J Sign Process Syst (2015) 78:35–47
DOI 10.1007/s11265-014-0932-x

Using Graphics Processing Units in an LTE Base Station

Qi Zheng · Yajing Chen · Hyunseok Lee · Ronald Dreslinski · Chaitali Chakrabarti · Achilleas Anastasopoulos · Scott Mahlke · Trevor Mudge

Received: 28 February 2014 / Revised: 11 July 2014 / Accepted: 15 July 2014 / Published online: 9 August 2014

Abstract Base stations have been built from ASICs, DSP processors, or FPGAs. This paper studies the feasibility of building wireless base stations from commercial graphics processing units (GPUs). GPUs are attractive because they are widely used massively parallel devices that can be programmed in a high level language. Base station workloads are highly parallel, making GPUs a potential candidate for a cost effective programmable solution. In this work, we develop parallel implementations of key kernels to evaluate the merits of using GPUs as the baseband signal processor. We also study the mapping method of key kernels onto a multi-GPU system to minimize the number of required GPUs and the overall subframe processing latency. Our results show that the baseband subsystem in an LTE base station, which supports $\leq$ 150 Mbps peak data rate, can be built with up to four NVIDIA GTX 680 GPUs and commercial motherboards. We also show that the digital processing subsystem for a 75 Mbps LTE base station can be built using two NVIDIA GTX 680 GPUs with power consumption of 188 W.

Q. Zheng (✉) · Y. Chen · R. Dreslinski · A. Anastasopoulos · S. Mahlke and T. Mudge
EECS Department, University of Michigan, Ann Arbor, MI, USA
e-mail: qizheng@umich.edu

H. Lee
Department of Electronics and Communications Engineering, Kwangwoon University, Seoul, Korea
e-mail: hyunseok@kw.ac.kr

C. Chakrabarti
School of ECEE, Arizona State University, Tempe, AZ, USA
e-mail: chaitali@asu.edu

Keywords LTE base station · Baseband processing · Graphics processing unit · Multi-GPU system · Power efficient

1 Introduction

Over the last decade more and more people have been using mobile devices to connect anywhere anytime. Applications supported by these devices, such as web browsing and real-time gaming, require high data rates. To address these needs, third (3G) and fourth (4G) generation wireless technologies have been deployed.

3GPP Long Term Evolution (LTE) is a standard for 4G wireless communication of high-speed data for mobile phones and data terminals. LTE is designed to increase the cell capacity and provide high data rate and is expected to support up to four times the data and voice capacity supported by HSPA [3]. LTE can achieve a peak data rate of 75 Mbps for uplink and 150 Mbps for downlink. In multiple antenna configurations, the peak data rate for downlink can be as high as 300 Mbps.

A wireless base station is responsible for coordinating the traffic and signaling between mobile devices and the network switching system, making them an integral part of the cellular network. Baseband processing requires giga-operations-per-second level throughput [6], making it one of the most computationally intensive components of a base station. Further complicating baseband processor design is the requirement that they must also support multiple wireless communication protocols. This makes the cost of a fixed application-specific integrated circuit (ASIC) solution more costly and drives the need for a programmable solution.

To support easy migration to newer and updated standards, a base station should be built with programmable processors that provide high throughput and low power. While some commercial DSPs [6, 8, 15] provide a good tradeoff between throughput and power consumption, they have to be integrated with accelerators, often designed by different companies, to implement a baseband system. Alternatively, a general-purpose processor has been shown to be a viable solution for CDMA with full programmability and good portability [2]. Unfortunately, this solution is costly because of the large number of Intel Xeon processors that are required.

In this paper we explore building an LTE base station with graphics processing units (GPUs). These processors provide GFLOPS-level throughput, and have high compute capability per Joule [16]. GPUs also have added language support like CUDA for general purpose programming. They provide programmers the ability to exploit high degrees of data-level parallelism (DLP) and thread-level parallelism (TLP). Thus, GPUs are ideal architectural platforms for LTE baseband processing where DLP and TLP are abundant. In addition, due to their high raw compute power per dollar, GPUs are very cost-efficient solutions.

In this work, we focus on the uplink receiver in the LTE base station, because most of the computations are done in the receiver side [24]. We develop highly parallel GPU implementations of all key kernel algorithms in LTE uplink baseband processing, and study their runtime performance under different antenna configurations and modulation schemes when implemented on the NVIDIA GTX680 GPU. We explore a multi-GPU configuration for high data rate applications, and study different mapping methods of key kernels onto a multi-GPU system when considering the LTE throughput constraint and the inter-GPU communication overhead. Our study shows that the baseband subsystem in an LTE base station, which supports up to 150 Mbps peak data rate, can be built with four NVIDIA GTX 680 GPUs and commercial motherboards. Finally, we estimate the power consumption by measuring the dynamic power of each kernel running on a GTX680 GPU. For a 75 Mbps LTE baseband uplink, the digital subsystem of our dual-GPU based LTE base station consumes 188 W, which is competitive with commercial systems.

The rest of the paper is organized as follows. Section 2 introduces the background information of baseband processing in an LTE base station. The GPU implementations of key kernels are described in Section 3. Section 4 introduces different multi-GPU systems, and the mapping methods of key kernels onto a multi-GPU system. Section 5 shows the kernel runtime performance, the minimum number of needed GPUs under different system configurations, and the power consumption. Related work is discussed in Section 6 and the paper is concluded in Section 7.

2 Baseband Processing in an LTE Base Station

The main baseband processing kernels in an LTE base station receiver are shown in Fig. 1. LTE uplink uses Single Carrier Frequency Division Multiple Access (SC-FDMA) for transmission [4]. The total number of subcarriers is fixed based on how much radio bandwidth is used. When there is more than one user, the subcarriers are shared, thereby lowering the data rate for each user.

A bird's-eye view of the processing flow is as follows. The received data from the channel is first processed through SC-FDMA FFT. Pilot signals are used to estimate the channel state information (CSI), which is then used in the MIMO detector to counteract the effects of the channel. The transform decoder performs IFFT on the equalized data. The modulation demapper retrieves bits by generating soft information, and the descrambling reorders soft information based on a predefined pattern. The rate matcher punctures soft information into a predefined length, and finally the Turbo decoder recovers binary information bits. A brief description of the key kernels is given below.

SC-FDMA SC-FDMA is a precoded Orthogonal Frequency Division Multiplexing (OFDM) scheme, which has an additional transform decoding step after conventional OFDM processing. In the LTE uplink receiver, the OFDM step is done using FFT, and the transform decoding step is done using a mixed radix IFFT. The largest size of OFDM FFT is 2048, and that of transform decoding IFFT is 1200.

Channel Estimation The LTE uplink transmission uses the comb-type pilot arrangement. Channel estimation takes the received signal and known pilot reference symbols to estimate the CSI, and then computes the channel coefficients. We implemented frequency domain least square channel estimation [14].

MIMO Detector MIMO technology is the use of multiple antennae at both the transmitter and the receiver with the aim of increasing performance and/or data rate. There are various MIMO detection methods, such as equalization-based detection, sphere decoding and lattice reduction detection. For LTE uplink, an equalization-based MIMO detector, such as zero-forcing (ZF) and minimum mean-square error (MMSE) equalizer, is usually used [17]. We used the MMSE-based MIMO detector in our GPU implementation.

Modulation Demapper The goal of the modulation mapper is to represent a binary data stream with a signal that matches the characteristics of the channel [20]. The binary sequences are grouped and mapped into complex-valued constellation symbols. The modulation demapper, on the

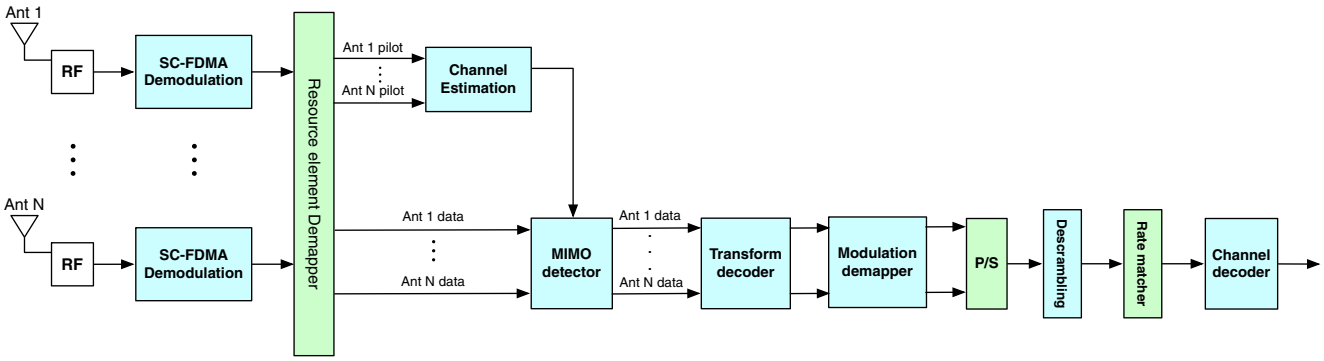


Figure 1 Baseband processing of the receiver in an LTE base station.

other hand, retrieves the binary stream from the signal by generating either hard or soft information. LTE uplink supports four different schemes: BPSK, QPSK, 16QAM and 64QAM. We implemented a soft decision modulation demapper.

Turbo Decoder Turbo codes are used for channel coding in LTE. The Turbo decoder architecture includes two Soft-Input-Soft-Output (SISO) decoders and one internal interleaver/deinterleaver. Inside each SISO decoder, a forward and backward trellis traversal algorithm is performed. The Turbo decoder works in an iterative fashion. For our GPU implementation, we set the number of iterations to be 5 [21].

3 Implementation of Key Kernels on GPU

The baseband of LTE uplink consists of kernels in the physical layer and the Turbo decoder. We will explain the parallelization schemes of these two parts separately.

3.1 Implementation of Kernels in the Physical Layer

A state-of-the-art GPU, such as NVIDIA GTX680, can launch thousands of threads at the same time. So an efficient implementation of kernels on a GPU involves exploiting parallelism at all levels so that enough number of threads are created to keep GPUs busy. There are several types of parallelism in the physical layer kernels: user-level, antenna-level, symbol-level, subcarrier-level and algorithm-level. The different types of parallelism are orthogonal to each other, and can be used at the same time to achieve a better GPU utilization.

User-Level Parallelism A base station serves several users simultaneously, and the signal processing that is done for each user's data is independent from others. There are some initial joint steps required that cannot be parallelized. After the initial steps, a kernel can process data from different users at the same time. The number of generated threads is the same as the number of users.

Antenna-Level Parallelism Data received by the different antennae in the uplink receiver can be processed simultaneously until they reach the transform decoder. Therefore, in these instances, the number of threads is equal to the number of receiver antenna.

Symbol-Level Parallelism The operations of a kernel for a subframe can be parallelized by processing SC-FDMA symbols in a subframe at the same time. The number of threads is as many as SC-FDMA symbols in a subframe.

Subcarrier-Level Parallelism We assume the subcarriers are evenly distributed among all users. If each subcarrier in an SC-FDMA symbol of each user is independent, then they can be calculated in parallel. The number of threads is the same as the number of independent subcarriers.

Algorithm-Level Parallelism There is parallelism inherent inside each algorithm, and it varies based on the kernel. For example in FFT, the operations in the nodes of each butterfly stage can be done in parallel.

In order to quantify the parallelism in each physical layer kernel, we define the following parameters:

- N_{FFT} – FFT/IFFT size
- $N_{Tx} \times N_{Rx}$ – antenna configuration
- N_{Mod} – number of points in a modulation constellation
- N_{sub} – number of subcarriers in a symbol per user
- N_{sym} – number of symbols in a subframe
- N_{usr} – number of users

SC-FDMA The primary operations in SC-FDMA are FFT and IFFT. To map FFT and IFFT efficiently onto a GPU, we employed user-level, antenna-level, symbol-level and algorithm-level parallelism. In FFT/IFFT, in each stage, the

butterfly nodes can be processed independently. So the number of threads created from algorithm-level parallelism is the same as the FFT/IFFT size. The total number of threads that can be generated for FFT/IFFT is $N_{usr} \times N_{Rx} \times N_{sym} \times N_{FFT}$.

In this study, we used cuFFT for the GPU implementation of FFT/IFFT. CuFFT is a CUDA library provided by NVIDIA for computing FFT/IFFT with the input sizes in the form of $2^a \times 3^b \times 5^c \times 7^d$ [13]. We can employ all four levels of parallelism by using cuFFT: the FFT/IFFT implementation of cuFFT exploits the algorithm-level parallelism, and we make use of the other types of parallelism by batching multiple FFT/IFFT computations.

Channel Estimation We implemented a least square based frequency domain channel estimation unit. User-level, antenna-level and subcarrier-level parallelism are considered. The total number of threads that can be generated is $N_{usr} \times N_{Rx} \times N_{sub}$.

MIMO Detector We mapped an MMSE-based MIMO detector on the GPU. We considered user-level, symbol-level and subcarrier-level parallelism. The total number of threads that can be generated for MIMO detector is $N_{usr} \times N_{sym} \times N_{sub}$.

Modulation Demapper Modulation demapping of a subcarrier value consists of two parts: metric calculation and likelihood ratio computing. For metric calculation, we computed the Euclidean distances between the subcarrier value and all complex values in the mapping constellation as the metrics. Algorithm-level parallelism results in as many threads as points in the constellation mapping for a subcarrier. For the logarithm likelihood ratio part, the number of threads is the same as the number of bits in a bit sequence. For example, QPSK groups two bits in a bit sequence and maps the sequence to a single value in the constellation; thus two threads are created for each subcarrier in this case. For metric calculation and likelihood ratio computing, the total number of threads that can be generated is $N_{usr} \times N_{sym} \times N_{sub} \times N_{Rx} \times N_{Mod}$, and $N_{usr} \times N_{sym} \times N_{sub} \times N_{Rx} \times log_2(N_{Mod})$, respectively.

In our GPU implementation of modulation demapper, we process the two stages using two sequential GPU kernel functions respectively. Although having two GPU kernel functions leads to higher function launching overhead compared with having only one, it results in more efficient implementation because it allows each stage to choose the level of parallelization independently. With only one kernel function, the launching overhead is low; however, the number of threads is constrained by the stage with the least amount of parallelism.

Table 1 summarizes the number of threads that can be created for each kernel. In our implementation using NVIDIA GTX680 GPU, we were able to launch all the threads, and our GPU utilization was very high.

3.2 Turbo Decoder

For the Turbo decoder, we consider algorithm-level parallelism in the form of codeword-level, subblock-level and trellis-level parallelism [23]. We include a brief introduction here, a more detailed description can be found in [23].

Codeword-Level Parallelism The codewords can be stored in a buffer so that they can be decoded in parallel. Using codeword-level parallelism results in long latency especially for the first codeword in the buffer.

Subblock-Level Parallelism A codeword can be divided into several subblocks, which are processed in parallel. This leads to a higher bit error rate, because the computations in the subblocks are not independent of each other. The performance loss can be compensated by employing recovery algorithms. There are two widely used algorithms: training sequence (TS) and next iteration initialization (NII). From an implementation perspective, TS requires additional operations, and NII needs additional memory.

Trellis-Level Parallelism In the trellis of Turbo code, we can parallelize the Turbo decoder by exploring state-level parallelism, forward-backward traversal (FB) and branch-metric parallelism (BM). State-level parallelism, in which the nodes in a stage are processed in parallel, does not affect BER. FB leads to more complex index and memory address computations, thereby lowering the throughput. BM is not as effective since the vector reduction parts cannot be parallelized.

In summary, trellis-level parallelism improves throughput without impairing latency and BER. Codeword-level and subblock-level parallelism improve throughput at the cost of either longer latency or higher BER. Both recovery schemes degrade the throughput but improve BER performance compared to only subblock-level parallelism.

Table 1 Number of threads in PHY layer kernels.

Kernel	Number of Threads
FFT/IFFT	$N_{usr} \times N_{Rx} \times N_{sym} \times N_{FFT}$
Channel estimation	$N_{usr} \times N_{Rx} \times N_{sub}$
MIMO detector	$N_{usr} \times N_{sym} \times N_{sub}$
Modulation demapper	$N_{usr} \times N_{sym} \times N_{sub} \times N_{Rx} \times N_{Mod}$ $N_{usr} \times N_{sym} \times N_{sub} \times N_{Rx} \times log_2(N_{Mod})$

We also explore how to use GPU memory effectively for Turbo decoder. The GPU memory system consists of on-chip memory, off-chip L2 cache and external DRAM. On-chip memory can be configured into $48KB/16KB$ or $16KB/48KB$ shared memory/L1 cache. Shared memory is software managed, and the benefit of using it is that any access to shared memory is always in the fast on-chip memory. When the number of threads is fixed, more shared memory usage per thread leads to faster memory access. However, this results in larger shared memory and smaller L1 cache, and thus increases the L1 cache miss rate. In our Turbo decoder implementation, we use shared memory to store the intermediate metrics. By varying the amount of intermediate metrics stored in shared memory, we can change the shared memory usage per thread. In order to maximize the GPU performance, we test implementations with different shared memory usages and different on-chip memory configurations. The corresponding results are given in Section 5.2.

4 LTE Base Station Baseband on a Multi-GPU System

In order to support high peak data rates of the LTE uplink, we may need more than one GPU to build the baseband subsystem in a base station. Therefore, it is important to explore how to map kernels onto multiple GPUs, considering the communication overhead of different multi-GPU systems.

4.1 Multi-GPU System

Employing multiple GPUs in an LTE base station can further speedup computation to help kernels meet the throughput constraint of an LTE subframe, when a high peek data rate is required. Inter-GPU communication overhead is a key concern of a multi-GPU system when there is data movement between GPUs. Multi-GPU systems can be classified into two types based on the inter-GPU connection: multiple GPUs within a single network node (MGSN), and multiple GPUs across multiple network nodes (MGMN).

GPUs within a Node In an MGSN systems, GPUs sit in the same network node and they communicate to each other through fast point-to-point interconnects on board. Figure 2 shows two most commonly used on-board interconnects on a commercial motherboard: connected through the PCI Express (PCI-E) switch, and through the I/O hub (IOH) chip. The achievable throughput of inter-GPU data movement is different between these two connections. When GPUs are connected through the PCI-E switch (shown in Fig. 2a), they can communicate through direct peer-to-peer (P2P) memory copies, which leads to a high communication throughput. When connected through IOH chips (shown in Fig. 2b), GPUs attached to the same IOH chip can still use direct P2P communication, achieving a high throughput. However, GPUs attached to different IOH chips cannot. This is because the GPUs connected through different IOH chips are not coherent with each other [19]. Therefore, the data transfer between GPUs attached to different IOH chips is staged via CPU memory, which lowers the communication throughput.

GPUs Across Multiple Nodes Due to the power supply and heat dissipation constraint, a motherboard can only support a limited number of GPUs. Commercial motherboards today support up to four GPUs for general-purpose computing [9]. When more GPUs are required, we have to use multiple network nodes, in which each node is an MGSN system. In an MGMN system (shown in Fig. 3), GPUs in the same node transfer data in the same way as an MGSN system. However, when GPUs in different nodes try to communicate, the data has to be staged via CPU memory and inter-node connection that is usually Ethernet. This increases the communication latency between GPUs, and makes it difficult to fulfill the real-time deadline of an LTE subframe.

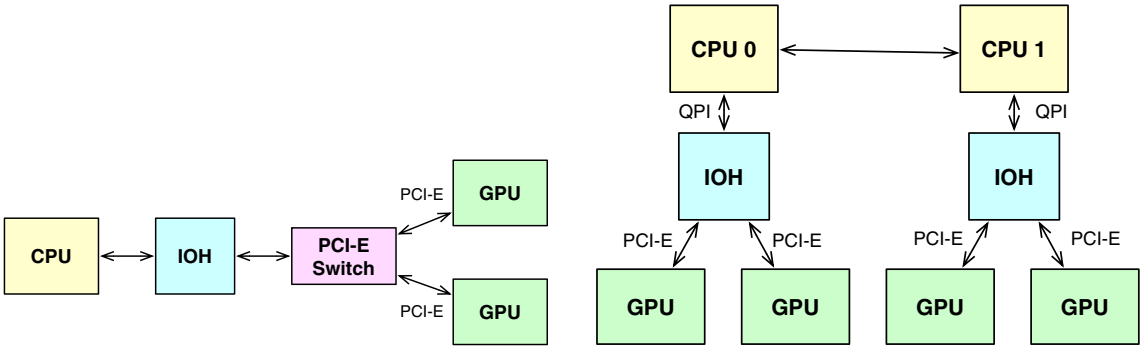
We use the inter-GPU and CPU-GPU communication throughputs from [19] in this study. Table 2 summarizes the throughput numbers. These numbers are representative of a multi-GPU system; however, these numbers may vary for different BIOS settings and IOH chips. For the inter-node connection, we simplify the communication overhead calculation by only taking the transfer latency on Ethernet cable into account. Since the data transfer between GPUs on different nodes is staged through several steps and has a fairly long latency, this simplification is reasonable and will not change the conclusion of our study.

4.2 Mapping Kernels on a Multi-GPU System

Figure 4 shows two different ways of mapping the LTE baseband kernels onto a multi-GPU system: sequential and pipelined.

Sequential Mapping Figure 4a shows an example of sequential mapping, in which kernels 1 and 2 are executed in sequence on the same GPU. All GPUs process kernel 1 in time slot $[t_0, t_1]$, and kernel 2 in time slot $[t_1, t_2]$. The overall execution time $T(= t_2 - t_0)$ must be shorter than that specified by the throughput constraint of the LTE system.

Pipelined Mapping The other way to process kernels 1 and 2 is pipelined mapping, which is shown in Fig. 4b. During time slot $[t'_0, t'_1]$, GPU-0 processes kernel 1 of packet k, while GPU-1 and GPU-2 process kernel 2 of packet $k-1$.



(a) Connected through PCI-E switch (b) Connected through IOH chips

Figure 2 Multiple GPUs within a single network node (MGSN).

Since kernel 2 takes longer time to run, its execution also overlaps with the inter-kernel data transfer from GPU-0 to GPU-1 and GPU-2. Because the processing of a packet is pipelined into multiple stages, only the runtime of each stage is required to be shorter than that specified by the throughput constraint.

Every mapping method has its advantages and disadvantages. The sequential mapping has a short processing latency for each packet, and usually no additional inter-GPU communication overhead. However, it requires more GPUs to accelerate the processing so that all kernels can be computed within the timing deadline. The pipelined method, on the other hand, needs fewer GPUs but longer latency. This is because the total processing time of a packet is the sum of runtimes of all the pipeline stages. The two mapping methods can be combined to get a better tradeoff between the number of GPUs and the overall packet processing latency. For instance, kernels can be combined into several groups in which kernels run sequentially on the same GPUs, and the groups can be pipelined on different sets of GPUs.

In order to help decide the best mapping method of the LTE baseband kernels onto a multi-GPU system, we assume that a mix of sequential and pipelined mapping methods is employed. The overall processing is pipelined into several stages. In each stage, some kernels are processed on the same GPUs sequentially. Let S be the number of pipelined stages, K be the number of kernels, N_i be the number of GPUs needed in the ith stage, P_i be the number of kernels running sequentially in the ith stage, $t_{run}(i, j, N_i)$ be the runtime of the kernel j in the ith stage using N_i GPUs, and $t_{comm}(i, i+1)$ be the inter-GPU communication overhead between the ith and $(i+1)$th stage. Our objective is to minimize the number of GPUs and the overall packet processing latency. This can be expressed as follows:

$$\alpha \times (min(\sum_{i=1}^{S} N_i)) + \beta \times (min(\sum_{i=1}^{S}\sum_{j=1}^{P_i} t_{run}(i, j, N_i) + \sum_{i=1}^{S-1} t_{comm}(i, i+1))) \quad (1)$$

where α and β are predetermined constants that reflect the relative importance of minimizing the number of GPUs and minimizing the processing latency.

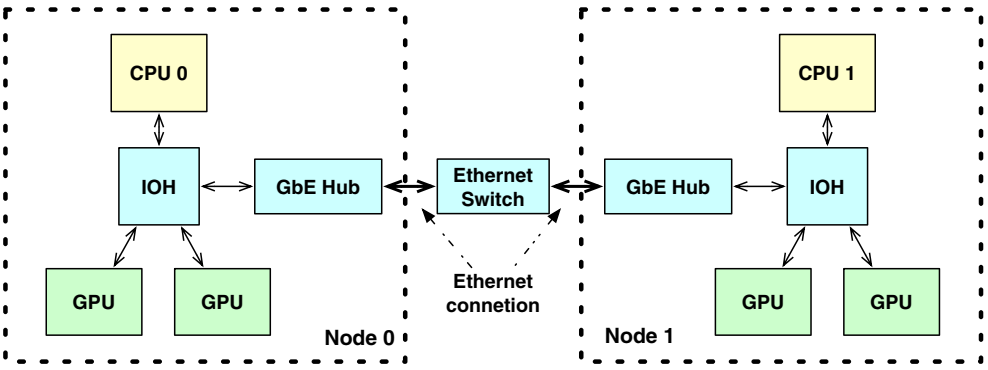


Figure 3 Multiple GPUs across multiple network nodes (MGMN).

Table 2 Inter-GPU and CPU-GPU copy throughputs in a non-uniform memory access system.

Inter-GPU copy	Throughput (GB/s)
Via PCI-E switch	6.3
Via IOH chip (attached to the same IOH chip)	5.3
Via CPU (attached to different IOH chips)	2.2
CPU-GPU copy	
GPU to local CPU	6.3
GPU to remote CPU	4.3
CPU to local GPU	5.7
CPU to remote GPU	4.9

Let $t_{deadline}$ be the timing deadline that is derived from the throughput constraint. Then for the ith stage, the timing constraint is given by:

$$\sum_{j=1}^{P_i} t_{run(i,j)} + t_{comm(i,i+1)} \leq t_{deadline} \quad (2)$$

5 Results

5.1 Experimental Environment

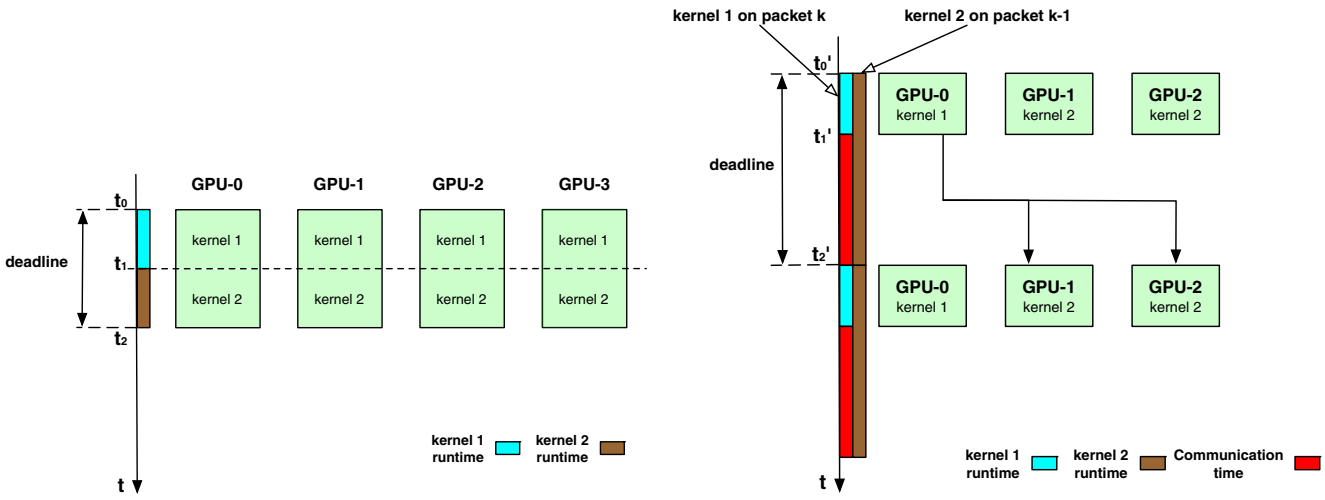
We used an NVIDIA GTX680 GPU to evaluate the performance of the key kernels. GTX680 is based on the Kepler architecture [12]. It has 8 Streaming Multiprocessors (SMX), and in each SMX there are 192 Streaming Processors clocked at 1GHz. A GTX680 GPU can launch at most 1024 threads at a time. There is a 64 KB on-chip memory, a 512 KB L2 cache and 2048 MB external memory. To monitor the GPU, we used GPU-Z, which is a lightweight tool designed to provide information such as the dynamic power consumption, the dynamic GPU load, the fan speed, etc. To launch GPU kernels, we used a 2.13 GHz Intel Core 2 processor, running the Linux 3.2.0-39 generic operating system.

In this study, we simulated a fading channel with additive white Gaussian noise. We evaluated kernel implementation performance corresponding to peak data rate. We also focused on the single-user scenario, because it does not exploit user-level parallelism and gives a worst case estimate of the GPU performance. In this operating scenario, the computations in a base station depend on the total number of available subcarriers.

5.2 Kernel Runtimes

We ran each kernel for the different configurations shown in Table 3. Table 4 shows the runtimes of different physical layer kernels of an LTE subframe. It demonstrates that modulation demapping takes the longest runtime. In addition, MIMO detection, channel estimation and modulation demapping have fairly long runtimes when more antennae or more complex modulation schemes are used.

For the Turbo decoder, we first studied the on-chip memory use. We implemented two GPU memory configurations ($48KB/16KB$ and $16KB/48KB$ shared memory/L1 cache) with state-level, subblock-level and packet-level parallelism [23]. We varied the number of subblocks from 1 to 512, and the number of packets from 1 to 84. We found that a larger L1 cache results in better runtime



(a) Sequential mapping

(b) Pipelined mapping

Figure 4 Mapping kernels onto a multi-GPU system.

Table 3 The configurations of each kernel.

Kernel	Configuration
Turbo decoder	code rate = 1/3, codeword length = 6144, iteration number = 5
Modulation demapper	16QAM and 64QAM
SC-FDMA FFT	2048
Decoding IFFT	1200
MIMO	$1 \times 1, 2 \times 2, 4 \times 4$

performance. For example, for the configuration with small input size (1 packet) and 64 subblocks, the larger L1 cache configuration achieves 30.8 % higher throughput compared with smaller L1 cache configuration. So, we used the 48K L1 cache configuration in the rest of the experiments. To maximize the performance, we assigned as many threads as possible in each CUDA thread block. The number of threads per CUDA thread block is constrained by the size of the shared memory and the shared memory usage per thread, Next we tried combinations of different parallelization schemes to find the best tradeoff among the decoding throughput, worst-case latency and BER [23]. Table 5 shows the performance of the Turbo decoder implementations. It demonstrates that the implementation that includes state-level parallelism and forward-backward traversal (row 3) achieves the best tradeoff. Compared with other implementations, it has the shortest worst-case codeword latency with good BER performance, and the throughput is also quite high. We use this implementation in the rest of the paper.

Our implementation of kernels exploits the parallelism at multiple levels. For instance, in our GPU implementation of a 4×4 64QAM system, there are 14,336 threads created for FFT, 4,800 threads for channel estimation, 14,400 threads for MIMO detection, 3,686,400 threads for modulation demapping metric calculation, 345,600 threads for modulation demapping likelihood ratio computing, and 8,192 threads for the Turbo decoder. Because an NVIDIA GTX680 GPU can launch at most 1,024 threads at a time, it is almost always fully utilized.

Table 4 PHY layer kernel runtimes (ms) of an LTE subframe.

Antenna configuration		1×1	2×2	4×4
FFT		0.06	0.07	0.08
IFFT		0.10	0.10	0.10
MIMO detector		0.02	0.03	0.52
Channel estimation		0.02	0.05	0.46
Modulation demapper	16QAM	0.08	0.15	0.28
	64QAM	0.47	0.92	1.81

Table 5 Performance of Turbo decoder implementations.

Schemes			TH[1]	WPL[1]	BER[1,3]
TL[2]	Subblock Num	CW[1] Num	(Mbps)	(ms)	
SL[2]	512	2	77.64	0.72	1.6×10^{-3}
SL	256	4	78.15	1.68	4.1×10^{-4}
SL,FB[2]	256	2	78.30	0.72	4.1×10^{-4}
SL,FB	128	7	80.58	3.08	2.0×10^{-4}

[1]TH = Throughput, WPL = Worst-case codeword Latency, SNR = Signal-to-Noise Ratio requirement, BER = Bit Error Rate, CW = Codeword

[2]TL = Trellis-level parallelism, SL = State-level parallelism, FB = Forward-Backward traversal

[3]BER here is the bit error rate when SNR = 1.0 dB

5.3 GPU-Based LTE Baseband System

A baseband processor must meet the latency and throughput requirements of the communication protocol. LTE supports multiple data rates up to 300 Mbps. Table 6 describes the kernel configurations for the different data rates. For high data rates, the computational load of LTE baseband processing is very high and a single GPU is not enough. For instance, LTE provides a throughput of 1 subframe/ms. Based on runtimes presented in Table 4, one GPU is not enough to meet this requirement for high data rates when multiple antennae or more complex modulation schemes are used. In such cases, we must employ the mapping method discussed in Section 4.

We use a mixture of sequential and pipelined mapping, in which the overall processing of an LTE subframe (shown in Fig. 1) is pipelined into several stages. In each stage, the processing of some kernels on subcarriers or symbols in a subframe is assigned evenly to several GPUs. Because Turbo decoding is the hotspot of LTE baseband [22, 24] and also not part of the PHY layer, it occupies a stage by itself. We use the formulation in Section 4.2 to explore kernel mappings that minimize both the number of GPUs and the packet processing latency under different system configurations.

Table 6 Kernel configurations for different peak data rates.

Data rate (Mbps)	SC-FDMA FFT	Decoding IFFT	MIMO	Modulation demapper
50	2048	1200	1×1	16QAM
75	2048	1200	1×1	64QAM
100	2048	1200	2×2	16QAM
150	2048	1200	2×2	64QAM
200	2048	1200	4×4	16QAM
300	2048	1200	4×4	64QAM

First, we evaluate the number of GPUs needed to process kernels in the PHY layer. Based on Tables 4 and 6, only one GTX680 GPU is needed to fulfill the 1 subframe/ms requirement when the peak data rate is no higher than 100 Mbps. However, when the peak data rate is higher, more GPUs are required. For the 150 and 200 Mbps peak data rate configurations, we can either process all PHY kernels on the same GPUs sequentially, or pipeline them between decoding IFFT and modulation demapper for 150 Mbps, and between channel estimation and MIMO detection for 200 Mbps. In either case, two GTX680 GPUs are needed. Since the pipeline mapping introduces additional inter-GPU communication overhead, the sequential mapping has a better overall packet latency. Therefore we chose the sequential mapping of PHY layer kernels for the 150 and 200 Mbps peak data rate configurations. For the 300 Mbps configuration, we have three options: 1) have one pipeline stage, and process all PHY kernels sequentially, 2) have two pipeline stages, where stage 1 implements FFT, channel estimation and stage 2 implements MIMO detection, decoding IFFT, modulation demapper, 3) have three pipeline stages, where stage 1 implements FFT, channel estimation, stage 2 implements MIMO detection, decoding IFFT and stage 3 implements modulation demapper. These three options require five, five and four GPUs, respectively. We pick option 3 to minimize the number of required GPUs.

Then we evaluate the Turbo decoding kernel. There is no timing deadline for Turbo decoding, because in our implementation multiple subframes are buffered in the GPU memory before being processed together. But the achievable throughput of the GPU implementation must be higher than the peak data rate of the uplink. Therefore multiple GPUs are still needed for high data rate configurations.

Table 7 summarizes the minimum number of GTX680 GPUs needed for PHY layer processing and Turbo decoding in the uplink receiver of an LTE base station. For the configurations with peak data rate $\leq$ 150 Mbps, up to four GPUs are required, which can fit in an MGSN system. Due to the fast on-board communication, the inter-GPU communication overhead is very small, so these configurations can be built with commercial motherboards. When using more than four GPUs, an MGMN system must be employed. In this case, the data transfer between PHY layer kernels and Turbo decoding requires inter-GPU communication across different network nodes. Because of the relatively slow Ethernet connection, and the multiple stages that the data has to traverse, our calculation shows that the inter-GPU communication latency across nodes is longer than what can be supported by the 1 subframe/ms requirement. This makes it infeasible to build such a system with today's commercial devices.

Based on our analysis above, the inter-GPU communication overhead across network nodes is the key obstacle to enable a GPU-based LTE base station to support an uplink data rate $\geq$ 200 Mbps. We studied what can be done to reduce this overhead: a better GPU, a more powerful motherboard, and faster inter-node connections. A better GPU can result in a smaller number of GPUs being required, and a more powerful motherboard can support more than four GPUs for general-purpose computing. Both of them make it possible to fit the GPUs required for very high uplink data rate on one motherboard. In this case, all inter-GPU communication is on-board and fast, and the communication latency will be smaller than what can be supported by the 1 subframe/ms requirement. A faster inter-node connection is another way to reduce the inter-GPU communication overhead across network nodes. 100 Gbit/s Ethernet has already been prototyped, using it would reduce the data transfer latency between network nodes to the timing deadline specified by the 1 subframe/ms requirement.

Table 7 The minimum number of GTX680 GPUs needed for the baseband system covering a cell.

Data rate	Number of GPUs		
(Mbps)	PHY	Turbo	Total
50	1	1	2
75	1	1	2
100	1	2	3
150	2	2	4
200	2	3	5
300	4	4	8

5.4 Power Consumption

We measured the dynamic power consumption of the LTE kernels using GPU-Z. Table 8 shows the power consumption of each kernel and the corresponding configuration. We also measured the power consumed by each kernel under different configurations, and observed very limited variation. The actual energy consumed by each kernel is presented in Table 9. It shows that the Turbo decoder consumes most of the system energy followed by modulation demapper.

Table 8 Power of each kernel on a GTX680 GPU.

Kernel	Configuration	Power (W)
Turbo decoder	Row 3 in Table 5	63.3
SC-FDMA FFT	2048	56.7
Decoding IFFT	1200	56.9
Modulation demapper	64QAM	56.3
Channel estimation	–	61.8
MIMO detector	4×4	57.7

Table 9 Energy consumption of each kernel processing 1 subframe at 75 Mbps on a GTX680 GPU.

Kernel	Energy (mJ/subframe)
Turbo decoder	144.0
SC-FDMA FFT	3.4
Decoding IFFT	5.7
Modulation demapper	26.5
Channel estimation	1.2
MIMO detector	1.3

For a system-level power assessment, we considered the configuration corresponding to a 75 Mbps data rate. Based on Table 7, we need two GTX680 GPUs for a 75 Mbps data rate, one for the Turbo decoder and the other for the PHY layer. We also need one Intel Core 2 CPU, whose maximum power is 63 W. Thus, the total power of the digital subsystem of the receiver is 188 W. We compared our implementation with the Alcatel-Lucent 9926 Base Band Unit [5] whose maximum power is 370 W with 74 Mbps peak uplink throughput. While 370 W includes both the transmitter and receiver power, the receiver processes more complex kernels, like Turbo decoding and MIMO detection, and consumes a significantly larger portion of the compute power. Even if we conservatively estimate that half of the power, 185 W, is consumed by the receiver, our proposed GPU-based solution is still quite competitive.

6 Related Work

GPU-based Solutions The GPU implementation of the transmitter in an LTE base station was presented in [18]. In contrast, we provided the GPU implementation of the receiver along with a detailed analysis of possible parallelization schemes and their effectiveness.

DSP-based Solutions There are several DSP-based solutions. Freescale's Modular AdvancedMC Platform [8] contains three MSC8156 DSPs for baseband processing. Each DSP has six StarCore SC3850, and a MAPLE-B baseband accelerator for Turbo/Viterbi decoder, FFT/IFFT, and multi-standard CRC check and insertion [7]. CommAgility's AMC-3C87F3 is a signal processing card for 4G wireless baseband. It contains three Texas Instruments' TCI6487 DSPs, each with three C64x+ cores and coprocessors for Viterbi decoder, Turbo decoder and Rake search/spread. Although the DSPs mentioned above are programmable, several key kernels are implemented using accelerators, which impairs the system flexibility. To support new protocols, new accelerators have to be designed and integrated with DSPs, leading to a long development cycle and high cost. In contrast, GPU-based solutions only need new software for the system update, which dramatically reduces time-to-market and cost. Additionally, if GPUs cannot support the high data rate of future protocols, they can be replaced by newer and faster GPUs, provided these newer GPUs support a high level programming paradigm such as CUDA.

FPGA-based Solutions Xilinx [10] and Altera [11] have developed FPGA solutions for baseband processing in LTE base stations. Although FPGAs have good flexibility, their relatively high price increases the cost of using them to build a base station. A GPU-based base station has a fairly short development cycle and little updating effort, because the software is implemented in a relatively simple high-level language.

GPP-based Solutions The Vanu Anywave base station [2] is the only fully programmable commercial base station to date. It is built with 4-13 Intel MPCBL0040 single board computers [1] based on the required cell capacity. An MPCBL0040 computer contains two Dual-Core Intel Xeon E7520 2.0 GHz processors. The Vanu Anywave uses GPPs instead of DSP. Currently it supports GSM/EDGE/CDMA2000 but does not support LTE.

GPPs have good flexibility and portability, but they cannot make full use of the available DLP in a wireless base station. This is why Vanu needs as many as 52 Intel Xeon cores to support CDMA2000, and more are expected in order to support LTE, leading to even higher power consumption. A GPU-based solution takes advantage of massive DLP. So fewer GPUs are needed in an LTE base station, which makes a GPU-based solution power efficient.

7 Conclusion

In this paper, we presented our work on building a baseband system for an LTE base station using commercial GPUs. The kernels in LTE baseband processing are highly parallel, and thus amenable to efficient GPU implementation. We implemented key kernels of an LTE baseband system on the NVIDIA GTX680 GPU, and evaluated the runtime performance. We also explored the kernel

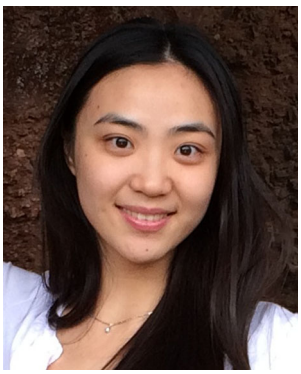
mapping method to minimize the number of required GPUs and the overall subframe processing latency, when using a multi-GPU system. We showed that an LTE base station that supports a 150 Mbps peak uplink data rate can be built by using four GPUs and the commercial motherboard [9]. To support higher uplink data rates, more complex antennae and modulation schemes are needed. In these situations a quad-GPU solution is no longer sufficient. We also showed that the GPU-based solution is power efficient. To support the digital subsystem of a 75 Mbps uplink, a dual-GPU LTE base station consumes 188 W, which is quite competitive with commercial solutions.

Acknowledgments We wish to thank Nilmini Abeyratne and Yuan Lin for their generous help and useful feedback on the paper. This work is supported by the National Science Foundation grant NSF-CNS-0910851 and ARM Ltd.

References

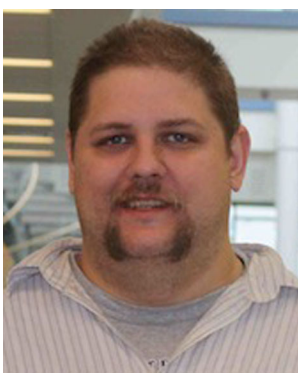
1. (2006). Intel solutions for the next generation multi-radio basestation. Intel application note. http://intel.com/design/intarch/applnots/307450.htm.
2. (2006). The vanu anywave base station subsystem. http://www.vanu.com/documents/technology/vanu-anywave-2006-05.pdf.
3. (2007). Long Term Evolution (LTE). Motorola White Paper. http://www.motorolasolutions.com/web/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6833_MotDoc_New.pdf.
4. (2007). Overview of the 3GPP long term evolution physical layer. Freescale semiconductor white paper. http://www.element14.com/community/servlet/JiveServlet/previewBody/13380-102-1-42319/3GPPEVOLUTIONWP.pdf.
5. (2009). Alcatel-Lucent 9926 digital 2U eNodeB baseband unit. Alcatel-lucent product brief.
6. (2009). LTE emerges as early leader in 4G technologies. White Paper. http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=spry124&;fileType=pdf.
7. (2010a). Accipiter systems 4G (LTE/WiMAX) base transceiver station AMC. Product Brief. http://www.accipitersystems.com/Files/Admin/wexford%20product%20brief%20final_4G_Proprietary_Removed_6_28_10.pdf.
8. (2010b). Freescale modular AdvancedMC platform for broadband/LTE base stations. http://cache.freescale.com/files/32bit/doc/fact_sheet/LTEWIMAXFS.pdf.
9. (2011). Gigabyte GA-X79-UD7. http://www.gigabyte.us/products/product-page.aspx?pid=4047#sp.
10. (2011). LTE baseband targeted design platform. Xilinx product brief. http://www.origin.xilinx.com/publications/prod_mktg/LTE-Baseband-SellSheet.pdf.
11. (2012). Designing basestation channel cards with FPGAs. ALtera product brief. http://www.altera.com/literature/po/wireless-channel-card.pdf.
12. (2012). NVIDIA GeForce GTX 680: The fastest, most efficient GPU ever built. White Paper. http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf.
13. (2013). CUFFT User Guide. http://docs.nvidia.com/cuda/cufft/index.html.
14. Coleri, S., Ergen, M., Puri, A., Bahai, A. (2002). Channel estimation techniques based on pilot arrangement in OFDM systems. *IEEE Transactions on Broadcasting*, *48*(3), 223–229.
15. Gardner, J.S. (2012). CEVA EXPOSES DSP SIX PACK-XC4000 family uses coprocessors to buff up the baseband. http://www.linleygroup.com/mpr/article.php?url=mpr/h/2012/10864/10864.pdf.
16. Huang, S., Xiao, S., Feng, W. (2009). On the energy efficiency of graphics processing units for scientific computing. In: *IEEE international symposium on parallel distributed processing (IPDPS'09)* (pp. 1–8).
17. Larsson, E. (2009). MIMO detection methods: how they work [lecture notes]. *IEEE Signal Processing Magazine*, *26*(3), 91–95.
18. Lee, S., Ahn, C., Choi, S. (2011). Implementation of Software-based 2X2 MIMO LTE base station system using GPU. In: *SDR-WInnComm*.
19. Micikevicius, P. (2012). Multi-GPU programming. In: *GPU technology conference*.
20. Proakis, J. G., & Salehi, M. (2008). In *Digital Communications, 5th edn*. New York: McGraw-Hill.
21. Wu, M., Sun, Y., Wang, G., Cavallaro, J. (2011). Implementation of a high throughput 3GPP turbo decoder on GPU. *Journal of Signal Processing Systems*, *65*(2), 171–183. http://dx.doi.org/10.1007/s11265-011-0617-7.
22. Zheng, Q., Chen, Y., Dreslinski, R., Chakrabarti, C., Anastasopoulos, A., Mahlke, S., Mudge, T. (2013a). Architecting an LTE base station with graphics processing units. In: *2013 IEEE workshop on signal processing systems (SiPS)* (pp. 219–224).
23. Zheng, Q., Chen, Y., Dreslinski, R., Chakrabarti, C., Anastasopoulos, A., Mahlke, S., Mudge, T. (2013b). Parallelization techniques for implementing trellis algorithms on graphics processors. In: *2013 IEEE international symposium on circuits and systems (ISCAS)* (pp. 1220–1223).
24. Zheng, Q., Chen, Y., Dreslinski, R., Chakrabarti, C., Anastasopoulos, A., Mahlke, S., Mudge, T. (2013c). WiBench: An open source kernel suite for benchmarking wireless systems. In: *2013 IEEE international symposium on workload characterization (IISWC)* (pp. 123–132).

Qi Zheng is a PhD candidate in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He received the B.S.E degree in electrical engineering from Harbin Institute of Technology, China, in 2010, and the M.S.E degree in computer engineering from University of Michigan, Ann Arbor, in 2012. His research interests include energy-efficient computer architecture of high throughput processor, and wireless communication system design.



Yajing Chen is a PhD student in Computer Science and Engineering at the University of Michigan, Ann Arbor. Her research interests focus on Software Defined Radio and throughput computing. She received MS in Electrical Engineering Systems at the University of Michigan, Ann Arbor.

Hyunseok Lee is an Associate Professor in the Department of Electronics and Communications Engineering at Kwangwoon University, Seoul, Korea. He received his Ph.D. in Computer Science and Engineering, University of Michigan, Ann Arbor, in 2007. He participated in the development of IS-95, cdma2000, WCDMA, and mobile WiMAX systems at Samsung Electronics, Suwon, Korea from 1992 to 2008. His research interest includes the low power signal processing architecture and embedded systems for wireless communications.



Ronald Dreslinski received the B.S.E. degree in electrical engineering, the B.S.E. degree in computer engineering, and the M.S.E. and Ph.D. degrees in computer science and engineering from the University of Michigan, Ann Arbor. He is currently a research scientist at the University of Michigan. His research focuses on architectures that enable emerging low-power circuit techniques.

Chaitali Chakrabarti received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1986 and 1990, respectively. She is a Professor with the Department of Electrical Computer and Energy Engineering, Arizona State University (ASU), Tempe and a Fellow of the IEEE. Her research interests include the areas of low power embedded systems design including memory optimization, high level synthesis and compilation, and VLSI architectures and algorithms for signal processing, image processing, and communications. She is currently an Associate Editor of the Journal of VLSI Signal Processing Systems, the IEEE Transactions of VLSI Systems and on the Senior Editorial Board of IEEE Journal on Emerging and Selected Topics in Circuits and Systems.

Achilleas Anastasopoulos (S'97-M'99) was born in Athens, Greece in 1971. He received the Diploma in Electrical Engineering from the National Technical University of Athens, Greece in 1993, and the M.S. and Ph.D. degrees in Electrical Engineering from University of Southern California in 1994 and 1999, respectively. He is currently an Associate Professor at the University of Michigan, Ann Arbor, Department of Electrical Engineering and Computer Science.

His research interests lie in the general area of communication theory, with emphasis in channel coding, multi-user channels, as well as connections between multi-user communications and decentralized stochastic control. He is the co-author of the book *Iterative Detection: Adaptivity, Complexity Reduction, and Applications,* (Reading, MA: Kluwer Academic, 2001).

Dr. Anastasopoulos is the recipient of the "Myronis Fellowship" in 1996 from the Graduate School at the University of Southern California, and the NSF CAREER Award in 2004. He served as a technical program committee member for ICC 2003 and Globecom 2004, and on the editorial board of the IEEE TRANSACTIONS ON COMMUNICATIONS.

Scott Mahlke is a Professor in the Electrical Engineering and Computer Science Department at the University of Michigan where he leads the Compilers Creating Custom Processors group (http://cccp.eecs.umich.edu). The CCCP group delivers technologies in the areas of compilers for multicore processors, energy efficient processor design, and reliable system design. Mahlke received the Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1997. Mahlke's achievements were recognized by being awarded the Most Influential Paper Award from the Intl. Symposium on Computer Architecture in 2007 and 2014 Monroe-Brown Foundation Education Excellence Award. He is a senior member of the IEEE Computer Society and the ACM. Contact him at mahlke@umich.edu.

Trevor Mudge received the Ph.D. degrees in Computer Sciencefrom the University of Illinois, Urbana in 1977. Since then, he has been on the faculty of the University of Michigan, Ann Arbor. In 2003 he was named the first Bredt Family Professor of Electrical Engineering and Computer Science after concluding a ten year term as the Director of the Advanced Computer Architecture Laboratory–a group of eight faculty and about 60 graduate students. He is author of numerous papers on computer architecture, programming languages, VLSI design, and computer vision. He has also chaired about 50 theses in these areas. His research interests include computer architecture,computer-aided design, and compilers. In addition to his position as a faculty member, he runs Idiot Savants, a chip design consultancy. Trevor Mudge is a Fellow of the IEEE, a member of the ACM, the IET, and the British Computer Society.