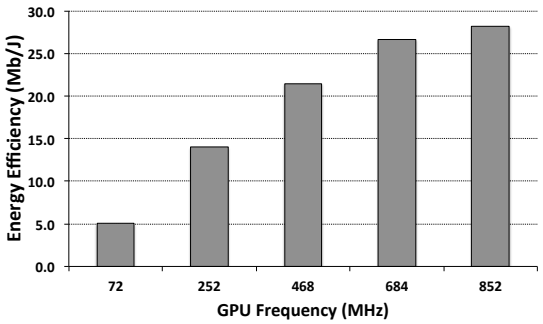

Figure 14: GPU energy efficiency (Mb/J) with different frequencies

Figure 12 and 13 show the achieved throughputs for 1×1 + 16QAM configuration of both the PHY layer and the Turbo decoder with different frequencies[1], and Table 3 presents the corresponding power consumption. As we can see, both the achieved throughputs and the power consumption have almost linear scaling with the frequency. Figure 14 shows the energy efficiency of different frequencies. The metric of energy efficiency being used is Mb/J, which presents the amount of data that can be processed per energy unit. As it shows, a high frequency achieves better energy efficiency. This is because that the achieved throughput increases faster than the power consumption when the frequency scales up.

Overall, the results suggest that for the same LTE configuration, running at the maximum frequency is optimal for a mobile GPU in terms of energy efficiency.

4. Implications for Future Mobile GPU Design

4.1. Reduce energy from the unused register file

When running the PHY layer kernels, we notice that the number of concurrent threads is usually not limited by the hardware computing resource. For example, for most kernels, they are limited by the maximum number of thread blocks and threads per streaming multiprocessor (SM). In this case, several hardware computing resources are underutilized. One good example is the register file (RF). Figure 15 shows the normalized RF utilization for different batching sizes. As we can see, there is hardly any situation that a RF utilization reaches 90%. In addition, when the batching size is small (≤ 2), the RF utilization is below 50%. These unused registers lead to a significant amount of energy waste in a mobile GPU. Therefore, hardware solutions are required to improve the RF utilization or reduce the energy consumed by the unused RF, such as reducing the RF size in a mobile GPU or power gating the unused registers temporarily.

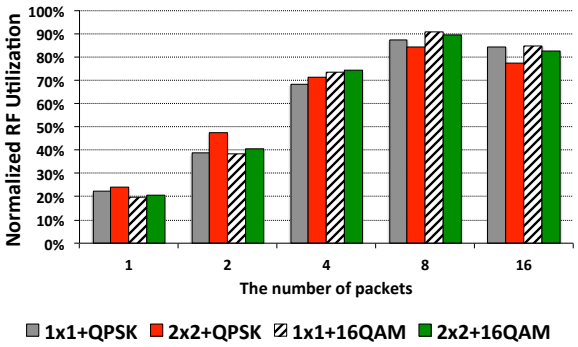

Figure 15: Normalized RF utilization with different packet batching sizes

4.2. ISA extension support for Turbo decoder

One major observation we made in our study is that due to its heavy computation, the Turbo decoder's achievable throughput was much lower than the kernels in the PHY layer. Therefore, there is a need to improve the performance of the Turbo decoder.

The Turbo decoder algorithm is actually in the family of trellis algorithm, a group of algorithms whose processing can be represented as the value propagation in a trellis. Trellis algorithms are widely used in many areas, including coding theory, speech recognition and data compression. Other well-known trellis algorithms include Viterbi algorithm [6] and Baum-Welch algorithm [3]. Because trellis algorithms are widely used, accelerating them on mobile GPUs is a worthwhile research goal to pursue.

There are hardware accelerators with instruction set support in commercial CPUs. One example is the AES accelerator [5] in X86 CPUs. Therefore, mobile GPUs can benefit from having the trellis accelerator and instruction set extension to improve the performance of the Turbo decoder.

4.3. Comparison with other baseband processors

There have been many processor designs for the wireless communication system. Table 4 shows the comparison between mobile GPUs and other baseband processors.

[1] This is the throughput without batching.

Table 4: Comparison between Mobile GPUs and Other Processors for Wireless Communication

Processor	Architecture Type	Supporting Protocol	Technology	Details
LeoCore [13]	ASIC	DVB-T/H, WiMAX	120 nm CMOS	70 mW@ 70 MHz
SODA [12]	DSP	WCDMA	90 nm CMOS	450 mW@400 MHz
MAGALI [4]	FPGA	LTE	65 nm CMOS	234 mW@400 MHz
BP-ASP [20]	DSP	LTE	130 nm CMOS	120 mW@118 MHz
Tomahawk [11]	DSP + ASIC	LTE, WiMAX	130 nm CMOS	940 mW@170 MHz
SoftLTE [10]	General-purpose CPU	LTE	45 nm CMOS	130 W@2.66 GHz
This work	General-purpose GPU	LTE	28 nm CMOS	1.39 W@852 MHz

As we can see, mobile GPUs have much lower power consumption than that of CPUs, but still higher than that of DSPs and ASICs. One reason is that the power consumption we measured is for the entire SoC, including both the CPU and the GPU, but only the GPU does the baseband processing. In addition, the hardware support for general-purpose programming model in mobile GPUs results in a higher power consumption, compared to customized baseband processors. To sum up, the power consumption of mobile GPUs is still in the accepting range for wireless communication in a mobile device, and we expect it to be lowered by the hardware improvements as we suggest above, making it more suitable for wireless communication applications.

5. Conclusion

In this paper, we evaluate using mobile GPUs for flexible wireless communications. Mobile GPUs can exploit the abundant DLP and TLP from the wireless signal processing kernels, and provide high throughputs. With recently added general-purpose programming support, mobile GPUs are promising general-purpose platforms for the future soft-defined wireless communication system.

However, there has been no work that studies using mobile GPUs for wireless communication applications. Therefore, we evaluate the performance of mobile GPUs, including achieved throughputs and latencies, when running an LTE downlink receiver. Our study shows that packet batching is an effective method to improve the performance of mobile GPUs. With packet batching, we can meet the LTE typical user data rate for all four configurations that we implemented, as well as the specification peak data rate for three configurations. Nevertheless, compared with previous baseband processors, including DSPs, FPGAs and ASICs, the mobile GPU still has high energy consumption. Therefore, we propose several suggestions for future mobile GPU designs, including having hardware support for the Turbo decoder and reducing the energy waste from the unused registers.

References

[1] [Online]. Available: http://www.wattsupmeters.com

[2] NVIDIA Tegra K1. [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-K1-whitepaper.pdf

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The annals of mathematical statistics*, pp. 164–171, 1970.

[4] F. Clermidy, R. Lemaire, X. Popon, D. Ktenas, and Y. Thonnart, "An open and reconfigurable platform for 4G Telecommunication: concepts and application," in *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009.* IEEE, 2009, pp. 449–456.

[5] M. C. Demands, "Intel Security Technology for the Cloud," 2012.

[6] G. D. Forney Jr, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[7] "Overview of the 3GPP Long Term Evolution Physical Layer," White Paper, Freescale Semiconductor, Jul. 2007.

[8] F. Hamady, A. Chehab, and A. Kayssi, "Energy consumption breakdown of a modern mobile platform under various workloads," in *2011 International Conference on Energy Aware Computing (ICEAC)*.

[9] S. Huang, S. Xiao, and W. Feng, "On the Energy Efficiency of Graphics Processing Units for Scientific Computing," in *IEEE International Symposium on Parallel Distributed Processing (IPDPS'09)*, 2009, pp. 1–8.

[10] Y. Li, J. Zhang, J. Fang, Q. Cui, K. Tan, and X. Tao, "Soft-LTE: A Software Radio Implementation of 3GPP Long Term Evolution Based on Sora Platform," in *Demo in ACM MobiCom 2009*, 2009.

[11] T. Limberg, M. Winter, M. Bimberg, R. Klemm, M. Tavares, H. Ahlendorf, E. Matúš, G. Fettweis, H. Eisenreich, G. Ellguth *et al.*, "A heterogeneous MPSoC with hardware supported dynamic task scheduling for software defined radio," in *Design Automation Conference*, 2009.

[12] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A low-power architecture for software radio," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2. IEEE Computer Society, 2006, pp. 89–101.

[13] D. Liu, A. Nilsson, E. Tell, D. Wu, and J. Eilert, "Bridging dream and reality: programmable baseband processors for software-defined radio," *IEEE Communications Magazine*, vol. 47, no. 9, pp. 134–140, 2009.

[14] "Quality of Service in LTE," Sandvine, 2014.

[15] "The Verizon Wireless 4G LTE Network: Transforming Business with Next-Generation Technology," White Paper, Verizon, 2012.

[16] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "Parallelization Techniques for Implementing Trellis Algorithms on Graphics Processors," in *The IEEE International Symposium on Circuits and Systems (ISCAS '13)*, 2013.

[17] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "Architecting an LTE base station with graphics processing units," in *2013 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2013, pp. 219–224.

[18] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "WiBench: An open source kernel suite for benchmarking wireless systems," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 123–132.

[19] Q. Zheng, Y. Chen, H. Lee, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "Using Graphics Processing Units in an LTE Base Station," *Journal of Signal Processing Systems*, vol. 78, no. 1, pp. 35–47, 2015.

[20] Z. Ziyuan, T. Shan, S. Yongtao, H. Juan, S. Gang, and S. Jinglin, "A 100 GOPS ASP Based Baseband Processor for Wireless Communication," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 121–124.