Datacenter Design for Future Cloud Radio Access Network

by

Qi Zheng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2016

For my wife Chunyang Zhai

ACKNOWLEDGEMENTS

First of all, I would like to start by thanking my advisor, Professor Trevor Mudge. He is not only an advisor for my academic and research work, but also a mentor for my future career and life. As an international student from the East, I learned a lot about the industry, the western history and culture, and the American society from Trev. All of these are more than helpful for my future development. Also, Trev gave me the freedom to explore research topics based on my own interests, and provided inspirational guidance and feedback whenever I needed it. I also would like to thank Professor Ronald Dreslinski, the co-chair of my thesis. Ron has been a big help to my research ever since I came to the University of Michigan. He provided detailed suggestion that inspired me to develop novel research ideas and new experimental methodologies.

Second, I would like to thank several other faculty members and senior researchers that I have closely worked with during my PhD career at University of Michigan. I have been working with Professor Chaitali Chakrabarti at Arizona State University through nearly all my life in the graduate school, and she is always very patient and thorough when advising my work or editing my “Chinglish” paper.

In addition, I would like to thank Yuan Lin, a former PhD student of Trev. Yuan helped me with many great research ideas, and I learned a lot from our "idea exchanging" discussion. I am also very honored to have the chance collaborating with Professor Scott Mahlke, Professor Achilleas Anastasopoulos, Professor Jason Mars and Professor Lingjia Tang at University of Michigan, and Professor Hyunseok Lee at Kwagnwoon University in Korean. Their suggestion and feedback is more than helpful for my research.

I want to thank all the colleagues from the "TRON" research group. I really enjoyed my five years in Ann Arbor, because I feel fortunate working with this group of talent. I will miss all of you, and the exciting flight, the remarkable can tower, the astonishing cooking, multilingualism, the supper abbreviation and Blue Oh: Pilot Nilmini Abeyratne, Yajing Chen, Cao Gao, Yiping Kang, Byoungchan Oh, Johann Hauswald, Jonathan Beaumont, Thomas Manville, Michael Cieslak, Michael Wollman, Mark Woh, Sangwon Seo, Korey Sewell, Joseph Pusdesris and The Tony Gutierrez.

I have known many new friends since I came to the United States. I would like to thank all graduate students in ACAL. Many of them asked me for suggestion before they ordered food for the CELab (the old ACAL) reading group, which is very honorable for me, as the free food is such an important part of a PhD student's life. I also thank all my Chinese friends who are mostly outside the division of computer science and engineering. They are the people that help me with my homesickness, and make me dumplings on the Chinese New Year.

Finally, I would like to thank my family for their love and support. I want to thank my wife Chunyang Zhai for going through with me the tough time of moving to a foreign and strange country. In addition, I want to thank her for letting me be the first “Doctor” in our small family. I would like to thank my parents for supporting and encouraging me these years. I as well want to thank my parents in law for their encouragement and advise.

Go Blue!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

API	application program interface
ASIC	application-specific integrated circuit
AWGN	additive white Gaussian noise
BER	bit error rate
BBU	baseband unit
BPSK	binary phase-shift keying
BTS	base transceiver station
CPU	central processing unit
C-RAN	cloud radio access network
CSI	channel state information
CUDA	compute unified device architecture
DLP	data-level parallelism
DRAM	dynamic random access memory
DSP	digital signal processor
DFT	discrete Fourier transform

DVFS dynamic voltage and frequency scaling

FFT fast fourier transform

FLOPs floating-point operation per second

FPGA field-programmable gate array

FSM finite-state machine

GPU graphics processing unit

GPGPU general-purpose computing on graphics processing unit

ILP instruction level parallelism

IPC instruction per cycle

LLR logarithmic likelihood ratio

LS least square

LTE long-term evolution

MGMN multiple GPUs across multiple network nodes

MGSN multiple GPUs within a single network node

MIMO multiple-input and multiple-out

MLP memory level parallelism

MMSE minimum mean square error

NII next iteration initialization

PCCC parallel concatenated convolutional code

OFDM orthogonal frequency-division multiplexing

QAM quadrature amplitude modulation

QPSK quadrature phase shift keying

RAN	radio access network
RRH	remote radio head
SC-FDMA	single-carrier frequency-division multiple access
SIMD	single instruction multiple data
SIMT	single instruction multiple thread
SISO	soft-input-soft-output
SM	streaming multiprocessor
SNR	signal-to-noise ratio
SoC	system-on-chip
TCO	total cost of ownership
TLP	thread level parallelism
TS	training sequence
WCDMA	wideband code division multiple access

ABSTRACT

Datacenter Design for Future Cloud Radio Access Network

by

Qi Zheng

Co-Chair: Trevor Mudge

Co-Chair: Ronald Dreslinski

Cloud radio access network (C-RAN), an emerging cloud service that combines the traditional radio access network (RAN) with cloud computing technology, has been proposed as a solution in the future to handle the growing energy consumption and cost of the traditional RAN. Through aggregating baseband units (BBUs) in a centralized cloud datacenter, C-RAN reduces energy and cost, and improves wireless throughput and quality of service. However, designing a datacenter for C-RAN has not yet been studied. To this end, in this dissertation, I investigate how a datacenter for C-RAN BBUs should be built on commodity servers.

I first design WiBench, an open-source benchmark suite that contains the key signal processing kernels of many mainstream wireless protocols, and study its characteristics. The characterization study shows that there is abundant data level parallelism (DLP) and thread level parallelism (TLP), and little branch instructions. Based on this result, I then develop high performance software implementations of C-RAN BBU kernels in C++ and CUDA for both CPUs and GPUs. In addition, I generalize the GPU parallelization techniques of the Turbo decoder to the trellis algorithms, an important family of algorithms that are widely used in data compression and channel coding.

Then I evaluate the performance of commodity CPU servers and GPU servers. The study shows that the datacenter with GPU servers can meet the LTE standard throughput with 4$\times$ to 16$\times$ fewer machines than with CPU servers. A further energy and cost analysis show that GPU servers can save on average 13$\times$ more energy and 6$\times$ more cost. Thus, I propose the C-RAN datacenter be built using GPUs as a server platform.

Next I study resource management techniques to handle the temporal and spatial traffic imbalance in a C-RAN datacenter. I propose a "hill-climbing" power management that combines powering-off GPUs and DVFS to match the temporal C-RAN traffic pattern. Under a practical traffic model, this technique saves 40% of the BBU energy in a GPU-based C-RAN datacenter. For spatial traffic imbalance, I propose three workload distribution techniques to improve load balance and throughput. Among all three techniques, pipelining packets has the most throughput improvement at 10% and 16% for balanced and unbalanced loads, respectively.

CHAPTER I

Introduction

1.1 Emerging of C-RAN

Mobile device users have increased rapidly over the two last decades. Based on the CTIA–The Wireless Association's annual wireless industry survey [6], the number of wireless subscribers has increased over ten times since 1995 (shown in Figure 1.1). By 2014, there were 355 million wireless subscribers and 300 thousand base stations in the United States, making wireless communication a market worth $200 billion annually [6]. This rapid growth of the mobile market has made wireless signal processing a key driving application of the computing technology, and a major consumer of the computing resources.

A crucial component of wireless communication is a radio access network (RAN), which connects mobile devices and the core network. Because of the need for 24/7 service availability and the growing requirements for high data rate, RAN systems consume significant energy and capital. In 2010, wireless base

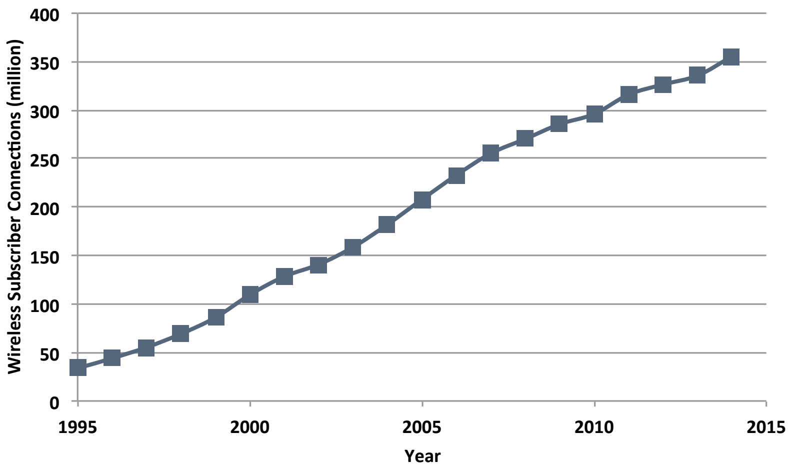


Figure 1.1 Annual wireless subscriber connections in the United States [6]

stations consumed 110 million kWh of energy and cost $40 billion on capital expenditure [41]. This constrains the traditional RAN growth in terms of both energy consumption and total cost of ownership (TCO).

In addition, the throughput of traditional RANs cannot meet the growing demand for higher data rates. Global mobile traffic increased 66-fold with a compound annual growth rate of 131% from 2008 to 2013, while the peak throughput of the wireless network only increased at 55% annually [41]. This has resulted in low data rates per user. For example, the typical user download speed of LTE is only 10% of the specification's peak data rate [19]. Therefore, the throughput of the traditional RAN is not enough, especially with new applications demanding high data rates, like 4K online videos and real-time online games. Consequently, we must find solutions to improve the throughput, energy and cost efficiency of the traditional RAN system.

To solve the problems that constrain the traditional RAN, cloud radio access network (C-RAN), a new emerging cloud service, has been proposed [41]. C-RAN is a domain specific cloud service that combines the traditional RAN with cloud computing technology. In C-RAN, the non-compute intensive remote radio heads (RRHs) are decoupled from the compute intensive baseband units (BBUs): RRHs remain at the distributed base station sites while BBUs are aggregated into a centralized cloud datacenter. The datacenter processes baseband computation from all sites that are connected to the datacenter through a high speed front-haul link.

C-RAN has many advantages including reduction in energy and TCO, and improvement in throughput and hardware utilization. On the front-end, removing the BBU from the base station makes them smaller and simpler, which reduces the energy and the TCO of the site. For example, site acquisition and rental fees are smaller as well as electricity costs and hardware upgrade costs. In addition, because the sites are smaller, more of them can be deployed in densely populated areas, which improves the quality of service. On the back-end, aggregating BBUs into a centralized datacenter saves maintenance cost, and improves hardware utilization and energy efficiency by sharing computing resources among sites. It also increases network capacity by enabling joint processing [93] (a technique to reduce interference from multiple base stations when a mobile device is at the edge of a coverage area). With higher hardware utilization, lower energy, and lower cost, operators can deploy more hardware to improve the throughput.

1.2 Contribution

Although C-RAN has been proposed for future wireless systems, the datacenter design for the C-RAN has not yet been studied. Therefore, in this dissertation, I explore the design of a datacenter for the future cloud based RAN. The primary goal of this work is to investigate how to build a C-RAN datacenter. In particular, we need to ensure C-RAN systems achieve the following targets:

- Meeting the throughput requirement specified in current and future wireless standards with commodity servers.
- Supporting the number of sites required by the current C-RAN design, and be able to scale up to support more sites for larger C-RANs in the future.
- Minimizing the energy consumption and the TCO of the C-RAN datacenter.
- Managing hardware resources in a C-RAN datacenter to handle the temporal and spatial imbalance in traffic.

To understand these challenges, I first design WiBench, an open source configurable benchmark suite that characterizes the computational features of the baseband signal processing systems of mainstream wireless protocols. Through the study of WiBench, several key features, such as the large amount of data level parallelism (DLP) and few branch instructions, are identified in order to guide the hardware choosing and software implementation for a C-RAN datacenter.

Based on WiBench, I realize a high-performance software model of the C-RAN BBU uplink receiver that includes all the kernels in the physical (PHY) layer and the Turbo decoder. I focus on the receiver side as it is significantly more computationally complex than the transmitter side, representing most of the computations in the C-RAN BBU. I use this model to investigate how to build a C-RAN datacenter with commodity general-purpose servers. I explore two major server platforms, which are multi-core CPUs and general-purpose GPUs. I realize high-performance implementation of an LTE BBU model in both C++ and CUDA for the evaluation on CPUs and GPUs, respectively. For the C++ implementation, I maximize the CPU performance by using automatic vectorization and openMP optimizations. For the CUDA implementation, I utilize various types of parallelism to maximize the GPU performance. In addition, I generalize the parallelization techniques of the Turbo decoder on GPUs to the trellis algorithm, a broader family of algorithms whose processing can be represented by a trellis. I explore different parallelization techniques to achieve the best tradeoff among the throughput, latency and the bit error rate.

To investigate the C-RAN datacenter design, we must first determine which server platform is better to be deployed. To this end, I compare CPU servers and GPU servers across performance, energy, and TCO. For the performance, I compare the throughput achieved by each type of server to the throughput defined by the LTE specification, and determine the amount of equipment that needs to be deployed in a C-RAN datacenter supporting 32 sites. Since the data and thread level parallelism present in many of the BBU kernels are better suited for the GPU architecture, the GPU server consistently achieves better performance than the CPU server. The results show that we need $4\times$ to $16\times$ as many CPU servers as the GPU servers in the equivalent datacenter. Then, I use the required number of CPUs and GPUs to evaluate the energy consumption and TCO in a 32-site C-RAN datacenter. The evaluation shows that the CPU servers consume, on average, 13$\times$ more energy and 6$\times$ higher TCO than the GPU servers. Therefore, I decide to choose GPU servers as the hardware platform to build the C-RAN datacenter.

In addition to the hardware platform selection, resource management in a C-RAN datacenter is also important. Since the traffic in the radio network is rarely equally distributed temporally and geographically, I investigate power management techniques that leverage temporal traffic imbalance to save energy, and load balancing techniques that leverage spatial traffic imbalance to improve datacenter throughput. For power management, I propose a "hill-climbing" management technique that combines powering-off GPUs and DVFS to reduce the datacenter power consumption without any performance impact. The study shows that in a 24-hour RAN traffic model, a datacenter with the proposed power management saves 40% of the BBU energy over no power management. For load balancing, I explore three techniques including fixed assignment, pipelining kernels and pipelining packets. These techniques methodically distribute the workload across multiple servers to improve the datacenter's throughput. The results show that pipelining kernels and pipelining packets achieve 12% and 16% more throughput than the fixed assignment at the cost of 40% longer latencies, which is still under the 4 ms LTE BBU latency budget. Overall, pipelining packets is the best load balancing technique due to its highest throughput and acceptable processing latency.

To sum up, the dissertation makes the following contributions:

- I design and characterize an open source configurable wireless signal processing kernel suite, which includes a rich set of key signal processing kernels that are used widely in mainstream wireless protocols.

- I develop different parallelization techniques on GPUs for the trellis algorithm, a family of algorithms that are widely used in data compression and channel coding. I study tradeoffs among the throughput, latency and the bit error rate.

- I investigate how C-RAN datacenters can be built on commodity server platforms. I achieve high-performance implementation of a model of the C-RAN BBU in both C++ and CUDA.

- I explore two major general-purpose server platforms, including multi-core CPUs and GPUs. I evaluate each server platform with performance, energy and TCO, and decide that GPU servers are the best hardware platforms to build the C-RAN datacenter.

- I propose a "hill-climbing" power management that combines powering-off GPUs and DVFS to match the temporal C-RAN traffic pattern. Under a practical traffic model, this technique saves 40% of the BBU energy.

- I propose three workload distribution techniques to balance the loads between sites. Among all three techniques, pipelining packets has the most throughput improvement of 16%.

In addition to the contributions mentioned above, I provide implications for future C-RAN datacenter designs, indicating that a C-RAN datacenter can benefit from architecture and instruction set support for trellis algorithms in general-purpose processors, and the support for internet service at the wireless edge.

1.3 Organization

The rest of this dissertation is organized as follows: in Chapter II the background information of the limitations of the traditional RAN and the benefits of C-RAN are discussed. In addition, traditional baseband processors and two major server platforms used in a datacenter, multi-core CPUs and general-purpose GPUs are introduced. Chapter III describes the design of WiBench, and the results of the corresponding characterization study. In Chapter IV, the methods to achieve high-performance implementation of the wireless baseband systems and trellis algorithms on GPUs are presented and evaluated. Chapter V discusses the design for a C-RAN datacenter, including the hardware choosing between multi-core CPUs and GPUs. It also shows the new resource management techniques for dealing with spatial and temporal traffic imbalance in a C-RAN datacenter. Finally, Chapter VI concludes the dissertation and proposes future research directions.

CHAPTER II

Background

In this chapter, I will introduce the radio access network, along with the motivating factors that leads to the invention of C-RAN architectures. I will also briefly introduce the traditional baseband processors, and the two major commercial general-purpose processors used in datecenters these days.

2.1 Radio Access Network

A RAN (shown in Figure 2.1) provides the wide-area wireless connection between mobile devices and the core network through radio technologies. Traditionally, it consists of a large number of distributed base transceiver stations (BTSs), which is responsible for coordinating the traffic and signaling between mobile devices and the network switching system [17], and a few centralized base station controllers, which handles radio resource allocations and user device handovers between BTSs. Since a RAN provides 24/7 services to mobile users, it

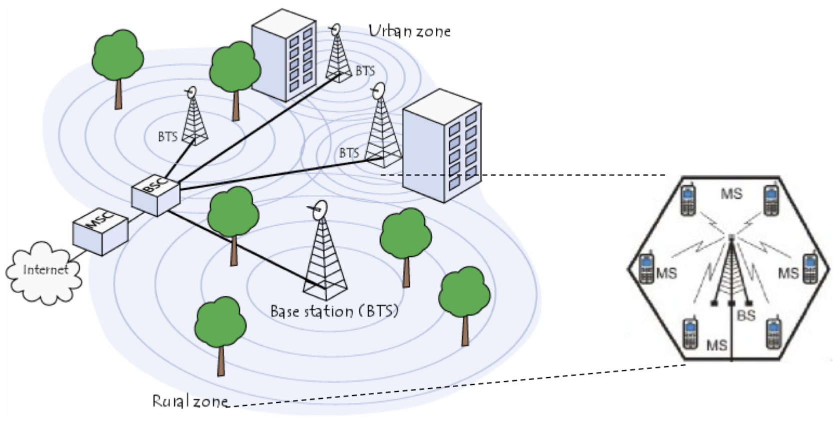


Figure 2.1 Radio access network [1, 2]

is the most costly part of mobile system infrastructure.

2.1.1 The limitations of traditional radio networks

In order to be always available to support 24/7 services as well as to meet the growing demands for high throughput, the traditional RAN requires every distributed base station to be in continuous operation and have peak-throughput computational capability. In addition, it needs to support frequent improvements to system hardware. However, this leads to many problems in today's RANs, including high power consumption, high TCO, limited network capacity, and low average hardware utilization.

High Power Consumption. Each BTS is responsible for the coverage of a small area and handles transmission/reception signals for all the devices in that area. BTSs must operate continuously. In addition, wireless providers are deploying ever more BTSs to increase coverage and offer more wireless services. Based

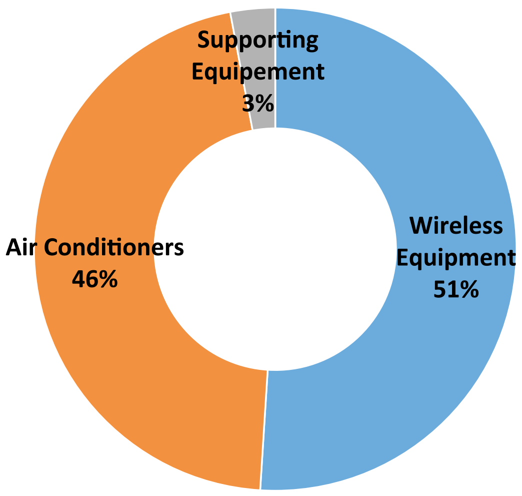


Figure 2.2 Power consumption breakdown in a wireless base station [41]

on a report from China Mobile [41], a major wireless operator, 72% of the RAN power consumption is from BTSs, and almost half of that power is consumed by the supporting facilities such as air conditioners (shown in Figure 2.2). Since reducing the number of BTSs is not an option (this will result in worse service quality and coverage), new technologies are in urgent demand to reduce the BTS power consumption.

High TCO. Rapidly increasing mobile data consumption leads to a growth in the operators' cost of the radio network. To improve the average revenue per user, operators need to reduce the RAN TCO. Currently, only 35% of the capital expenses of a BTS is spent on wireless equipment, while the remaining 65% is spent on site acquisition, civil works, equipment installation, etc. [41]. The most effective way to reduce TCO is to have fewer distributed BTS sites, which reduces the costs of both construction and maintenance. Since that is not an option, other ways to decrease the cost of the non-wireless functionality must be found without sacrificing network capacity and coverage.

Limited Network Capacity and Low Utilization. The fast growth of mobile devices and applications requires higher data rate from the wireless network. From 2008-2013, global traffic increased 66-fold with a compound annual growth rate of 131%. This means that wireless operators need to continuously increase network capacity and coverage. One way to achieve this is to have more BTSs to cover each area. However, deploying more BTSs per coverage area results in higher TCO and energy consumption, and is sometimes not even feasible due to inter-cell interference in high density areas, such as a football stadium. Therefore, the growth of network capacity of traditional RANs is limited by the energy and cost. In addition, BTSs are typically over-provisioned to support peak capacity, but the traffic of peak capacity only occupies 7% of the networks daily traffic [94], resulting in low RAN hardware utilization at other times, which exacerbates the problem of high energy consumption and TCO for the traditional RAN.

2.2 Cloud Radio Access Network

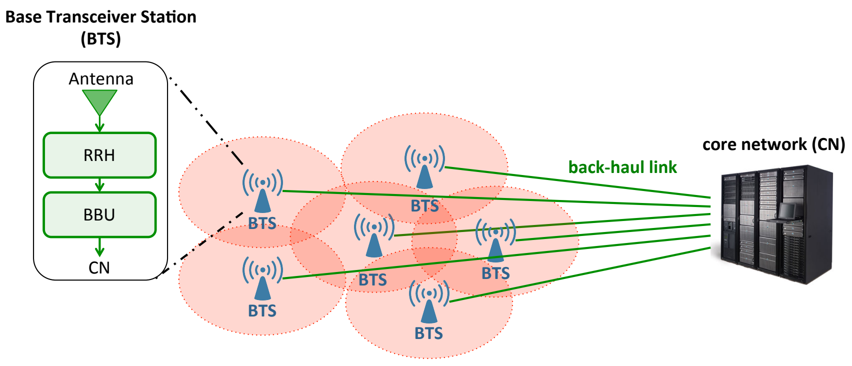
Due to the limitations mentioned above, the traditional RAN is constrained by high energy consumption and high cost to provide continuously increasing network capacity. This has inspired C-RAN, a new approach to designing base stations.

2.2.1 Introducing C-RAN

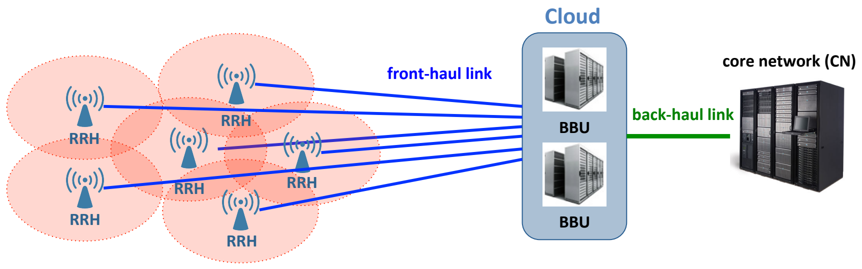
Cloud computing has had much success in the IT domain for centralized computing and energy/cost savings. The RAN and cloud computing share many features such as a large customer base, big geographical coverage area, and high traffic load. Recently, C-RAN, a domain specific cloud service that combines the traditional RAN with cloud computing technology, was proposed to solve the problems in the traditional RAN.

In the traditional RAN (shown in Figure 2.3a), a BTS is a distributed unit consisting of closely connected pairs of RRHs and BBUs. Every BTS processes data from the site that it covers, and transfers it back to the core network through back-haul links. C-RAN (shown in Figure 2.3b), on the other hand, decouples the RRH and BBU. The RRHs remain at distributed sites while the centralized cloud aggregates all the BBUs (BBU pool). Given a large geographical area, the cloud processes all the baseband computation jobs received from distributed RRHs. A high speed front-haul link [28, 64, 40] connects these two components.

In C-RAN, each RRH at a site is still responsible for transmitting and receiving radio signals and analog/digital conversion. Because the BBU no longer resides at the site, the distributed RRH is much smaller and simpler; it consumes much less energy and has less complexity that can lower its cost. This enables an increase in the density of RRH deployments in crowded areas to improve the quality of service. Typically, RAN base stations employ multiple BBUs to support different protocols such as 3G, WiMax, and LTE. With C-RAN removing the BBU from the



(a) Traditional RAN



(b) C-RAN

Figure 2.3 The structures of a traditional RAN system and C-RAN system. In a traditional RAN, each BTS is consisted of its own RRH and BBU, and is distributed to cover each site. In C-RAN, only RRHs are distributed, and BBUs sit in a centralized location, i.e. the cloud.

base station, the RRH can be used as a universal solution for all the protocols to further reduce cost.

2.2.2 Benefits of C-RAN

The biggest change that C-RAN has is that all of the baseband processing, which consumes most of the computational resources, is now moved from the distributed sites to a centralized location (the cloud). This enables sharing of computational resources between different sites, which achieves better hardware utilization when there is unbalanced traffic from sites. Another major advantage of C-RAN is the ability to more easily perform joint processing to remove inter-site interference. A mobile device located at the edge of a BTS's coverage area experiences low signal strength and interference from neighboring BTSs. With traditional RAN, multiple BTSs dynamically coordinate to jointly process signals to/from the mobile device, a process which involves handoffs between BTSs that incurs delay and wastes resources from multiple BTSs. In the C-RAN model, joint processing is easier, faster, and frees up valuable bandwidth.

In summary, C-RAN has the promise to provide the following benefits:

Better Energy Efficiency. C-RAN can reduce the energy consumption in the RAN system. The resource sharing in the cloud BBU pool leads to better resource utilization, improving the energy efficiency of the hardware. In addition, the distance between RRHs to users can be reduced due to the cancellation of inter-site interference by joint processing, which leads to lower RHH transmission power and thus saves energy. Based on field tests, C-RAN can save up to 71% of power compared to a traditional RAN [41, 54].

Lower Cost. Because BBUs are aggregated in the cloud, C-RAN reduces the cost of maintenance through centralized management and operation. In addition, smaller sites have smaller initial costs in site acquisition and equipment installation as well as smaller operational costs in site rental fees, electricity costs, and maintenance. Overall, C-RAN can save 44% of TCO when compared with a traditional

RAN [41].

Higher Capacity, Better Utilization. In C-RAN, the cloud BBU pool allows the sharing of traffic data and channel information between different sites. This enables joint processing and increases network capacity. It also allows more sites to be installed in high density areas, improving the quality of service. Since multiple sites share the same C-RAN BBU pool, C-RAN can dynamically allocate the compute resources to each site based on the traffic conditions, which achieves better hardware utilization. Previous work shows that 19% of compute resources can be saved by using C-RAN [35].

2.2.3 Design challenges for C-RAN

Although C-RANs outperform traditional RANs in every aspect, as shown in previous works [41, 54, 35], there remain a number of design challenges of C-RAN that has yet to be explored, such as designing a low-latency front-haul connection and the virtualization of the BBUs. The front-haul connection between RRHs and BBU datacenters in C-RAN is still an ongoing research area [80, 64, 29]. The original solution proposed for C-RAN is to use optical cable [80], for fast data transfer and very little latency impact. Cheap alternatives of optical cables are under development. In this thesis, I focus on the computer architectural challenges of the hardware and software design inside a C-RAN datacenter.

Many C-RAN solutions [63, 42, 41] opt for using general purpose processors instead of system-on-chip (SoC) with DSP cores and ASIC accelerators. Their evaluations show that general purpose platform based C-RAN datacenters outperform SoCs with much lower TCOs. This is due to the fact that SoCs are custom-designed for every wireless technology standard. Upgrading to newer wireless standards requires a redesign of SoCs, resulting in longer time to market and higher capital expenditure. Also, multiple versions of SoCs are required to support different protocols used at the same time, leading to even higher costs. As a new standard is introduced, the adoption rate is not instantaneous, meaning that ratios of hardware for the old standards versus the new standards change over time. This results in the need to continuously update the system. General purpose processors, on the other hand, are easier and cheaper to deploy and upgrade, as the C-RAN BBU system is implemented in software and can be dynamically configured for multiple standards. Therefore, in this thesis, I only focus on designing C-RAN datacenters with general purpose platforms, and will not consider SoCs. Although a solution with SoCs may have better energy efficiency, the lack of flexibility will result in higher TCOs of C-RAN systems, as shown in the previous works [63, 42, 41].

Moving BBUs to general purpose platforms introduces new challenges for C-RAN design. We need to develop high-performance software implementations of the wireless signal processing on general purpose servers. In addition, the datacenter should be able to support at least 20 sites to fulfill the specification's throughput requirement of the wireless standards, at low energy consumption and TCO. To achieve these, I analyze the computational features of the C-RAN baseband system, and study how to optimize them for high throughput and low latency processing.

2.3 Traditional baseband processors

A baseband processor is a hardware processing unit that manages all the radio related functions, and is one of the key devices in a RAN. Traditionally, there are several different types of hardware platforms deployed for baseband processing, achieving different design tradeoffs among performance, energy efficiency, flexibility and cost.

2.3.1 ASIC

ASICs (Figure 2.4) are among the most widely used baseband processors in today's commercial wireless base stations, as well as mobile devices. ASICs are customized integrated circuits for particular uses, therefore, they have fixed functionalities and supported configurations. Because of the customization, ASICs can achieve both good performance and high energy efficiency. In a wireless base station, a baseband ASIC is usually called a modem, and there are many modems in each base station to process data from different communication channels or different cells.

However, the biggest issue of ASICs is the customized functionality, which eliminates the flexibility. Consequently, ASIC-based baseband processors require more efforts and longer time to market when a base station is upgrading to a new protocol.

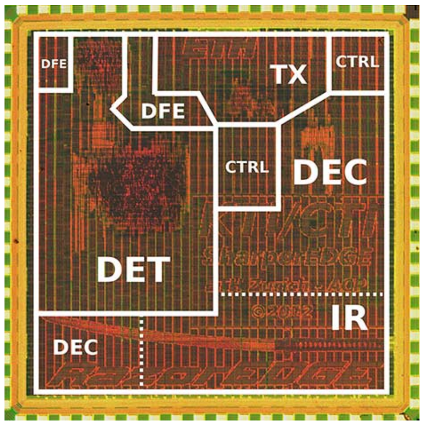


Figure 2.4 GSM/EDGE Baseband ASIC [78]

2.3.2 DSP

DSPs are specialized processors that target on digital signal processing applications, and also widely used in commercial wireless systems. There are many DSPs designed by both industry and academia as baseband processors for different wireless protocols [58, 86, 111, 112]. DSPs are usually designed with very long instruction word (VLIW) and single instruction multiple data (SIMD) pipelines to explore instruction level parallelism (ILP) and DLP in applications. Fully programmable DSPs only exist in the academic research [86, 112]. Commercial DSPs contain many accelerators for different purposes, tyring to achieve a good tradeoff between energy efficiency and flexibility. Therefore, when upgrading a base station to new protocols, DSPs with new accelerators still need to be designed, which leads to a long time to market. In addition, since many hardware components in a DSP, such as the memory system and the shuffling network, are highly specialized, it is non-trivial for a programmer to even write a working program.

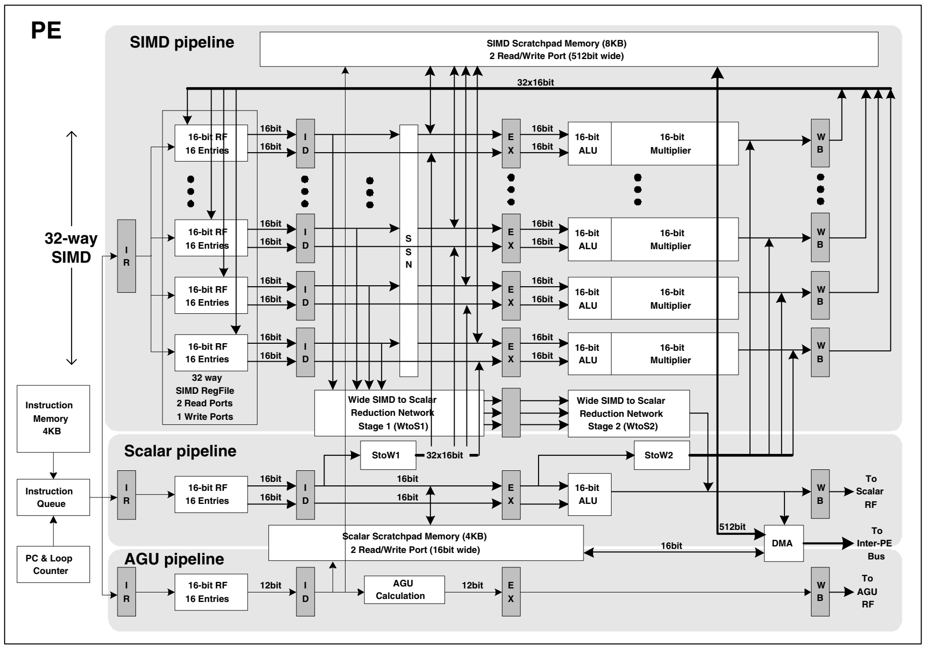

PE
SIMD pipeline
SIMD Scratchpad Memory (8KB)
2 Read/Write Port (512bit wide)
32x16bit
32-way SIMD
16-bit RF 16 Entries
16bit
16-bit ALU
16-bit Multiplier
SSN
32 way SIMD RegFile 2 Read Ports 1 Write Ports
Wide SIMD to Scalar Reduction Network Stage 1 (WtoS1)
Wide SIMD to Scalar Reduction Network Stage 2 (WtoS2)
Instruction Memory 4KB
Instruction Queue
PC & Loop Counter
Scalar pipeline
StoW1
StoW2
Scalar Scratchpad Memory (4KB)
2 Read/Write Port (16bit wide)
512bit
DMA
To Scalar RF
To Inter-PE Bus
AGU pipeline
AGU Calculation
12bit
To AGU RF

Figure 2.5 SODA, a DSP designed for WCDMA [86]

2.3.3 FPGA

Field-programmable gate array (FPGA) is another type of hardware platforms that is traditionally used for baseband processing. Big companies like Xilinx and Altera have developed commercial FPGA-based solutions. FPGAs have good programmability, as users can change the hardware functionality by reprogramming FPGAs through hardware description languages (HDLs), such as Verilog HDL and VHDL. This makes FPGAs very attractive when finding tradeoffs between energy efficiency and flexibility in a base station.

However, FPGAs have relatively high prices compared to other baseband processors. In addition, the design and verification of the FPGA-based hardware is difficult and time consuming, compared with regular programming software.

2.4 General-purpose processors

There are two major types of general-purpose processors used in datacenters: the traditional multi-core CPU, and the newly emerging GPU.

2.4.1 Central Processing Unit

Central Processing Units (CPUs) are the most commonly used general-purpose processors in a datacenter these days. With many hardware designs to provide high-performance computing, such as branch predictor, out-of-order execution, SIMD extension, and cache-based memory hierarchy, CPUs can explore several types of parallelism (like DLP and ILP) in a program, and provide fast processing for a single job. In addition, with the quick development of the semiconductor technology, CPUs with multiple cores and supports of multithreading to explore thread level parallelism (TLP) are widely deployed. This enables CPUs also to provide high throughputs for many concurrent works, making CPUs the perfect hardware platform for current datacenter applications. As there will be higher transistor density on a chip, processors with 10 to 100 cores will be available [25], and the throughputs provided by CPUs will increase even higher.

In addition to the high performance, CPUs also have good general-purpose programmability along with mature tool-chain supports. C libraries with SIMD intrinsics and multithreading application programming interfaces (APIs) such as

OpenMP make it easy for regular programmers to make use of DLP and TLP in an application. Compared to any ASIC or specialized hardware solution, this greatly reduces the development time and maintenance effort of a product.

2.4.2 Graphics Processing Unit

Graphics Processing Unit (GPUs) are originally designed for graphics applications, and have become emerging general-purpose processors that achieve high computing throughput through effectively explore DLP and TLP in a program. Due to their highly parallel architecture, GPUs have high raw compute power per dollar and Joule, make them very attractive for many datacenter applications.

2.4.2.1 GPU architecture

Although the microarchitectures of GPUs vary between different vendors, they all deploy the single-instruction multi-thread (SIMT) execution model in a similarly way. Therefore, in this section, I use the NVIDIA Fermi [96] architecture as an example.

Figure 2.6 shows the architecture of a Fermi GPU. A Fermi GPU is consisted of several streaming multiprocessors (SMs), a shared L2 cache and an external high-bandwidth DRAM. Each SM contains thirty-two execution units, a workload scheduler (warp schedule), a register file, and a 64 KB L1 memory that is configured as a combination of data cache and shared memory. In each SM, threads are issued to execution units in groups of 32, called warps. A warp works in the SIMD style:

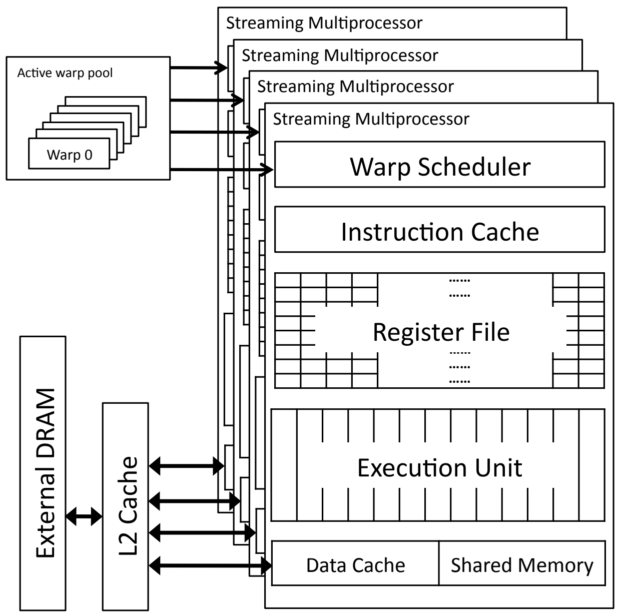


Figure 2.6 Fermi GPU Architecture

threads in the same warp execute the same instruction on different pieces of data. In every clock cycle, a warp that is ready for execution is selected and issued by the scheduler. To hide the long memory access latencies of the L2 cache and external DRAM, a GPU also supports fine-grained multithreading. The scheduler selects a warp from the active warp pool every cycle and issues an instruction from that warp. In the next cycle, the scheduler can select a different warp, because it supports a zero-cycle context switch. Thousands of threads are concurrently present in an SM as candidates for issuing, in order to make full use of available computing resources and keep execution units busy. In addition, there are usually multiple SMs in a commercial GPU, making the total number of concurrent threads supported even higher. Consequently, although the performance of a single thread is not improved, the overall throughput of a GPU is very high, usually on the level of giga-FLOPs or even tera-FLOPs.

2.4.2.2 Computer Unified Device Architecture

Computer Unified Device Architecture (CUDA) is a parallel computing platform and programming model generated by NVIDIA, which enables general-purpose computing on their GPUs. By using CUDA, programmers have direct access to the NVIDIA GPU's virtual instruction set and parallel computing components.

CUDA is designed similar to C/C++ in terms of the programming language fashion. When writing a GPU function (called a "kernel"), programmers need to use keywords "__global__" or "__device__" before the function definition to specify that it is a GPU function. Inside a kernel, everything is the same as a function written for CPU in C/C++, except that programmers need to specify the right index of the data for every instruction, because multiple copies (each is called a "thread") of the same code with different data indices will be automatically created by the CUDA driver during the runtime.

When launching the GPU kernel, programmers need to set the number of threads that will be created. In order to manage the shared computing resources (such as the shared memory and the register file) between threads, CUDA provides "thread block" and "grid" as the resource managing units. A thread block is a collection of threads that share computing resources and will be launched to the same SM. It is the smallest resource management unit in CUDA. A grid is a collection of thread blocks, and can be set as a three dimensional structure. When programmers set the number of threads, they need to specify the number of threads per thread block, and the number of thread blocks per grid. In addition, the data movement between the CPU and GPU memories needs to be explicitly coded in the program.

CUDA also has mature tool-chain support, including debugger, optimized high-performance libraries and profiler. This makes CUDA and GPUs promising and attractive for general-purpose applications with plenty of DLP and TLP.

CHAPTER III

WiBench: Characterize Wireless Baseband Signal Processing

The rapid growth in the number of mobile devices and the higher data rate requirements of mobile subscribers have made wireless signal processing a key driving application of mobile computing technology. In addition, although the future wireless access network will be cloud based, the underlying signal processing algorithms are the same as existing wireless protocols. Therefore, to guide a better design of hardware platforms for both the wireless network infrastructure and mobile equipements, it is very important for computer architects and system designers to understand and characterize the performance of existing and upcoming wireless protocols.

In this chapter, I present a newly developed open-source benchmark suite called WiBench. It consists of a wide range of signal processing kernels used in many mainstream standards such as 802.11, WCDMA and LTE. The kernels include Fast Fourier transform (FFT), multiple-input and multiple-out (MIMO), channel estimation, channel coding, constellation mapping, etc. Each kernel is a self-contained configurable block which can be tuned to meet the different system requirements. Several standard channel models have also been included to study system performance, such as the bit error rate. The suite also contains an LTE uplink system as a representative example of a wireless system that can be built using these kernels. WiBench is provided in C++ to make it easier for computer architects to profile and analyze the system.

Through characterizing WiBench, architectural analyses on each individual kernel and on the entire LTE uplink are performed, indicating the hotspots, available parallelism, and runtime performance.

3.1 Overview and Background

The mobile market has experienced a rapid increase over the last decade. It is expected that by the end of 2013 there will be almost as many mobile-cellular subscriptions as there are people in the world [75]. The number of mobile broadband subscribers, who access the internet wirelessly through mobile devices, has climbed from 268 million in 2007 to 2.1 billion in 2013—a 40% annual increase rate [75]. To support this growth the number of base stations has also increased exponentially [101]. All indications show that this trend is likely to continue, at least in the near future.

In order to design better hardware platforms for both the wireless access network and mobile devices, computer architects and system designers will have to understand and characterize the performance of wireless protocols. In a nutshell, wireless protocols encode the raw information in the transmitter side, and recover it in the receiver side. These processes consume significant computing resources and power in a handheld system. For instance, a GSM subsystem in a smartphone consumes 30%-50% of the overall power [39], and an even larger portion is used in more recent WCDMA and LTE protocols. In addition, the portion of the global CO_2 footprint for wireless networks will be 13% of the total allocation to information and communication technology (ICT) by 2020, according to the Climate Group [44]. Clearly it is important that wireless devices be power-efficient—requiring designers to understand the power/performance characteristics of the algorithms within these protocols.

Benchmarks are an important tool for characterizing power/performance trade-offs in different application domains. Examples of important benchmark suites include SPEC benchmarks [100] for general-purpose computing, PARSEC benchmarks [36] for multithreaded applications, MEVBench [43] for mobile computer vision applications, and BBench [68] for interactive smartphone applications. Although there exist some benchmarks for wireless communication, they either are out-of-date, lack essential algorithm details, or distorted the computational characteristics by introducing addition overhead.

In this chapter, I develop an open source configurable kernel set for wireless signal processing called WiBench[1]. The set consists of important signal processing

[1]WiBench is available through http://wibench.eecs.umich.edu.

kernels that are widely used in many wireless standards such as 802.11, WCDMA or LTE. The kernels include FFT, MIMO detection, channel estimation, channel coding, constellation mapping, and scrambling. Each kernel is a self-contained configurable block. Such a system can be used to build multiple wireless protocols and evaluate their performance. To demonstrate this feature, I include an LTE uplink benchmark in WiBench. LTE is a fourth generation wireless communication standard (4G) that is being deployed worldwide. It is designed to deliver data rates up to 100 Mbps. The configurability of WiBench kernels allows the LTE uplink to support a variety of specification data rates ranging from 1.56 to 100 Mbps. I also include several standard channel models in WiBench so that system researchers can use it to evaluate the bit error rate (BER) performance of their system. WiBench is provided in C++, which enables architecture researchers to characterize applications, and MATLAB, which helps debugging and functional verification.

The key contributions of this work are:

- An open source configurable wireless signal processing kernel suite, which includes a rich set of key signal processing kernels that are used widely in mainstream wireless protocols.

- An LTE uplink in the benchmark that illustrates how to build a wireless application by assembling kernels. The configurability of the kernels allows us to support different specification data rates. Users can similarly establish their own applications to model WCDMA or Wi-Fi.

- Benchmark support for several standard channel models that allows system designers to evaluate their decisions by examining BER.
- A demonstration of WiBench for hardware design which analyses and identifies the hotspots, available parallelism, and runtime performance at the kernel and system levels.

The rest of chapter is organized as follows. In Section 3.2, I describe the kernel suite, and provide details of each kernel in the benchmark and the LTE uplink. I also explain the design philosophy of WiBench. In Section 3.3, I examine the characteristics of each individual kernel and the LTE uplink, and provide suggestion for efficient hardware design. Section 3.4 presents the related work.

3.2 Benchmark Description

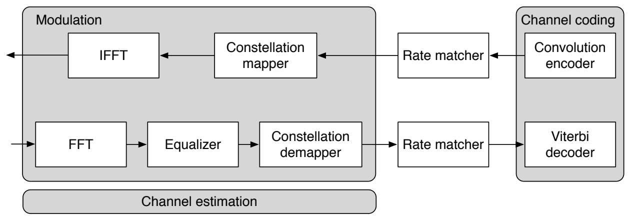
3.2.1 Design philosophy

WiBench was built to handle multiple wireless protocols. Thus, unlike some recent benchmarks [101], WiBench was constructed with configurable kernels, which are the basic blocks for multiple wireless systems. The intent is for users of current and possibly future wireless systems to be able to design their own system using these building blocks and characterize them. Figure 3.1 illustrates the downlink flow charts of several mainstream wireless systems. It shows that different wireless systems actually share a lot of common signal processing kernels. In this work, I selected kernels that are most frequently used and are representative

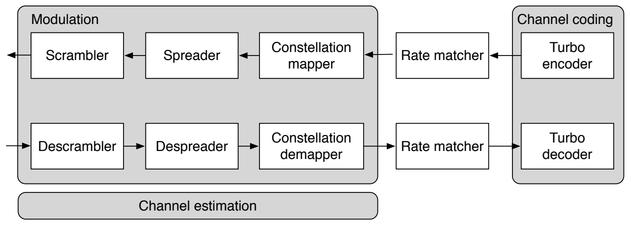
Table 3.1 The components of WiBench

Category	Benchmark
Kernels	Channel coding/decoding
	Rate matching
	Scrambling/Descrambling
	Constellation mapping/demapping
	MIMO detection
	FFT/IFFT
	Sub-carrier mapping/demapping
	Channel Estimation
Channel models	Gaussian Random Channel model (GRC)
	Extended Pedestrian A model (EPA)
	Extended Vehicular A model (EVA)
	Extended Typical Urban model (ETU)
Applications	LTE uplink

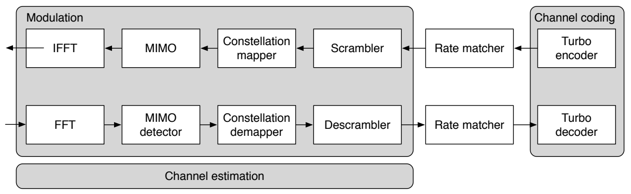
of the algorithms that are used in many wireless protocols. I also include several standard channel models so that system designers can test the performance of their systems under different channel conditions. Additionally, I show users how to use these kernels to build their own wireless systems by including an LTE uplink system in the benchmark. Table 3.1 summarizes the details of the benchmark. WiBench is originally written in C++, but a MATLAB version is also provided to facilitate debugging and functional verification.



(a) 802.11a



(b) WCDMA



(c) LTE

Figure 3.1 The downlink flow charts of 802.11a, WCDMA and LTE [86]. This figure shows that different wireless systems have many common signal processing kernels, such as FFT/IFFT, channel coding, constellation mapping, etc. I picked the most frequently used algorithms to include in the benchmark.

3.2.2 Introduction of kernels

3.2.2.1 Channel coding

Channel coding is the technique used to control errors in data transmission over noisy channels that enable reliable delivery of digital data. There are many different channel coding techniques such as convolutional codes [109], Turbo codes [31], and Low Density Parity Check codes (LDPC) [61], etc. The Turbo codes I chose

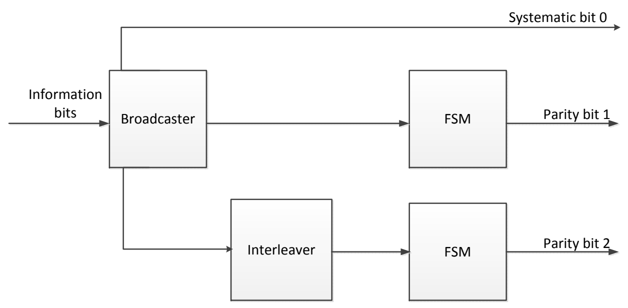


Figure 3.2 The structure of the Turbo code encoder. The Turbo encoder consists of two FSMs and an interleaver. The outputs of the encoder are the original input sequences interleaved with outputs of two FSMs.

belong to a high-performance forward error correction family of codes widely used in 3G/4G mobile communications.

The scheme of the Turbo encoder is a Parallel Concatenated Convolutional Code (PCCC) with two Finite State Machines (FSM) and one internal interleaver. The structure of the Turbo encoder for $R = 1/3$ code is shown in Figure 3.2; One information bit is encoded into three transmitted bits.

The Turbo decoder architecture includes two Soft-Input-Soft-Output (SISO) decoders [87] and one internal interleaver/deinterleaver as illustrated in Figure 3.3. Inside each SISO decoder, a forward and backward trellis traversal algorithm is performed [87]. The Turbo decoder works in an iterative fashion—increasing the iteration number results in a better error correction performance at the cost of higher computation. The Turbo code implementation supports 188 different input

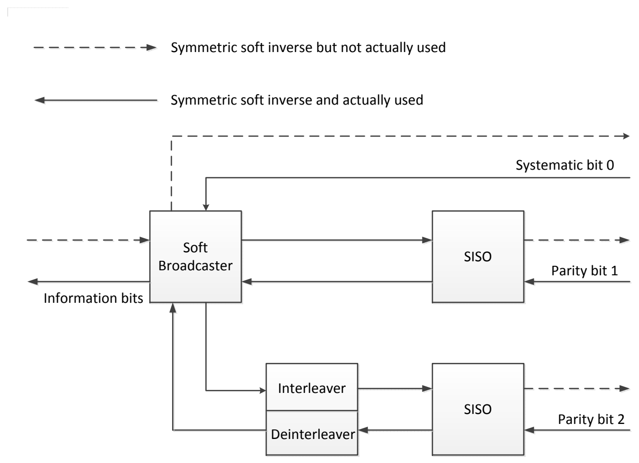


Figure 3.3 The structure of the Turbo code decoder. It consists of two Soft-Input-Soft-Output decoders and an interleaver. The decoder works in an iterative fashion.

lengths from 40 to 6144.

3.2.2.2 Rate matching

The purpose of rate matching is to provide a variety of channel coding rates from a single “mother code” with a fixed rate *R*. This considerably increases the flexibility of a system in terms of the performance-complexity tradeoff of channel coding. Rate matching is performed by puncturing or by repeating coded bits. Internally, the rate matching algorithm buffers the incoming bit stream and does bit collection, selection and pruning.

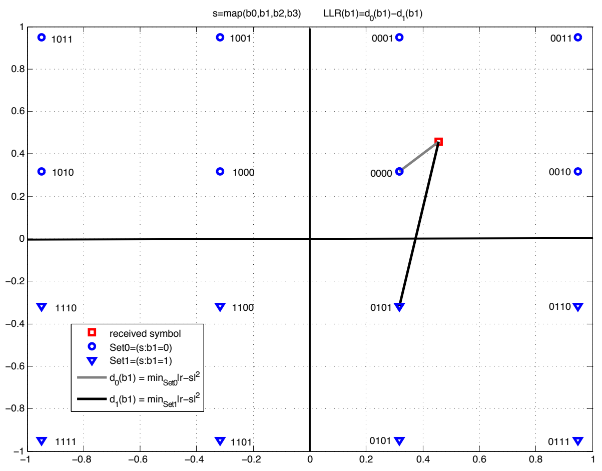


Figure 3.4 The constellation demapping of 16QAM. In constellation mapping, every four binary bits are mapped to one of the sixteen complex values (circles and triangles). In constellation demapping, the distances between a received symbol (square) and all sixteen complex values (circles and triangles) is computed and the distances are used to recover the four bits of data.

3.2.2.3 Scrambling/Descrambling

Scrambling encrypts and randomizes data. It encodes the transmitted information to make it unintelligible to a potential eavesdropper. The bit stream in a subframe is scrambled with a User Equipment (UE) specified scrambling sequence in the transmitter, which is reversed by descrambling at the receiver side. The implementation supports arbitrary lengths of scrambling.

3.2.2.4 Constellation mapping/demapping

The goal of constellation mapping is to represent a binary data stream with a signal that matches the characteristics of the channel [98]. The binary sequences are grouped and mapped into complex-valued constellation symbols. Figure 5.12

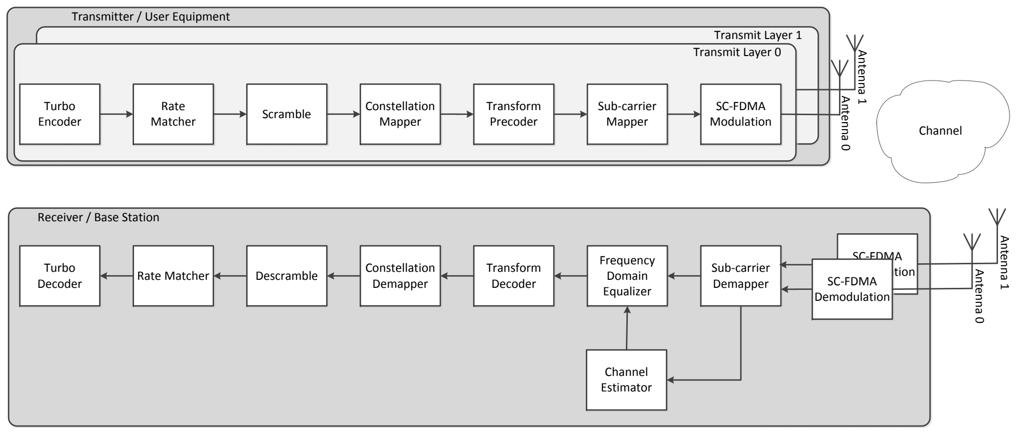


Figure 3.5 The system flow graph of the LTE uplink. It contains the Turbo coding, rate matching, scrambling, constellation mapping, transform precoding, sub-carrier mapping, SC-FDMA modulation, channel estimation and equalization.

shows a 16 Quadrature Amplitude Modulation (16QAM) constellation, where every four bits are mapped to one of the sixteen complex values (circles and triangles in Figure 5.12). I implemented BPSK (1-bit Constellation), QPSK (2-bit Constellation), 16QAM (4-bit constellation), and 64QAM (6-bit constellation) mapping in the benchmark.

Constellation demapping retrieves the binary stream from the signal by generating either hard or soft information. Hard information selects and outputs the binary representation of the closest symbol to the received signal (e.g., (0000) in the example of Figure 5.12). Soft information computes likelihood ratios for each bit that will be used by the channel code decoder as bit metrics. Figure 5.12 interprets the process of generating logarithmic likelihood ratios (LLRs) for the second bit (i.e., bit $b1$) of a received symbol r.

3.2.2.5 Multiple-Input Multiple-Output

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. *MIMO for spatial multiplexing* transmits independent data streams from each of the multiple transmit antennae, thus increasing the system data rate. *MIMO for diversity* transmits a single data stream from each of the multiple transmit antennae. The single data stream is coded by space-time coding, which improves the reliability of data transmission. There are various MIMO detection methods, for example, linear detection, sphere decoder, lattice reduction detection, etc. I include some widely used algorithms in WiBench, including a Least Square (LS) based zero forcing detection and a tree based sphere decoder. The MIMO detection module includes different antenna configurations including 1×1, 2×2 and 4×4.

3.2.2.6 FFT/IFFT

Discrete Fourier Transform (DFT) is one of the most frequently used transformations in science and engineering. It transforms a finite set of samples of a function in the time domain into frequency domain; inverse IDFT reverses this operation. FFT is a fast algorithm to compute DFT. It requires only $O(NlogN)$ operations to get the same result as DFT. I utilized FFTW [60] to implement FFT/IFFT. FFTW is a C library for computing the DFT that adapts to the running hardware platform to maximize performance. Its performance is competitive with, or even better than, some highly-tuned FFT implementations such as Suns Performance Library and IBMs ESSL library [59]. The kernel supports any FFT/IFFT size in the form of $2^a \cdot 3^b \cdot 5^c \cdot 7^d$.

3.2.2.7 Sub-carrier mapping/demapping

The mapping kernel inserts data and reference symbols into the sub-carrier. If multiple users exist in the system, their data will be mapped into non-overlapping sub-carriers. The demapping kernel extracts data and reference symbols from the sub-carrier for each user in the system.

3.2.2.8 Channel estimation

In order to achieve reliable communication most kernels in the receiver side require knowledge of the channel parameters, also known as Channel State Information (CSI) [18]. CSI can be obtained in two ways. One is to insert known symbols as pilots into data sequences, and the performance of pilot signals is used for estimation. The other is blind estimation by using knowledge of statistical characteristics of the received signal. Most blind methods suffer from several drawbacks such as slow convergence speed, high complexity, and poor performance. As a result pilot aided channel estimation is more common, therefore, I adopt it for WiBench. There is a choice of algorithms for pilot aided channel estimation, including Least Square (LS) estimation and Minimum Mean Square Error (MMSE) estimation. MMSE estimation provides better performance than LS, but requires more computation and sophisticated statistical characteristics of the channel. I include both LS and MMSE kernels in the benchmark.

3.2.3 Channel models

The channel model represents the characteristic degradation of the signal as it is transmitted wirelessly through the environment. In order for system designers to measure the BER that a particular receiver configuration experiences there are several standardized channel models. The basic channel model is a Gaussian random channel (GRC) which introduces Gaussian noise to the signal. In addition to the GRC, I include several other channel models—Extended Pedestrian A model (EPA), Extended Vehicular A model (EVA), and Extended Typical Urban model (ETU) [21]—which provide more realistic channel scenarios.

3.2.4 Introduction of the application: LTE uplink

I built an LTE uplink system to illustrate how to use the kernel and channel models provided in WiBench to create a complete wireless link. In addition, the LTE uplink system is an essential component consisting of the cloud wireless access network. The LTE uplink system is organized as shown in Figure 3.5. I implemented the entire physical layer as well as the most compute-intensive parts of the transport layer including the Turbo decoder and rate matching. The LTE uplink supports configurations covering all transmission bandwidths whose specification data rate ranges from 1.56 to 100 Mbps. In Section 3.3 I will evaluate the performance of each kernel in the LTE uplink and show an example system analysis by determining the BER under different channel conditions. In the following subsections, I describe in more detail the specific kernel choices for the LTE application.

3.2.4.1 Turbo encoder/decoder

The FSM of the Turbo encoder in the LTE specification is an 8-state recursive systematic convolutional encoder [22]. For the analysis, I set the iteration number of the Turbo decoder at 5. Although I have fixed the iteration number, WiBench could be used to explore the trade-off between BER performance and the amount of computation for different numbers of iterations.

3.2.4.2 Single carrier frequency diversity multiple access (SC-FDMA)

SC-FDMA is a precoded Orthogonal Frequency Diversity Multiplexing (OFDM) scheme, which has an additional transform precoding step that precedes the conventional OFDM processing. OFDM processing encodes data on multiple carrier frequencies. OFDM is applied in the LTE downlink (base station to user equipment), while SC-FDMA is realized in the uplink (user equipment to base station). Compared to OFDM, SC-FDMA has two main advantages that are critical to the uplink transmission: 1) SC-FDMA has a lower Peak-to-Average Power Ratio; 2) SC-FDMA is less sensitive to frequency offsets than OFDM.

In the transmitter, I implement the OFDM step of SC-FDMA by performing IFFT and inserting a Cyclic Prefix (CP). In the receiver, I eliminate the inter-symbol interference by removing CPs and converting data from the time domain to the frequency domain by FFT. The transform precoding step of SC-FDMA is done with a $2^a \cdot 3^b \cdot 5^c$ mixed radix FFT, while the IFFT is performed in the transform decoder at the receiver side.

3.2.4.3 Channel estimation

The LTE uplink transmission uses the comb-type pilot arrangement [23], where only time domain interpolation needs to be applied. The uplink pilot reference symbols from different transmit antennae occupy the same sub-carriers. However, pilot reference symbols are designed so that they can be distinguished from each other at the receiver side. Channel estimation takes the received signal and known pilot reference symbols to estimate the CSI, which is then used to compute the channel coefficients. I selected the frequency domain least square estimator that provides an acceptable performance with reasonable computation under the assumption that I have no knowledge of the channel [45].

3.2.4.4 Equalizer

The equalizer I apply is a zero forcing MIMO detector in the frequency domain. Taking advantage of OFDM/SC-FDMA, channel equalization in LTE can be implemented simply by a Frequency Domain Equalizer (FDE) with the coefficients estimated by the channel estimator.

3.3 Characterization of WiBench

Table 3.2 System configurations of the profiling platforms

Feature	Configuration	
	Datacenter platform	Mobile platform
Operating System	Linux 3.2.0-38	Linux 3.2.0-39
Processor	Intel Core i7 2600	Intel Atom 330
Frequency	3.40 GHz	1.60 GHz
L1 I-Cache	32 KB	32 KB
L1 D-Cache	32 KB	24 KB
L2 Cache	256 KB	512 KB
Last Level Cache	8 MB	N/A
Memory	16 GB DDR3	4 GB SDRAM
Out-of-order	Yes	No
Single core issue width	4	2
SIMD	128-bit, SSE2, SSE3, SSSE3, SSE4	128-bit, SSE2, SSE3, SSSE3

To expose the computational features of the wireless signal processing as well as illustrate how WiBench can be used for hardware design and system study, four studies to characterize the benchmark suite are performed.In addition, two types of processors, one for datacenters and the other for mobile systems, are deployed to demonstrate the wireless system characteristics on both wireless base stations and embedded platforms.

First, I profiled each individual kernel, determining how each performs on

Table 3.3 The configurations of the individual kernel

Kernel	Configuration
Turbo decoder	code rate $= 1/3$, codeword length $= 1184$
Descrambling	sequence length $= 300$
Constellation demapping	QPSK, sequence length $= 150$
FFT	128
IFFT	75
MIMO	2×2, sequence length $= 75$

different processors. This type of analysis can be used by hardware architects to design the underlying hardware to achieve power-efficient systems, and by code designers to better target optimization points. Second, I explore the performance of the LTE uplink included in the benchmark for different bandwidth requirements. Third, I show how different LTE uplink configurations with the same bandwidth impact the relative importance of each kernel. Finally, I perform an analysis of how the LTE uplink performs, in terms of BER, under one type of channel conditions. This type of analysis can be used by system designers to explore how their design performs under different channel conditions.

3.3.1 Experimental setup

The analyses are performed on cores that are used in both datacenter systems and embedded devices, because wireless applications run on both embedded platforms (e.g. smartphones) and server-like machines (e.g. wireless base stations).

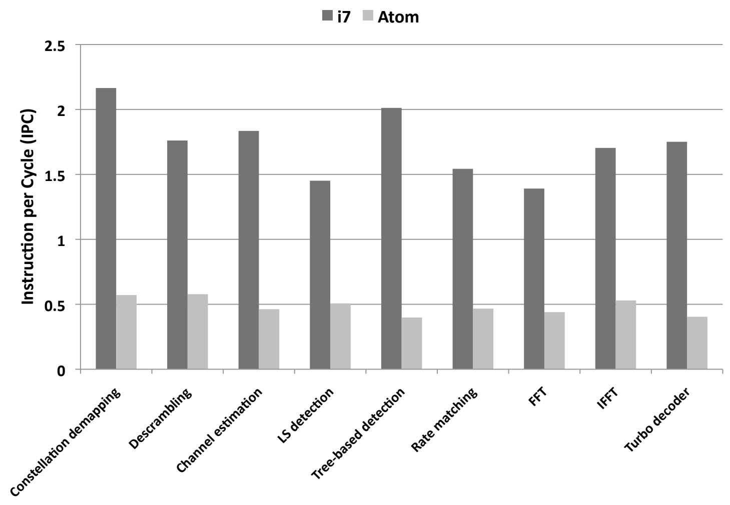
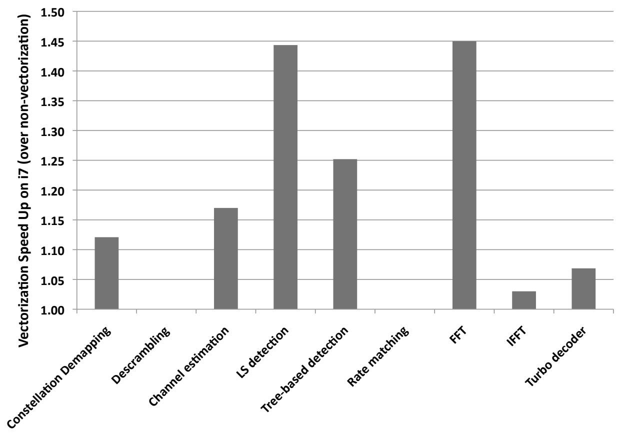
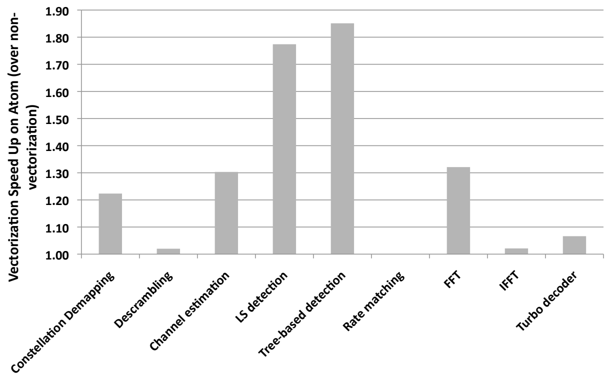


Figure 3.6 IPCs for desktop and embedded processors. The IPCs of kernels on the i7 processor are higher than those on the Atom processor even taking the issue width difference into account. Because the i7 is an out-of-order processor, it can dynamically schedule instructions and take advantage instruction and memory level parallelism.

For the server class processor, an Intel Core i7-2600 CPU running Linux 3.2.0-38-generic was used. For the embedded system, an NVIDIA ION box with an Intel Atom 330 processor and 4 GB of SDRAM was deployed. The Intel Atom is the Intel's line of low-power, low-cost microprocessors [10], whose SoC platform is used in many smartphones and tablets such as Lenovo K800, Motorola RAZR i, Safaricom Yolo, Samsung Series 5 Slate, and HP ElitePad 900 [3, 14]. The detailed configuration of the systems are presented in Table 3.2. The benchmarks were compiled using GNU g++ compiler suite version 4.6.3 with O2-level optimization. Intel VTune Amplier XE 2013 was used to gather code hotspot information and instructions per cycle (IPC) for the wireless benchmarks. VTune Amplifier XE is a performance profiler provided by Intel for x86 based processors. It provides information on code performance, including the hotspots, CPU utilization, multithread synchronization overhead, etc.



(a) i7 processor



(b) Atom processor

Figure 3.7 Vectorization Impact on (a) i7 and (b) Atom for configurations in Table 3.3. These graphs show speedups achieved when kernels were compiled with automatic vectorization flags turned on. The results suggest that hardware platforms should include vectorization support when running these kernels.

3.3.2 Individual kernel characterization

I firstly analyze the performance of each individual kernel in WiBench. Table 3.3 describes the configurations of every kernel used in this study.

Figure 3.6 compares IPCs when running WiBench kernels on the two platforms. The IPC values demonstrate the raw processor performance of signal processing kernels on different hardware platforms. Based on Figure 3.6, the i7 processor, with dynamic out-of-order scheduling, can make use of ILP and memory level parallelism (MLP) in order to issue more instructions per cycle than the Atom processor, even taking the issue width difference into account. However, since out-of-order execution requires more complex hardware, leading to a high power consumption, this improvement must be balanced against the limited power budget of embedded platforms.

Since digital signal processing algorithms are usually abundant in DLP, next, I study the performance of using the SIMD extension in each processor to speedup the kernel performance. I used automatic vectorization flags (-ftree-vectorize -msse2 -ffast-math) during the compiling time to take advantage of the SIMD extension. The corresponding results are shown in Figure 3.7. When the compiler automatic vectorization is enabled, SIMD instructions are inserted automatically by the compiler to replay mainly for loops. By using automatic vectorization, I can get as much as 1.45$\times$ speedup on the i7 and 1.85$\times$ speedup on the Atom.

However, since vectorization is implemented by the compiler, there is a limited range over which it works. Therefore, I study the algorithms of each kernel, and

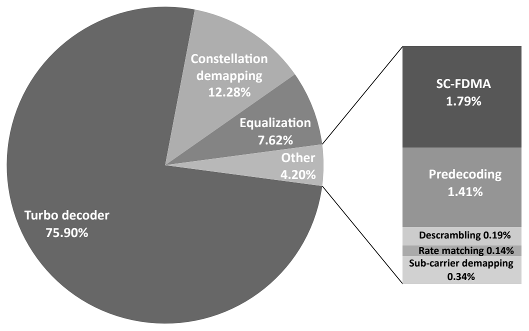
Table 3.4 The theoretical SIMD width of individual kernels for the configurations in Table 3.3

Kernel	SIMD width
Turbo decoder	8
Rate matching	1
Descrambling	300
Constellation demapping	600
LS detection	150
Tree-based detection	300
FFT	128
IFFT	75
Channel estimation	300

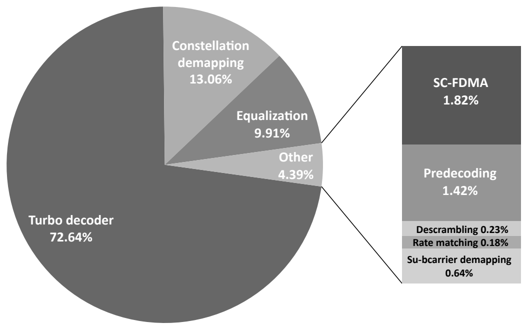
manually analyze the theoretical SIMD width of each kernel (shown in Table 3.4). All of the kernels do not achieve this speedup when using automatic vectorization. This is mainly because the for loops are written in the way that it is difficult for the compiler to extract DLP, such as there is plenty of memory aliasing. More speedup is expected if the program is vectorized manually using SIMD intrinsics. Overall, the results indicate that hardware platforms designed for these kernels should include SIMD-type support and that hand-optimized code/libraries will continue to be needed in order to attain better performance.

3.3.3 Application example: LTE uplink system

Secondly, I profile the LTE Uplink provided in the benchmark with respect to hotspots and runtime performance. Because most of the computations are done in the receiver side, I only profile kernels in the LTE uplink receiver. I perform four studies: 1) a characterization of one LTE uplink modulation configuration across different specification data rates; 2) a characterization of different LTE uplink modulation configurations for a fixed specification data rate; 3) an analysis on the sizes of data transfered between kernels; and, 4) a study of the BER for the LTE uplink under a Gaussian Random Channel model.



(a) i7 processor



(b) Atom processor

Figure 3.8 Breakdowns of the LTE uplink runtime among the kernels on (a) i7 and (b) Atom. The results indicate that hardware designers should put much concern on expediting the Turbo decoder. It should be either highly optimized for the specific platform or implemented by a hardware accelerator.

Table 3.5 The configurations of the LTE uplink at 100Mbps

Kernel	Configuration
Turbo decoder	code rate $= 1/3$, codeword length $= 6144$
Constellation demapping	16QAM
FFT	2048
IFFT	1200
MIMO	2×2

3.3.3.1 LTE uplink characterization for different specification data rates

At first, I study the breakdown of runtime for the LTE uplink to determine the computational hotspots. The configuration with 100Mbps specification data rate is used in this hotspot study, and is specified in Table 3.5. Figure 3.8 shows the time spent by each kernel as a fraction of the overall system runtime for both the i7 and the Atom platforms. As we can see from the results, the Turbo decoder takes more than 70% of the execution time, indicating that it is the dominant kernel in the LTE uplink. Thus, to achieve high throughput applications, the Turbo decoder should be either highly optimized in the software for the specific platform or be implemented as a hardware accelerator.

In addition, I evaluate the total runtime of the LTE uplink with different subframe sizes, which illustrates the performance of the LTE uplink system as the workload changes. Figure 3.9 demonstrates that the processing time of an LTE uplink subframe increases proportionally to the subframe size. Since the size of a

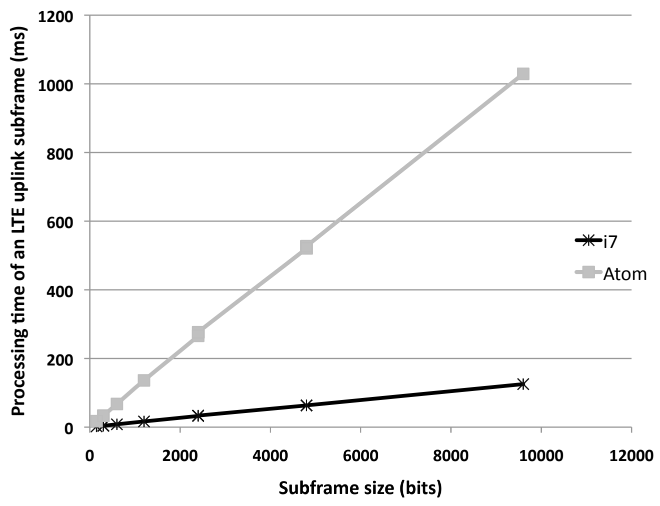
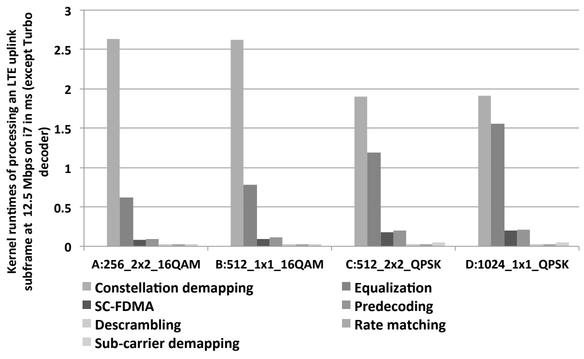


Figure 3.9 Processing times of an LTE uplink subframe with different subframe sizes. The larger the subframe size, the higher the specification data rate. It shows that the processing time of an LTE uplink subframe is proportional to the subframe size. This indicates a linear scaling of the dynamic operation count for most LTE uplink kernels.

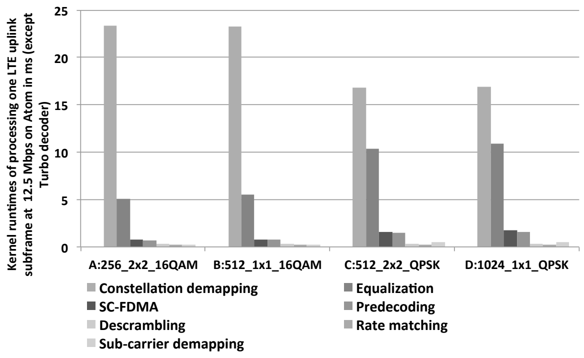
subframe is proportional to the system specification data rate, the processing time of an LTE uplink subframe is therefore proportional to the system specification data rate. This indicates that the dynamic operation count for most LTE uplink kernels scale linearly.

Table 3.6 The configurations of the LTE uplink at 12.5 Mbps

Configuration	FFT	IFFT	MIMO	Constellation Demapping
A	256	150	2×2	16QAM
B	512	300	1×1	16QAM
C	512	300	2×2	QPSK
D	1024	600	1×1	QPSK



(a) i7 processor



(b) Atom processor

Figure 3.10 The physical layer kernel runtimes in LTE uplink with different configurations at 12.5 Mbps on (a) i7 and (b) Atom processor. The results suggest optimizations on the constellation demapping and equalization, and show the kernel importance change as the system configuration changes.

3.3.3.2 Different LTE configurations with the same specification data rate

To study the influence of the LTE uplink configuration on the computational feature, I look into the runtime changes of each individual kernel for different LTE uplink configurations. For this study, the LTE uplink specification data rate is fixed at 12.5 Mbps. While this data rate can be achieved with many different configurations, I choose four representative configurations, presented in Table 3.6.

These configurations differ in the size of the OFDM symbol, number of antennas and constellation size. For instance, a large FFT configuration with simpler constellation can be used for bad channel conditions with larger bandwidth usage, while a small FFT configuration with complex constellation and more antennae can be used for good channel condition but limited bandwidth. The Turbo decoder is assumed running on a specialized accelerator and its runtime is excluded. This is a reasonable assumption because these accelerators are typical even in programmable wireless signal processors. Figure 3.10 shows the results. From the figures, the following conclusions can be derived.

- The constellation demapping and equalization kernels take most of the execution time (when excluding Turbo) for all four configurations. Therefore, hardware and software optimizations should be done to accelerate these two kernels.

- The importance of each kernel changes as the system configuration changes, even if the data rate remains the same. While constellation demapping is much more important than all the other kernels, equalization, FFT and IFFT are also important for Configuration D.

3.3.3.3 Data transfer between kernels

Memory system design is an important part for a computing system. In this study, I look at how much data is transfered between different kernels. Figure 3.11 demonstrates the data movement between kernels when processing one LTE subframe for the configuration in Table 3.5. The values in the red circles represent the amount of data movement, which indicates the minimum sizes needed for the buffers containing the intermediate results between adjacent kernels. Because on-chip memory is an expensive resource, this information helps domain specific computer architects design their memory system.

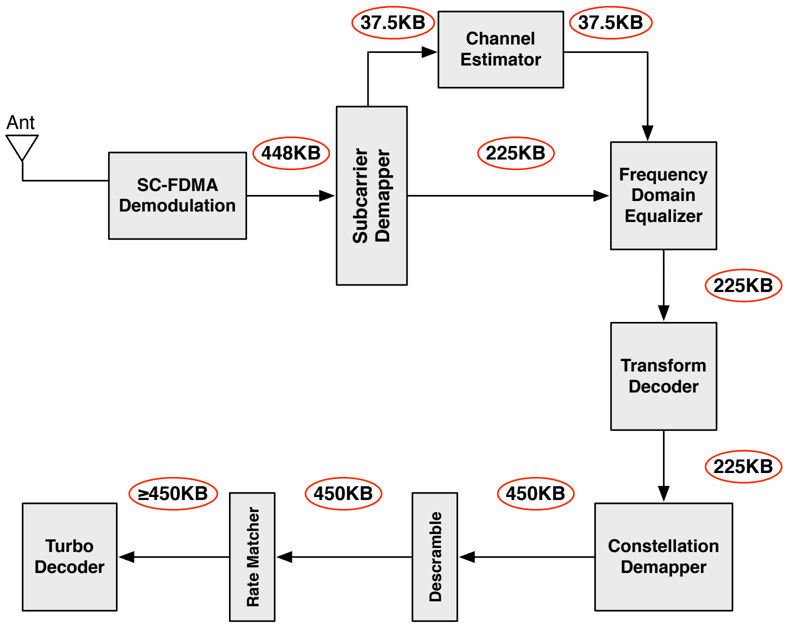


Figure 3.11 The sizes of data movement between kernels. The results show how much data needs to be stored in the buffers for the intermediate results between adjacent kernels to process one LTE subframe for the configuration in Table 3.5.

3.3.3.4 Exploring channel models and BER

In this section, I study the BER performance of the LTE uplink system under a Gaussian random channel (the amplitude follows a Rayleigh distribution) with additive white Gaussian noise (AWGN). This is an example to show how system designers can connect kernels through a channel model, and inject noise to measure BER of an LTE uplink. The kernel configurations are the same as those in Table 3.3. The BER performance is shown in Figure 3.12. BER is calculated by collecting the difference between the information bits encoded in the transmitter and those decoded at the receiver end. Perfect CSI means that the receiver knows the exact channel impulse response when processing received data. The FD LS curve expresses the performance of a system running with a frequency domain least square channel estimator, which is a more realistic scenario.

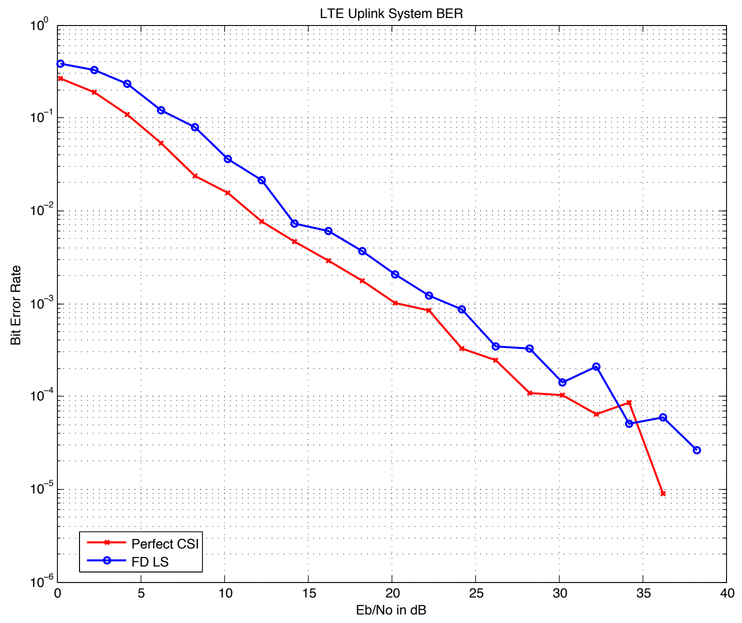


Figure 3.12 BER of LTE uplink through Gaussian random channel with AWGN. This graph shows the BER performance of the LTE uplink system under a Gaussian random channel with AWGN.

3.3.4 Architectural implications from the WiBench characterization

Based on the WiBench characterization study, I come up with the following implications on the computing system design for wireless signal processing:

- The signal processing kernels in a wireless communication system are very typical streaming applications, which are computation intensive with little data reuse. This indicates that the traditional cache based memory hierarchy may not be the most effective solution for wireless systems.

- As shown, the Turbo decoder is the most important kernel in an LTE uplink system. Thus a processor designed for an LTE system should have architecture support to efficiently execute the Turbo decoder. Since the Turbo decoder has a small SIMD width and little speedup with automatic vectorization, it should be mapped to a hardware accelerator—which is often the case in today's practice.

- Constellation demapping and equalization (consisting of MIMO detection and channel estimation) have very large theoretical SIMD widths (from Table 3.4), and also achieve appreciable speedups with automatic vectorization by the compiler. Therefore, a wide SIMD-type engine should be included in the processor to accelerate these two kernels.

- In addition, most kernels have very few branches and little control code. Most of branches in these kernels are from the for loops, whose iterations are pre-determined based on the wireless system configuration. This indicates that a complicated branch predictor is not needed in the processors for wireless signal processing.

- Finally, I noted that the importance of each kernel varies when system configurations are different. All together, these observations illustrate the usefulness of a benchmarking infrastructure to evaluate wireless signal processing systems.

3.4 Related Work

The wireless communication community works on open problems in telecommunication as well as next generation technologies. Their primary focus is on the impact of communication theory and algorithm optimization on system performance, typically measured in terms of BER. There are many open source system simulators written in MATLAB or built through Simulink [13, 65, 12] to aid in this analysis. MATLAB and Simulink are easy to use due to their interactive natures and a large number of built-in functions. However, MATLAB and Simulink are not suitable for hardware design because their abstraction levels are too high. To aid in the co-design of systems and their underlying architecture, I release both MATLAB and C++ versions of all kernels in WiBench—System designers may explore BER through MATLAB, and architects can explore power-efficient hardware organizations that execute the C++ code. While most system simulators are in MATLAB/Simulink, there are several that include C/C++ versions which will be discussed in the following paragraphs.

First, the closest related work to WiBench is GNU Radio [9], a free and open source software toolkit providing signal processing blocks for software radio implementation. GNU Radio uses a “block” abstraction to connect signal processing kernels, which are implemented in C++, together with a few lines of Python code. Each block is equipped with its own input/output buffers. The GNU Radio suite then uses a runtime scheduler that activates each block when there is enough data in its input buffer and space in its output buffer to perform the function. It is designed to run on commodity hardware. To construct a complete end-to-end wireless system, the user must first understand algorithmic details of these blocks. WiBench has a different goal, which is to support hardware exploration of domain specific hardware solutions. To this end, I provide all key kernels as well as the entire system in WiBench. In addition, the GNU radio block class introduces non-kernel overheads. This may lead to a distorted picture of how the signal processing algorithms would perform on domain specific hardware. WiBench’s behavior is closer to the actual computational characteristics of wireless signal processing kernels, similar to the approach used in the design of several high-performance DSP prototypes [86, 111, 112].

Second, MiBench [67] is a set of embedded applications released over a decade ago. Telecommunication, one of the six categories in the benchmark, contains GSM related processing—FFT/IFFT, GSM voice encoding and decoding algorithms, Adaptive Differential Pulse Code Modulation encode/decode, and CRC32 checksum algorithm. These represent only a small portion of wireless signal processing kernels. Since the release of MiBench in 2001, communication technology has seen rapid development including several generations of technology enhancements rendering many of the MiBench kernels irrelevant. WiBench is designed specifically for wireless signal processing and includes many state-of-the-art algorithms that will be used in next generation technology.

Third, LTE Uplink Receiver PHY Benchmark [101] is an open source, freely available benchmark that represents the baseband processing of an LTE base station. The benchmark implements SC-FDMA modulation, channel estimation, transform decoding and soft symbol demapping, and is capable of generating different number of users with different workloads. However, this benchmark mainly aims to simulate the workload change in an LTE base station to study the power management strategy, rather than the characterization of wireless algorithms for hardware design. In addition, it only includes some parts of the LTE uplink and is missing the details of several important kernels, for example the Turbo decoder is represented simply as a sleep function. Ultimately this limits the use of this benchmark in a wider scope of wireless system design. The WiBench contains all the signal processing kernels for LTE in both MATLAB and C++ versions.

Finally, the BDTI^{TM} OFDM receiver benchmark [4] is a commercial benchmark for evaluating multi-core and other high-performance processing engines for communication applications. Public information about this benchmark is limited, but their website does indicate that they still use the Viterbi decoder rather than the more state-of-the-art Turbo decoder present in WiBench. The BDTI^{TM} OFDM receiver benchmark requires a license for use, in contrast to WiBench.

3.5 Summary

As the mobile market continues to grow rapidly wireless signal processing is becoming one of the primary uses of computing technology. Consequently, closer attention is being paid to the hardware platform design and its energy efficiency. Computer architects usually benefit a great deal from analyzing application benchmarks during design time to gain insight into energy and performance tradeoffs. In this chapter, I presented an open source benchmark suite of wireless system kernels and channel models to support hardware and system design of wireless signal processing platforms. I characterized the benchmark suite on two different types of processors to illustrate the computational features of the wireless communication system. Users can easily build their own wireless systems by simply assembling the kernels together to realize a target configuration.

CHAPTER IV

Implementing Wireless Baseband on General-Purpose GPUs

Based on the characterization study in Chapter III, a general-purpose processor that can make use of abundant DLP in the wireless baseband signal processing algorithms and provide high throughputs is an ideal candidate for the future C-RAN datacenter. In addition, due to the large number of concurrent service requests from many users in the wireless network, there is also a large amount of TLP in a C-RAN datacenter. These make both traditional server based CPUs and newly developed GPUs promising candidate processors for the future C-RAN datacenter.

However, one challenge of building the C-RAN datacenter is to realize high performance software implementation of the baseband signal processing, which was traditionally implemented in the hardware. Therefore, I study how to achieve high performance software-based baseband signal processing for both CPUs and GPUs, through exploring each of their architectural features.

For the CPU implementation, I start with optimizing the WiBench C++ code. Better floating-point functional units, multi-core parallelism and SIMD extensions make multi-core CPUs more capable of processing wireless signals than ever before. As the multi-core CPU platform, I use a state-of-the-art Intel Xeon server processor with 32 hardware threads and SSE/AVX SIMD extensions. To maximize the CPU performance, I extracted parallelism in the WiBench C++ code by using automatic vectorization and openMP optimizations. OpenMP provides APIs that implement multithreading across most processor platforms. By inserting OpenMP pragmas around the code that can be parallelized, multiple concurrent and independent threads are created. This can take advantage of the multi-core feature of the Intel Xeon processor, improving the overall throughput of CPUs.

A more challenging work is to realize high performance software implementation on GPUs. This is due to the highly parallel processor architecture and long memory access latencies that GPUs have. Therefore, in this chapter, I mainly focus on techniques to achieve highly parallelized implementations of key wireless baseband kernels on GPUs.

4.1 Overview

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 [92]. 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 [106], 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 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 [106, 58, 5] 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.

In this chapter, I will explore building an LTE base station with GPUs. These processors provide GFLOPs/TFLOPs-level throughput, and have high compute capability per Joule [73]. GPUs also have added language support like CUDA for general-purpose programming. They provide programmers the ability to exploit high degrees of DLP and 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.

I will demonstrate how the digital baseband system for an LTE base station can be built with commercial GPUs. Firstly, I show the parallelization techniques for trellis algorithms, a family of algorithms whose processing can be represented as the value propagation in a trellis. Since the Turbo decoding algorithm is an important member of the trellis algorithm, these parallelization techniques can be implemented with the Turbo decoder on GPUs. Second, I parallelize the physical layer kernels by exploring different types of parallelism to maximize the GPU performance. Then I study kernels' runtime performance under different antenna configurations and modulation schemes when implemented on NVIDIA GPUs, and explore a multi-GPU configuration for high data rate applications. Finally, I 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 the dual-GPU based LTE base station consumes 188 W, which is competitive with commercial systems.

The rest of this chapter is organized as follows. Section 4.2 introduces the performance challenges when implementing applications on GPUs. Section 4.3 introduces baseband processing in an LTE base station. The GPU parallelization techniques of the trellis algorithm and key physical layer kernels are described in Section 4.4 and Section 4.5, respectively. Section 4.6 introduces different multi-

GPU systems, and different kernel mapping methods onto a multi-GPU system. Section 4.7 shows the GPU performance, the minimum number of needed GPU under different system configurations, and the power consumption. Related work is discussed in section 4.8 and the paper is concluded in section 4.9.

4.2 Performance challenges on a GPU

A GPU is a programmable processor providing GFLOPs/TFLOPs-level throughput, which makes it a good platform for computation intensive applications. However, it is a challenge to implement an algorithm that makes full use of the GPU resources. There are two main causes of underutilization: pipeline stall and thread inadequacy. Pipeline stall occurs when dispatch units fail to issue an instruction, mainly due to long memory access. Thread inadequacy happens if the number of thread blocks is smaller than that of SMs, or the number of threads in each thread block is not a multiple of thread context size, 32 for the case. To keep all the cores of a GPU active, an adequate amount of workload must be created. The parallelization schemes proposed here help create such a workload.

The memory system consists of on-chip memory, off-chip L2 cache and external memory. On-chip memory can be configured as either $48KB/16KB$ or $16KB/48KB$ shared memory/L1 cache. Shared memory is software managed, so when shared memory usage per thread is fixed, more shared memory leads to more threads and the GPU is better utilized. However, this also leads to smaller L1 cache and thus longer access time.

4.3 Baseband processing in an LTE base station

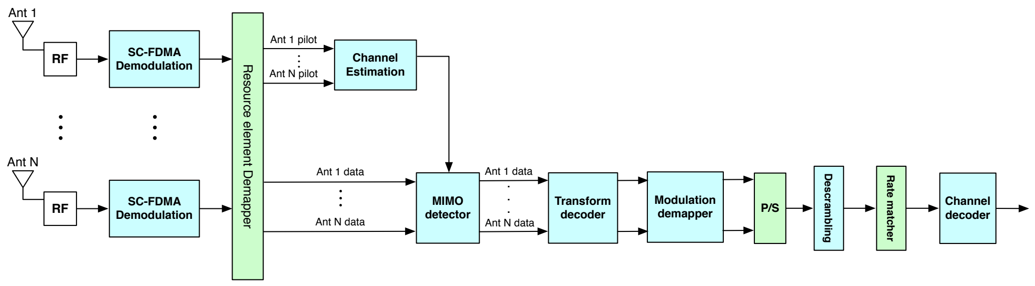


Figure 4.1 Baseband processing of the receiver in an LTE base station

The main baseband processing kernels in an LTE base station receiver are shown in Fig 4.1. LTE uplink uses SC-FDMA for transmission [57]. 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. The received data from the channel is first processed through SC-FDMA FFT. Pilot signals are used to estimate the CSI, which is then used in the MIMO detector to counteract the effects of the channel. The transform decoder performs IDFT 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. I give a brief description of the key kernels below.

SC-FDMA: SC-FDMA is a precoded 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 IDFT. The largest size of OFDM FFT is 2048, and that of transform decoding IDFT 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. I implemented frequency domain least square channel estimation [45].

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 MMSE equalizer, is usually used. I used the MMSE-based MIMO detector in the 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 [98]. The binary sequences are grouped and mapped into complex-valued constellation symbols. The modulation demapper, on the 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. I implemented a soft decision modulation demapper.

Turbo decoder: Turbo codes are used for channel coding in LTE. The Turbo decoder architecture includes two 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 the GPU implementation, I set the number of iterations to be 5.

4.4 Parallelizing trellis algorithms on GPUs

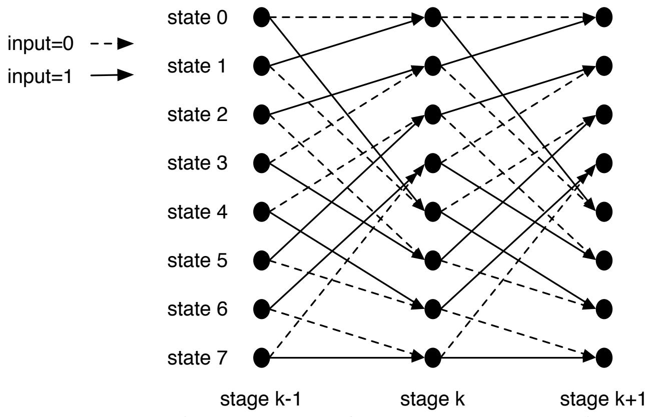


Figure 4.2 Trellis structure of Turbo codes used in LTE

4.4.1 Introduction of trellis algorithms

The trellis is a widely used graph in coding theory that describes the progression of symbols within a code. There are many popular trellis algorithms, including Viterbi algorithm [55], Baum-Welch algorithm [33], Turbo decoding algorithm [31], etc. These algorithms are used in many systems, such as in speech recognition, communication protocols and data compression. In order to meet the timing deadlines of such systems, high throughput implementations of trellis algorithms are required. For example, the Viterbi algorithm that is used as the convolutional code in the WCDMA wireless protocol requires a 2Mbps decoding throughput in the downlink.

The trellis is a graph representation of the state transitions of an FSM for all possible input sequences. Fig. 4.2 is a typical structure of a trellis. Each column is a unit of time called a stage and each node represents a possible FSM state at each stage. A branch between two states corresponds to a possible state transition, depending on the input to the FSM. In the forward direction, the forward metric of a state s_k at stage k is the maximum (over all possible transitions from s_{k-1} to s_k) of the sum of the forward metric of states s_{k-1} and the branch metric corresponding to the transition from s_{k-1} to s_k. Similarly for the backward direction. After computing the forward and backward state metrics, the metric of each possible transition at stage k is evaluated as the sum of the forward metric of the starting state s_{k-1}, the branch metric corresponding to the transition from s_{k-1} to s_k and the backward state metric of s_k. Finally at each stage k, the metric corresponding to each input bit is evaluated as the maximum among all transition metrics corresponding to the same input bit. In Fig. 4.2, for example, the metric for bit 0 is the maximum metric of eight transition metrics represented with dashed lines.

4.4.2 Parallelization techniques

As explained in Section 4.4.1, trellis algorithms have the inherit dependency between processing of adjacent stages, and are usually the hotspot of the applications due to their iterative fashions. In addition, tradeoffs between achieved throughput, worst case latency and bit error rate are required when implementing trellis algorithms. Therefore, it is critical to explore different parallelization techniques to achieve the best tradeoff. In this work, I consider three levels of parallelism: packet-level, subblock-level and trellis-level.

4.4.2.1 Packet-level Parallelism

A packet is a formatted unit of data in a computer or communication network. In a GPU implementation, the input packets can be stored in a buffer so that they can be processed in parallel. The disadvantage of packet-level parallelism is that it results in long latency especially for the first packet in the buffer. This impairs the quality of service of time-constrained applications. The number of threads in a packet-level parallelism scheme is proportional to the number of packets that are processed in parallel.

4.4.2.2 Subblock-level Parallelism

A packet can be divided into several subblocks, which are processed in parallel. While this increases the number of threads, it leads to higher bit and packet error rates since the computations in each of the subblocks are not really independent from each other. Specifically, the computation of the ith subblock depends on the computations in the last stage of the $(i-1)$th subblock. Thus, if subblocks are processed in parallel, the initial values of latter subblocks are incorrect resulting in higher output error rate. One way to compensate for this performance loss is by employing recovery algorithms, e.g., training sequence (TS) and next iteration initialization (NII) [116].

In the TS algorithm, additional computations are done on the $(i-1)$th subblock to generate the dummy initial values of the ith subblock. The longer the training sequence, the larger is the number of additional computations and lower is the BER.

In the NII algorithm, the outputs of the $(i-1)$th subblock in the previous iteration are used as the initial values of the ith subblock in the current iteration. The idea behind NII is that the results of each iteration converge closer to the correct values than those of previous iterations. From an implementation perspective, TS requires additional operations, and NII needs additional memory.

4.4.2.3 Trellis-level Parallelism

There are three types of trellis-level parallelism. The first is state-level parallelism, in which the nodes in a stage is processed in parallel. There are no computational dependencies among the nodes in a stage and the processing of a node only depends on the nodes that are connected to it in the adjacent stages. State-level parallelism does not affect BER, and the number of threads due to state-level parallelism is proportional to the number of states in a stage.

The second type of parallelism is forward-backward traversal where the values are propagated in both forward and backward directions, and the propagations are independent. Forward-backward traversal (FB) results in more complex index and memory address computations, because two propagations must be separated

Table 4.1 Summary of parallelization schemes

Scheme	Throughput	Latency	Bit Error Rate
Packet-level	Better	Worse	No_Change
Subblock-level	Better	No_Change	Worse
Trellis-level	Better	No_Change	No_Change
Subblock+NII	Worse	No_Change	Better
Subblock+TS	Worse	No_Change	Better

during the calculation. Therefore, more instructions are executed to support FB parallelism, thereby lowering the throughput.

The third type of parallelism is branch-metric parallelism (BM), where the branches from a node in stage k to others in stage $k+1$ are processed in parallel. This is not as effective since the vector reduction parts cannot be parallelized. However for higher radix trellis that is obtained by combining multiple stages together, more threads can be generated from BM. Also less memory is used in this case. Overall, two threads are created in FB and the number of threads created by BM is equal to the radix degree.

Table 4.1 summarizes the parallelization schemes. From this table, we see that trellis-level parallelism improves throughput without impairing latency and BER. Packet-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.

4.4.3 Exploring parallelization tradeoffs

4.4.3.1 Experimental framework

I selected a representative trellis algorithm, the Turbo decoding algorithm [31], and an NVIDIA GTX470 GPU to evaluate the performance tradeoff of the different parallelization schemes. The NVIDIA GTX470 GPU is based on Fermi architecture [96]. It can support at most 448 threads running at a time. It has a 64KB on-chip memory, a 768KB L2 cache and 1280MB external memory. I chose the Turbo decoding algorithm as a case study since it is used in the Turbo code in LTE. The corresponding trellis structure has 8 states in a stage, and is the same as shown in Fig. 4.2; however, the values propagate through the trellis in both directions. The LTE Turbo code configuration is used in the simulations: the packet size is 6144 bits and the code rate is 1/3 [24]. The baseline implementation is a sequential one (without any parallelization) with 0.0178 Mbps throughput and 345ms packet latency.

4.4.3.2 Performance of different parallelization techniques

I implemented two GPU memory configurations ($48KB/16KB$ and $16KB/48KB$ shared memory/L1 cache) with state-level, subblock-level and packet-level parallelism. I varied the number of subblocks from 1 to 512, and the number of packets from 1 to 84. I found that a larger L1 cache results in better timing performance. For instance, 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, I used the 48K L1 cache configuration in the rest of the experiments.

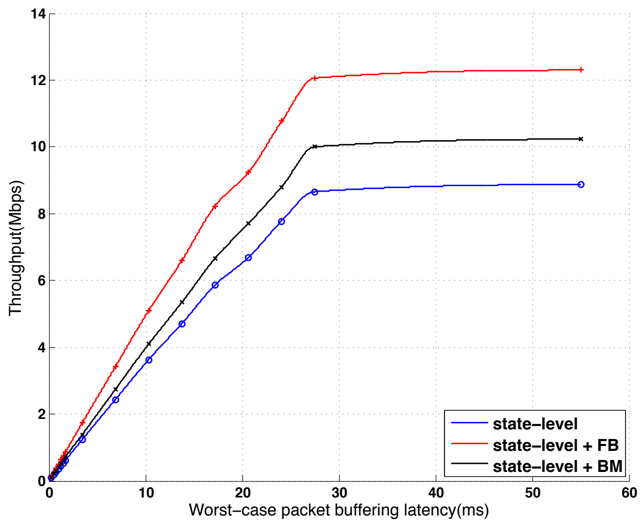


Figure 4.3 Throughputs and latencies of different trellis-level parallelization schemes when the packet size is the same as the subblock size and multiple packets are processed in parallel

First, I studied the performance of different trellis-level parallelism schemes for different packet latencies. I use packet buffering latency as a metric to represent packet-level parallelism since it is a function of the number of packets being processed in parallel. Fig. 4.3 shows the performance of the different schemes when the subblock size is the same as the packet size. As the number of packets increases, the throughput increases. However, the throughput gains slow down when the number of packets is quite large. This is because the GPU is fully loaded and having more threads is not beneficial any more. The throughput improves quite a bit when state-level parallelism is combined with either FB or BM parallelism. Compared with the baseline scheme, state-level, state-level+FB and state-level+BM achieve a speedup of 5.1x, 7.4x and 5.8x, respectively. All schemes achieve a BER of 10^{-5} when the signal-to-noise ratio (SNR) is 1.0 dB.

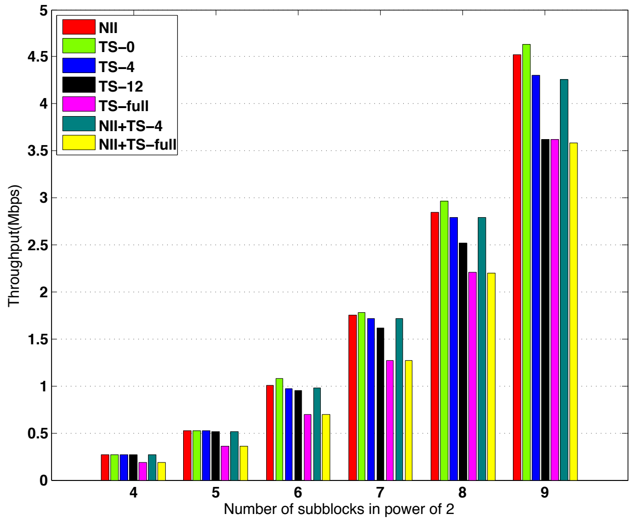


Figure 4.4 Throughputs of schemes with different number of subblocks (per packet) and recovery schemes for bit error rate of 10^{-5}

Next, I fixed the latency by considering only one packet and studied the effect of different numbers of subblocks and recovery schemes such as TS and NII. Fig. 4.4 and Fig. 4.5 show the throughput and SNR requirement for the different schemes. The length of a packet is 6144 bits, which equals the product of the subblock length and the number of subblocks. The SNR requirement presented here is the lowest value to achieve the given bit error rate of 10^{-5}. The SNR requirement of the baseline scheme is 0.9 dB. From this figure, I derive the following conclusions. 1) Increasing the number of subblocks provides higher throughput due to more parallelism, but has higher SNR requirement due to wrong initial values and shorter subblocks. 2) Longer training sequences have lower SNR requirement but lower

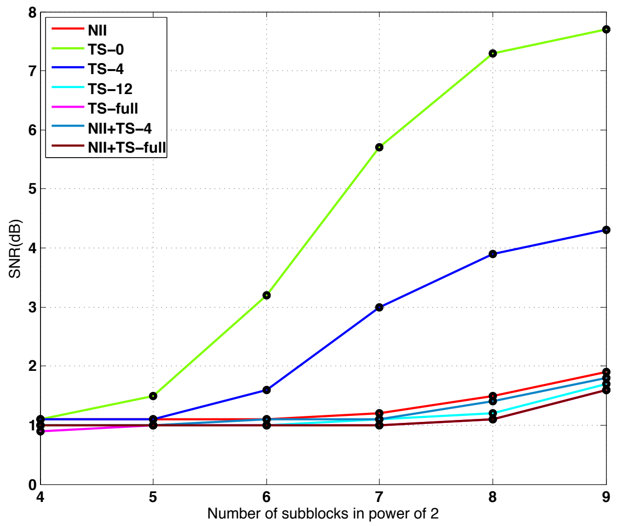


Figure 4.5 SNR requirement of schemes with different number of subblocks (per packet) and recovery schemes for bit error rate of 10^{-5}

throughput due to additional calculations. The SNR requirement saturates when the training sequence is long. For instance, TS-12 has almost the same SNR requirement as TS-full in which the training sequence is as long as a subblock. Additional computational overhead due to recovery schemes does not affect the throughput as much because of the high computational power provided by GPU. 3) Among the recovery schemes, the combination of NII and TS is the best. The scheme NII+TS-4 has nearly the lowest SNR requirement with a throughput of 4.26 Mbps when 512 subblocks are used per packet. Its throughput is comparable with that of NII or TS-4, but it has a lower SNR requirement.

I also study the effect of increasing the radix of the trellis algorithm. Radix-4, which is derived by combining two stages into one, helps to double threads from BM compared with radix-2, and reduces the required memory because there is only half the number of stages. However, since twice the number of threads are generated, the amount of work each thread undertakes is still the same and the total amount of work done in a radix-4 implementation is two times that of radix-2. So, radix-4 is useful only when GPU is not fully loaded and the benefits from compactness outperform the overhead of redundancy. The experiment shows that radix-4 outperforms radix-2 when the packet number ≤ 4.

Table 4.2 Parallelization tradeoff

Schemes			TH*	WPL*	SNR*	BER*
TL+	Subblock Num	Packet Num	(Mbps)	(ms)	(dB)	
-	512	1	4.26	1.44	1.7	1.6×10^{-3}
SL+	512	1	20.49	0.55	1.7	1.6×10^{-3}
SL	256	2	21.09	1.07	1.3	4.1×10^{-4}
SL,FB+	256	1	19.65	0.56	1.3	4.1×10^{-4}
SL,FB	128	10	29.00	4.58	1.1	2.0×10^{-4}

* TH = Throughput, WPL = Worst-case Packet Latency, SNR = Signal-to-Noise Ratio requirement, which is the lowest value to achieve BER of 10^{-5}, BER = Bit Error Rate when SNR = 1.0 dB
+ TL = Trellis-level parallelism, SL = State-level parallelism, FB = Forward-Backward traversal

4.4.3.3 Parallelization tradeoff

Different systems have different requirements for throughput, latency and BER. For instance, real-time gaming requires low latency but medium throughput and BER; TCP-based service requires both low BER and high throughput but can tolerate long latency. In the study, I combined different schemes to determine which combinations of parallelism were suitable for which applications. First, I found that subblock parallelism with a combination of NII and short TS (NII+TS-4) achieves the best tradeoff between bit error rate and throughput. Next, we implemented NII+TS-4 with packet-level and trellis-level parallelization schemes to meet the 16.67Mbps LTE uplink throughput. Table 4.8 shows the throughput, latency, SNR requirement and BER of the different schemes. Note that trellis-level parallelism improves the throughput significantly and should be used at all times (row 2). If SNR requirement is low, subblock-level parallelism has to be used with caution and trellis-level and packet-level parallelisms are better options (rows 3 and 4). If the system has a rigid latency constraint, trellis-level or subblock-level parallelisms should be used to achieve high throughput with low latency (rows 2 and 4).

Table 4.3 Performance comparison

Work	GPU	Original Throughput (Mbps)	Scaled Throughput (Mbps)	BER*
[81]	Tesla C1060	2.1	3.77	1.0×10^{-2}
[116]	GeForce 9800	2.4	3.50	1.0×10^{-4}
[114]	GTX 470	27.5	27.5	Not known
This work	GTX 470	29.0	29.0	2.0×10^{-4}

* BER here is the bit error rate when SNR = 1.0 dB

Table 4.3 compares the performance of the techniques with other LTE Turbo decoder implementations on a GPU. For a fair comparison, I scaled the throughputs in [81] and [116] by the processor frequency and the number of processors in the GPU. The scheme achieves the best throughput with good BER. While [114] has comparable throughput, it requires processing 50 packets to achieve 27.5 Mbps. In comparison, the scheme needs to process 10 packets to achieve 29.0 Mbps throughput, resulting in significant reduction in the worst-case packet latency.

4.5 Parallelizing physical layer kernels on GPUs

State-of-art GPUs, such as NVIDIA GTX680, can launch thousands of threads at the same time. So an efficient implementation of kernels on a GPU involve 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 baseband signal processing that is done for each user data is independent from others after some initial joint processing at least. Therefore, 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 I assume the subcarriers are evenly distributed among all users. Each subcarrier in an SC-FDMA symbol of each user is independent, and can be calculated in parallel. The number of threads is the same as the number of subcarriers.

Algorithm-level Parallelism There is parallelism inherent in 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 show the parallelism of each physical layer kernel, I define the following:

- N_{FT} – 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, I 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 $N_{usr} \times N_{Rx} \times N_{sym} \times N_{FT}$.

In this study, I 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 $2^a \times 3^b \times 5^c \times 7^d$ [8]. I can employ all four levels of parallelism by using cuFFT: the FFT/IFFT implementation of cuFFT exploits the algorithm-level parallelism, and I make use of the other types of parallelism by batching multiple FFT/IFFT computations.

Channel estimation: I 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: I mapped an MMSE-based MIMO detector on the GPU. I 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, I 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, and 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.

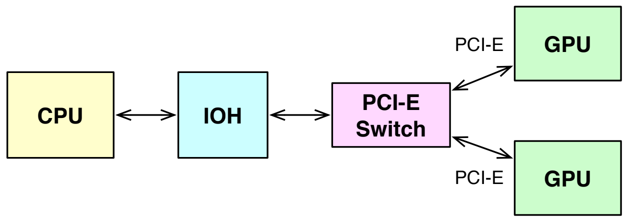
Table 4.4 summarizes the number of threads that can be created for each kernel. In the implementation using NVIDIA GTX680 GPU, I was able to generate all the threads, resulting in very high GPU utilization.

Table 4.4 Number of threads in PHY layer kernels

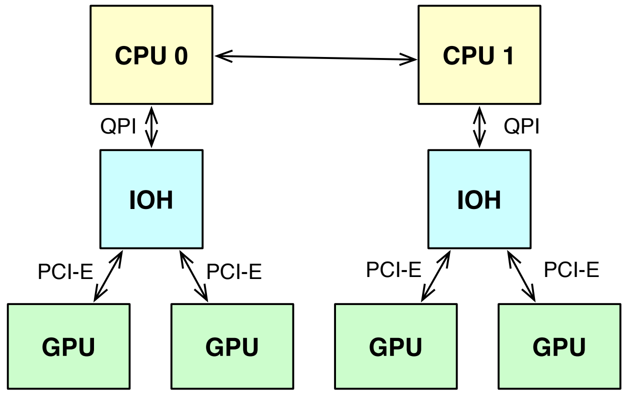
Kernel	Number of Threads
FFT/IFFT	$N_{usr} \times N_{Rx} \times N_{sym} \times N_{FT}$
Channel estimation	$N_{usr} \times N_{Rx} \times N_{sub}$
MIMO detector	$N_{usr} \times N_{sym} \times N_{sub}$
Modulation demapper	$N_{usr} \times N_{sym} \times N_{sub} \times N_{Rx} \times N_{Mod}$ $N_{usr} \times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

4.6 Mapping kernels onto 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.



(a) Connected through PCI-E switch



(b) Connected through IOH chips

Figure 4.6 Multiple GPUs within a single network node (MGSN)

4.6.1 Multi-GPU system

Employing multiple GPUs in an LTE base station can further speedup computation to help kernels meet the real-time deadline 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

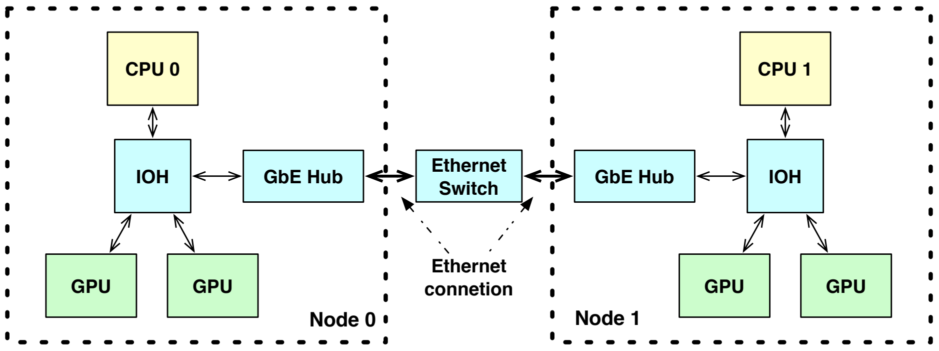
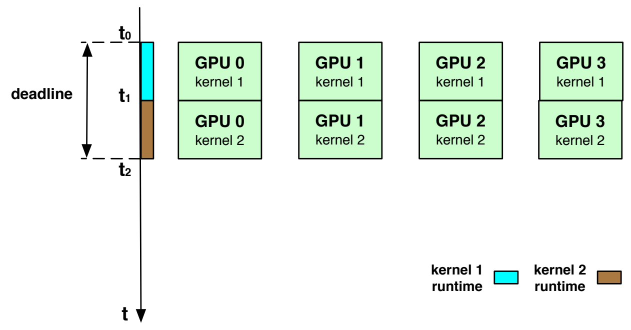


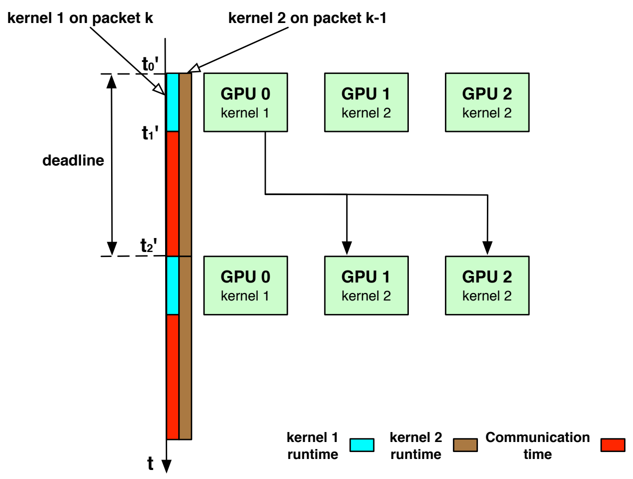
Figure 4.7 Multiple GPUs across multiple network nodes (MGMN)

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. Fig 4.6 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 4.6a), 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 4.6b), 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 [91]. Therefore, the data transfer between GPUs attached to different IOH chips is staged via CPU memory, which lowers the communication throughput.



(a) Sequential mapping



(b) Pipelined mapping

Figure 4.8 Mapping kernels onto a multi-GPU system

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 [20]. When more GPUs are required, multiple network nodes must be used, in which each node is an MGSN system. In an MGMN system, 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.

Table 4.5 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

I use the inter-GPU and CPU-GPU communication throughputs from [91] in this study. Table 4.5 summarizes the throughput numbers. The experiment configurations of Table 4.5 were not specified in [91], and it should be noted that these numbers may vary among different BIOS settings and IOH chips. For the inter-node connection, I 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 the study.

4.6.2 Mapping kernels on a multi-GPU system

Fig 4.8 shows two different ways mapping the LTE baseband kernels onto a multi-GPU system: sequential and pipelined.

Sequential mapping Fig 4.8a shows an example of the sequential mapping, in which kernel 1 and 2 are executed in sequence. All GPUs process kernel 1 in the 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 the deadline of the real-time system.

Pipelined mapping The other way to process kernel 1 and 2 is the pipelined mapping, which is shown in Fig 4.8b. During time slot $[t'_0, t'_1]$, GPU 0 processes kernel 1 on the packet k, while GPU 1 and 2 process kernel 2 on the previous packet $k-1$. Since kernel 2 takes a longer time to run, its execution also overlaps with the inter-kernel data transfer from GPU 0 to GPU 1 and 2. Then GPU 1 and 2 start working on the packet k, and at the same time GPU 0 processes the next packet. Because the processing of a packet is pipelined into multiple stages, only the runtime of each stage is required to be shorter than the real-time deadline.

Every mapping method has its advantages and disadvantages. The sequential mapping has a short overall processing time of each packet, because all kernels together must be done within the real-time deadline. In addition, there is usually no additional inter-GPU communication overhead. However, it requires more GPUs to accelerate the processing so that all kernels can be packed into one time slot of the real-time deadline. The pipelined method, on the other hand, needs fewer GPUs because of a looser timing requirement. It only requires each pipelined stage, either a kernel processing or a data transfer, to be done in a time slot of the deadline. The disadvantage of the pipelined method is that the overall processing time of each packet is longer, 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, I assume that a mix of the 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 used GPUs in the ith stage, P_i be the number of kernels running sequentially in the ith stage, $t_{run(i,j)}$ be the runtime of the kernel j in the ith stage, $t_{comm(i,i+1)}$ be the inter-GPU communication overhead between the ith and $(i+1)th$ stage, and $t_{deadline}$ be the real-time deadline. The mapping decision is to minimize the number of GPUs and the overall packet processing latency. This can be expressed as follows:

$$min(\sum_{i=1}^{S} N_i) \tag{4.1}$$

$$min(\sum_{i=1}^{S}\sum_{j=1}^{P_i} t_{run(i,j)} + \sum_{i=1}^{S} t_{comm(i,i+1)}) \tag{4.2}$$

and for the *ith* stage, there is a deadline to fulfill:

$$\sum_{j=1}^{P_i} t_{run(i,j)} + t_{comm(i,i+1)} \leq t_{deadline} \tag{4.3}$$

4.7 GPU performance evaluation

4.7.1 Experimental Environment

I used an NVIDIA GTX680 GPU to evaluate the performance of the key kernels. GTX680 is based on the Kepler architecture [95]. 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, I 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, I used a 2.13 GHz Intel Core 2 processor, running the Linux 3.2.0-39 generic operating system.

In this study, I simulated a fading channel with additive white Gaussian noise. I evaluated kernel implementation performance corresponding to peak data rate. I also focused on the single-user scenario because it gives a worst case estimate of the GPU performance, due to the lack of parallelism presented by multiple users. In this operating scenario, the computations in a base station depend on the total number of available subcarriers.

Table 4.6 The configurations of kernels

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	$1 \times 1, 2 \times 2, 4 \times 4$

Table 4.7 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

4.7.2 Kernel Runtimes

I ran each physical layer kernel for the different configurations shown in Table 4.6. The implementation of these kernels exploits the parallelism at multiple levels. For instance, in the 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 4.7 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.

Table 4.8 Performance of Turbo decoder implementations

Schemes			TH[1]	WPL[1]	BER[1,4]
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

For the Turbo decoder, I ran the same experiments of the tradeoff study, and tried to find the new best tradeoff on the new GTX680 GPU. Table 4.8 shows the performance of the Turbo decoder implementations. It demonstrates that the implementation in row 3 achieves the best tradeoff. I use this implementation in the rest of the paper.

4.7.3 GPU-based Wireless Baseband System

Table 4.9 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

A baseband signal processing processor must meet both the latency and throughput requirements of the communication protocol. LTE supports multiple data rates up to 300 Mbps. Table 4.9 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, to process a subframe of 1ms, the sum of runtimes of all kernels on PHY layer can be no larger than 1 ms. Based on runtimes presented in Table 4.7, 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, I assign the processing of subcarriers in a symbol onto multiple GPUs. Assuming that the subcarriers are allocated evenly among GPUs and that Turbo decoding is done by a separate set of GPUs, I estimate the minimum number of GPUs needed to meet the 1ms deadline under different system configurations. Table 4.10 shows the minimum number of GTX680 GPUs needed for PHY layer processing and the Turbo decoder. This analysis shows that for a 75 Mbps data rate, two GPUs are needed–one for processing the physical layer kernels and one for the Turbo decoder. When the peak data rate is higher, 100 Mbps, two GPUs are needed for the Turbo decoder. This is because a single GPU can only support Turbo decoder up to a 78 Mbps data rate. The GPUs are connected through PCI-Express on a board with low communication latency. A commercial motherboard can support up to four GPUs [103]. When more GPUs are required for higher data rates, multiple boards have to be interconnected through Ethernet, which leads to longer latency.

Table 4.10 The minimum number of GTX680 GPUs needed for 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	5	4	9

4.7.4 Power Consumption

Energy consumption a key metric when building a wireless network system. Therefore, I measured the GPU dynamic power consumption of the LTE kernels by using GPU-Z. Table 4.11 shows the power consumption of each kernel and the corresponding configuration. I 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 4.12. It shows that the Turbo decoder consumes most of the system energy followed by modulation demapper.

Table 4.11 Power of each kernel on a GTX680 GPU

Kernel	Configuration	Power (W)
Turbo decoder	Row 3 in Table 4.8	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

For a system-level power assessment, I considered the configuration corresponding to a 75 Mbps data rate. Based on Table 4.10, I need two GTX680 GPUs for a 75 Mbps data rate, one for the Turbo decoder and the other for the PHY layer. I 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. I compared it with the Alcatel-Lucent 9926 Base Band Unit [27] 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 I conservatively estimate that half of the power, 185 W, is consumed by the receiver, then the proposed GPU-based solution is still quite competitive.

4.8 Related Work

There have been several previous works that implemented the wireless baseband system on different hardware platforms.

GPU-based solutions: The GPU implementation of the transmitter in an LTE

Table 4.12 Energy consumption of each kernel processing 1 subframe at 75Mbps 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

base station was presented in [82]. In contrast, I 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 [58] 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 [26]. 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 [115] 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 [108] is the only fully programmable commercial base station to date. It is built with 4-13 Intel MPCBL0040 single board computers [74] 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.

4.9 Summary

In this chapter, I presented the work on implementing an LTE baseband signal processing system on commercial GPUs. Most kernels of the LTE baseband processing are highly parallel, and thus amenable to efficient GPU implementation. I firstly studied the parallelization techniques for trellis algorithms, a family of algorithms that is widely used. Since the Turbo decoder, the hotspot of the LTE uplink baseband system, is a member of the trellis algorithm, these techniques can be applied for the GPU implementation of the Turbo decoder. Then I implemented all the key kernels of an LTE baseband system on NVIDIA GTX680 GPUs, and evaluated their runtime performance. I showed that an LTE base station that supports a 75 Mbps peak uplink data rate can be built by using two GPUs. To support higher uplink data rates, more complex antennae and modulation schemes are needed. In these situations a dual-GPU solution is no longer sufficient. I 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.

CHAPTER V

The C-RAN Edge: future RAN on General Cloud Platform

In this chapter, I will study how to design general-purpose datacenters for C-RAN. Firstly, I will compare the performance, energy efficiency and total cost of ownership between the multi-core CPUs based C-RAN datacenters with general-purpose GPUs based ones. The results indicate that a GPU-based C-RAN datacenter outperforms a CPU-based solution. Therefore, I propose the C-RAN datacenter be built using GPUs as a server platform.

Next, I further study resource management techniques to handle the temporal and spatial traffic imbalance in a GPU-based C-RAN datacenter. I propose a "hill-climbing" power management that combines powering-off GPUs and dynamic voltage and frequency scaling (DVFS) to match the temporal C-RAN traffic pattern. For spatial traffic imbalance, I propose three workload distribution techniques to improve load balance and throughput.

Finally, I provide implications for future C-RAN datacenter designs, indicating that a C-RAN datacenter can benefit from architecture support for trellis algorithms in general-purpose processors, and the support for internet service at the wireless edge.

5.1 LTE C-RAN baseband unit

An LTE baseband unit contains the downlink transmitter and the uplink receiver. Since the receiver recovers the information from a noisy communication channel, it needs to estimate the channel conditions and iteratively decode the data. This means that most of the computations in an LTE baseband unit occur on the receiver side. Therefore, I focus on the baseband processing in the uplink receiver in this work.

Figure 5.1 shows the main signal processing components in an LTE C-RAN BBU. The LTE uplink uses the SC-FDMA scheme [57]. For every remote radio site, data received from all of its users is first converted into the digital signal by the RF module. Then the digital signal passes through baseband kernels as shown in Figure 5.1, and is recovered into binary information bits and fed to the core network. In addition, a C-RAN BBU executes the same process on data from multiple radio sites at the same time.

Table 5.1 summarizes algorithms and configurations for each C-RAN BBU kernel as being implemented. In the rest of this chapter, I use this implementation as a benchmark to design a complete datacenter for the C-RAN BBU uplink receiver.

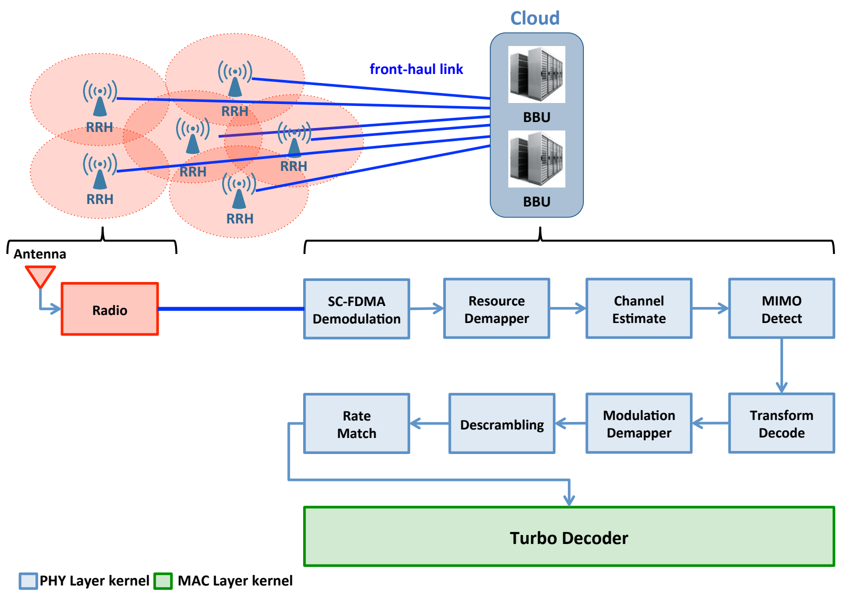


Figure 5.1 LTE baseband uplink receiver

5.2 C-RAN datacenter design

A C-RAN datacenter design is largely influenced by a push towards general-purpose processors and away from proprietary hardware platforms [41, 69]. The two major design spaces for general-purpose servers are multi-core CPUs and GPUs. In this section, I explore both design spaces and compare not only the performance, but energy and TCO to determine the best general-purpose platform for C-RAN datacenters.

(1) Multi-core CPU: Better floating-point functional units, multi-core parallelism, SIMD extensions, and large on-chip caches make CPUs more capable of processing wireless signals than ever before. In fact, Intel Labs China has already shown interest in using x86-based servers in C-RAN datacenters [63]. As the

Table 5.1 The algorithms and configurations of BBU kernels

Kernel	Algorithm	Configuration
SC-FDMA demodulation	FFT	2048, 1536, 1024, 512, 256, 128
Transform decode	Mixed-radix IFFT	1200, 900, 600, 300, 144, 72
Channel estimation	Least-square [46]	Pilot-based
MIMO detect	Minimum mean-square error equalizer [80]	1×1, 2×2, 4×4
Modulation demapper	Soft decision demapper [105]	QPSK, 16QAM
Turbo decoder	Max-log MAP [110]	1/3 code rate, 5 iterations

multi-core CPU platform in this study, I use a state-of-the-art Intel Xeon server with 32 hardware threads, SSE/AVX SIMD extensions, and 20MB L3 cache. To implement a C-RAN BBU, I modified the CPU implementation in Chapter IV with the extension to support multiple sites.

(2) GPU: GPUs are attractive platforms for C-RAN datacenters due to many vector and matrix operations and transform computations in BBU kernels that are well suited for the GPU architecture. Many works have already studied various implementations of wireless signal processing and LTE kernels on the GPU [117, 118, 62, 97, 30, 104]. As the GPU platform, I use NVIDIA Tesla K40 GPUs. The details of the CUDA implementation on GPUs are presented in Chapter IV.

Table 5.2 Description of CPU and GPU Test Platform

Unit	Multi-core CPU Platform	CPU+GPU Platform
CPUs	2× Xeon E5-2630 v3 @ 2.40GHz	2× Xeon E5-2620 v2 @ 2.10GHz
GPUs	-	1× NVIDIA Tesla K40 m
Caches	64KB L1, 256KB L2, 20MB L3	64KB L1, 256KB L2, 15MB L3
Memory	64GB DDR3 @ 2133 MHz	CPU: 16× 16GB DDR3 @ 1866 MHz
		GPU: 8× 12GB GDDR5, 288GB/s

Table 5.3 BBU Configurations

Configuration	LTE Specification Throughput *per site*
1×1 + QPSK	25 Mbps
1×1 + 16QAM	50 Mbps
2×2 + QPSK	50 Mbps
2×2 + 16QAM	100 Mbps

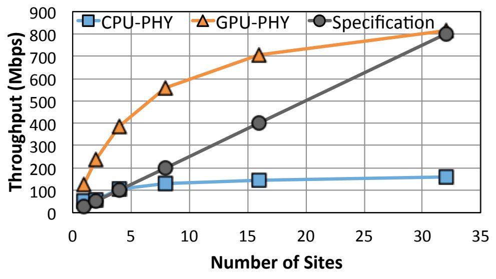
5.2.1 Experimental setup

In the study, I evaluate using the off-the-shelf CPU and GPU servers to build a C-RAN datacenter, whose information is summarized in Table 5.2. For the LTE BBU, Table 5.3 shows the four configurations that I evaluated with 1 and 2 antennae and with modulation schemes QPSK and 16QAM. The throughput expected by the LTE specification depends on the number of antennae and the modulation scheme. For each configuration, I varied the number of base stations (sites) between 1 and 32, and the specification throughput is directly proportional to the number of sites.

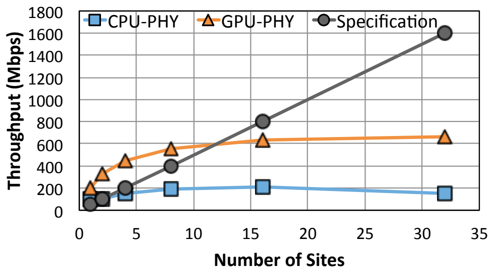
To simplify the experiment, I assume that the user data is already stored in the CPU and GPU memories, and I evaluate processing a sequence of LTE subframes that are loaded from the server memory. In addition, I assume that every user deploys the same modulation and antenna configuration, and the LTE subcarriers are evenly distributed among users. For the GPU implementation, a single host program manages multiple sites through a single CUDA stream.

5.2.2 C-RAN performance evaluation on CPUs and GPUs

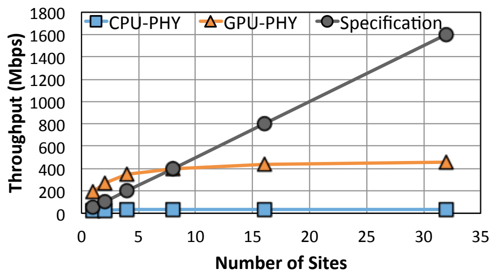
First I present the results of running the CPU and GPU implementations of the C-RAN BBU kernels.



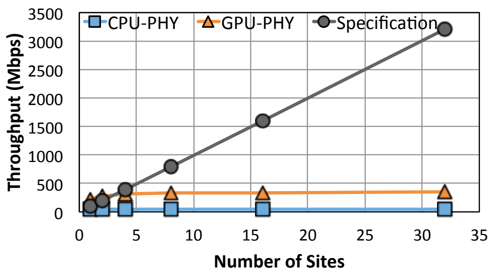
(a) 1×1 MIMO + QPSK



(b) 1×1 MIMO + 16QAM



(c) 2×2 MIMO + QPSK



(d) 2×2 MIMO + 16QAM

Figure 5.2 Throughput of the PHY layer kernels compared to the throughput specified by the LTE specification

Figure 5.2 shows the results of the throughput achieved in the PHY layer by both the CPU and GPU in comparison to the expected throughput of the LTE specification. From the graph we can see as the complexity of the algorithm increases from 1 antenna to 2 antennae and from QPSK to 16QAM, the LTE specification throughput also increases and it becomes harder for both the CPU and the GPU to meet. However, the GPU always demonstrates better achievable throughput across all configurations and any number of sites. This is because that the abundance of threads available on GPUs can make use of many types of parallelism present in the BBU kernels. On the whole, GPUs can take advantage of three key computational features common to all BBU kernels:

1. **Data level parallelism.** Most kernels consist of nested loops with independent iterations, showing a high degree of DLP. Transform and matrix operations are performed over large data sets stored in vectors, which are easily parallelized.

2. **Low branch divergence.** Kernels have simple control flow and consist mostly of loop iterations. Control flow is data independent and data accesses are regular and predictable.

3. **Thread level parallelism.** C-RAN aggregates the processing of multiple sites, exhibiting TLP.

The throughput achieved for the Turbo decoder in the MAC layer is shown in Figure 5.3. The Turbo decoder had similar results across all four BBU configurations, therefore I show here only the 1×1+QPSK result. As it shows, the Turbo decoder throughput on both the CPU and GPU are much lower than the specification throughput. To investigate this, I used the NVIDIA Profiler [7] to profile the GPU utilization when running the LTE BBU. Figure 5.4 shows the achieved

GPU occupancy for the PHY layer kernels and the Turbo decoder. As it presents, the GPU occupancies of the PHY layer configurations saturate at 80%, while the Turbo decoder only reaches 60%. This low occupancy of the Turbo decoder is due to the fact that the Turbo decoder uses almost twice as many registers as the PHY layer kernels, limiting the number of concurrent threads that can be issued. In addition, the Turbo decoder starts saturating with 4 sites, which is earlier than the PHY layer. This is because that the Turbo decoder is computationally intensive and reaches the GPU concurrent thread limitation with a small number of sites.

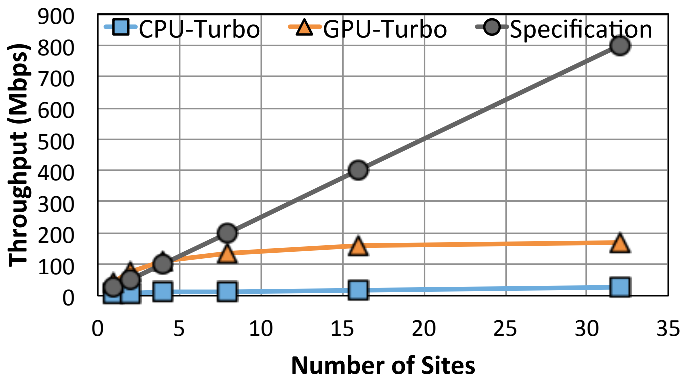


Figure 5.3 Throughput of the Turbo kernel compared to the LTE specification

5.2.3 Building a C-RAN datacenter with CPUs and GPUs

A C-RAN datacenter is expected to cover an area with a radius over 20 km and more than 20 sites [63, 69]. To satisfy this design goal, I target on building a datacenter capable of supporting 32 RRH sites at peak load. In this section, I aim to identify how many servers of each platform type are required in a 32-site

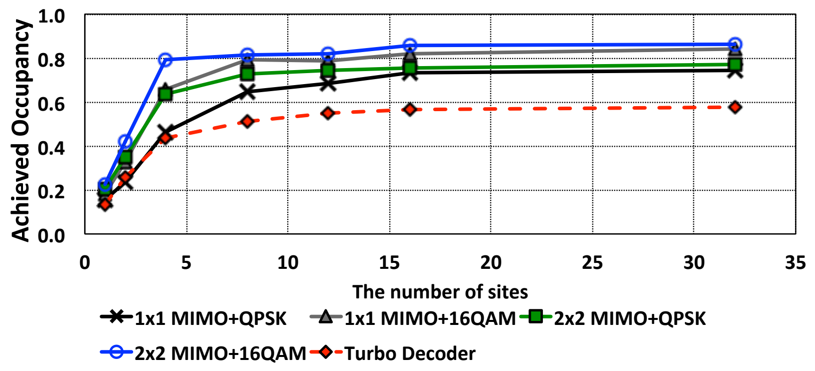


Figure 5.4 Achieved GPU Occupancy of the C-RAN BBU. The PHY layer kernels saturate at 80% occupancy, and the Turbo decoder saturates at 60%.

C-RAN datacenter.

In order to identify the number of machine per BBU configuration, I first identify how many sites at full load one machine can support. For example, in the 1×1+QPSK configuration in Figure 5.2(a), a single GPU can achieve the specification throughput at 32 sites. Thus, a single GPU will be sufficient in that datacenter. However, a CPU server can achieve the specification throughput of only 4 sites. Thus, 8 CPU servers will be required in that datacenter to support 32 sites. In this manner, I calculate the number of CPU servers and GPU servers required per datacenter of each configuration. Results are presented in Figure 5.5. As shown, the GPU-based solution requires $4\times$ to $16\times$ more CPUs than the equivalent GPU-based datacenter.

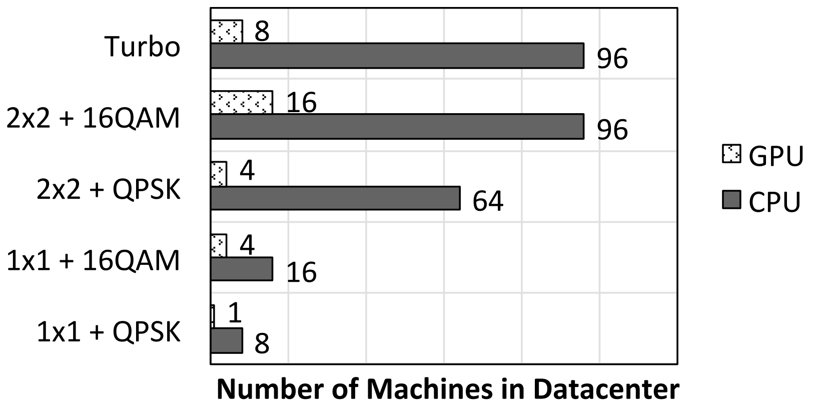


Figure 5.5 Number of servers to realize a 32-site datacenter

5.2.4 Energy and TCO analysis

Although the GPU achieves better performance than the CPU, servers that accommodate GPUs are more power hungry and expensive. Thus, the energy and cost efficiency are among the primary concerns when building a C-RAN datacenter. In this section, I evaluate the energy and TCO for both platforms.

5.2.4.1 Energy analysis

I measured the CPU energy using the “Watts Up?” power meter, which samples and logs power values every second. I subtract the idle power to eliminate the influence of peripherals. I measured the GPU energy using the NVIDIA system management interface —*NVIDIA-smi*. I sampled the GPU power every 100 ms and averaged them to get the power for each configuration. I found that the GPU power did not vary much between configurations, and was roughly between 64W – 66W.

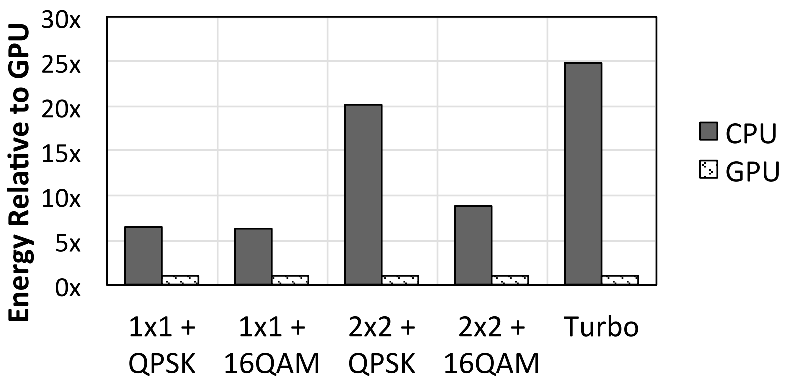


Figure 5.6 Energy required to process 32 sites at full load

In Figure 5.6, I present the energy consumed by CPUs normalized to the energy consumed by GPUs. These results are collected for a C-RAN datacenter processing 32 sites at peak load. The CPU consumes between 6× to 25× more energy than the GPU.

Table 5.4 Server Design for TCO Analysis

Parameter	High-Density CPU Server [15]	GPU-Server [16]
CPU	8× Intel Xeon E5-2630 v3	2× Intel Xeon E5-2620 v3
GPU	-	{1-4}× Tesla K40 m
Memory	32× 4GB DDR4 DIMM	16× 8GB DDR4 DIMM
HDD	4× 500GB 3.5" SATA	4× 500GB 3.5" SATA
Server Price	$13,032	$5,978 + #GPUs×$3,099
Server Power	1000W	357W + #GPUs×235W

5.2.4.2 TCO analysis

A TCO analysis is important in building a C-RAN datacenter because while GPUs have better performance and energy consumption, they increase the cost of designing a datacenter. Expensive GPU boards can cost as much as a non-GPU server and need to be programmed with platform-specific languages such as CUDA. I present a cost analysis to determine whether the benefits gotten from GPUs are worth the cost.

Table 5.5 TCO Parameters

Parameter	CPU-Only	GPU-Server
Server Price	$13,032	$5,978
Price per 235W GPU	-	$3,099
Server Power	1000W	357.0W
Server Maintenance	5% of Capex/mo	
Server Lifetime	3 years	
Datacenter Price	$10/W	
Datacenter Maintenance	$0.04/W/mo	
Loan Amortization Period	12 years	
Power Usage Efficiency	1.1	
Electricity Cost	$0.067 per kWh	
Interest Rate	8%	

The TCO analysis is modeled after the technique from Barroso and Hölzle [32]. To minimize costs of the CPU, I chose high-density servers, which are larger than typical servers at 2U or 4U and can accommodate many server cartrdiges, allowing the cost of the chassis, cooling, and network switches to be shared. The server configurations and the server prices I used were obtained from Thinkmate.com and are shown in Table 5.4. The parameters used for the TCO calculation are shown in Table 5.5.

Figure 5.7 shows the results of the TCO analysis. The results are normalized to the TCO of the GPU-based datacenter. Assuming all 32 sites are always operating at peak load, the TCO of the CPU-based datacenter is on average 6× higher and can be up to 12× higher than that of the GPU-based datacenter. I conclude that in

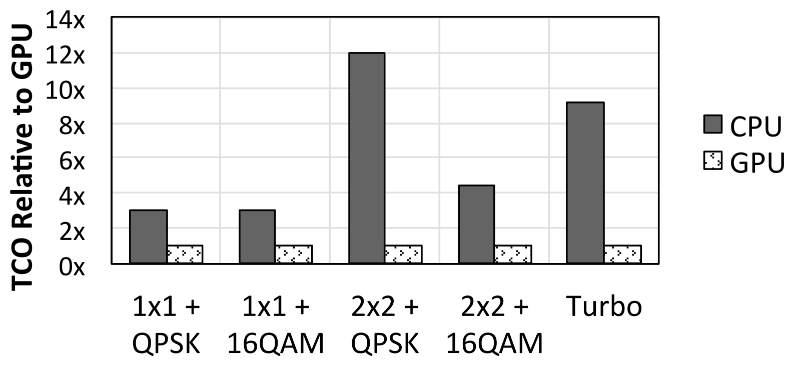


Figure 5.7 TCO of the 32-site datacenter

addition to better performance and lower energy, the GPU-based server also has lower cost than the CPU-based server, indicating that the GPU-based server is the best general-purpose platform for the C-RAN datacenter.

5.3 Resource Management in a C-RAN Datacenter

In the previous section, I determined that a datacenter consisting of GPUs are the best design option for C-RAN. One of the major benefits of centralizing the BBUs into a datacenter is the capability to share resources and save power. We must be able to effectively allocate the datacenter's resources to match the workload and maintain quality-of-service.

In the radio network, traffic is never equally distributed in a day or between sites. If we have all machines up and running and do not carefully distribute the workload among the machines, it results in over-provisioned power consumption and unbalanced load. In this section, I investigate power management techniques to reduce the datacenter's power consumption. In addition, I try to find the best way to distribute the workload to achieve better throughput during times when traffic is unequally distributed.

5.3.1 Power management

The RAN traffic varies constantly, and the peak traffic only carries 7% of the time in a day [94]. Because a C-RAN datacenter is over-provisioned for the peak traffic, power management is required to reduce energy during most times of a day when the traffic is below the peak. There are two typical power saving techniques in the server domain: coarse-grained turning on/off of processors and fine-grained DVFS.

5.3.1.1 Power management techniques

With shutting down of processors, a minimum number of processors are kept powered on to meet the quality-of-service, and the rest of the processors are turned off to save power. While shutting down processors is an effective method to reduce power consumption in a datacenter, turning on/off a processor has long transition overhead, usually on the minute level. This means that this technique cannot react to a quick fluctuation in traffic.

DVFS, on the other hand, can react to quick traffic changes due to its micro-second level transition overhead [83]. Compared to the latency budget in an LTE BBU (usually 4 ms), the DVFS transition overhead is negligible. The drawback of

DVFS is that every processor has a limited number of frequencies and voltages, restricting the achievable power saving. For example, an NVIDIA K40 GPU has only five (core, memory) frequency settings in MHz: (324, 324), (666, 3004), (745, 3004), (815, 3004) and (875, 3004).

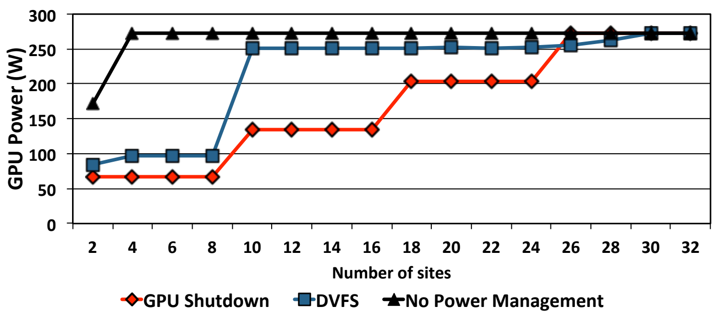


Figure 5.8 Power consumption of different power management techniques. Four GPUs and 2×2+QPSK are used in the study. Due to the coarse-grained frequency adjustment of NVIDIA K40 GPUs, DVFS saves less power than GPU shutdown.

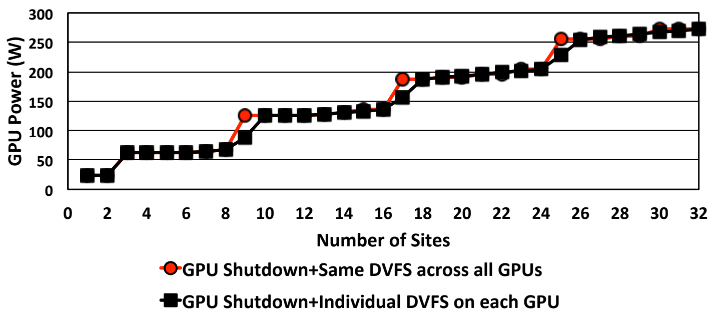


Figure 5.9 Power consumption of combining GPU shutdown and DVFS. Four GPUs and 2×2+QPSK are used in the study. Per-GPU DVFS achieves more power savings due to the flexibility of choosing an optimal frequency for each GPU.

5.3.1.2 Power management for a C-RAN datacenter

In Figure 5.8, I compare the power consumption of both the GPU shutdown technique and the DVFS technique. I used four GPUs and 2×2+QPSK as the case study. The traffic metric is the number of sites, i.e. I consider one unit of the workload to be the workload from a single site running at peak load. The baseline has no power management techniques with all four GPUs turned on and operating at the highest frequency to handle traffic from all 32 sites. For the GPU shutdown technique, the minimum number of GPUs required to meet the throughput and latency is powered on and workload is evenly distributed among all active GPUs. For the DVFS technique, all four GPUs are active, but only operating at the minimum frequency that is needed to meet the throughput and latency. As shown, both GPU shutdown and DVFS save power compared to no power management. In addition, GPU shutdown saves more power than DVFS. This is because that NVIDIA K40 GPUs have limited coarse-grained frequency adjustment as stated earlier. On average, GPU shutdown and DVFS saves 36% and 20% of power respectively.

Although GPU shutdown saves more power than DVFS, it cannot react to short traffic changes, which is typical in the RAN. In addition, the GPU peak frequency is still over-provisioned for low traffic cases. Therefore, I combine the GPU shutdown and DVFS as the power management technique in a C-RAN datacenter. To assess the effectiveness of this technique, I study its power saving under the oracle knowledge of traffic and power consumption of each combination

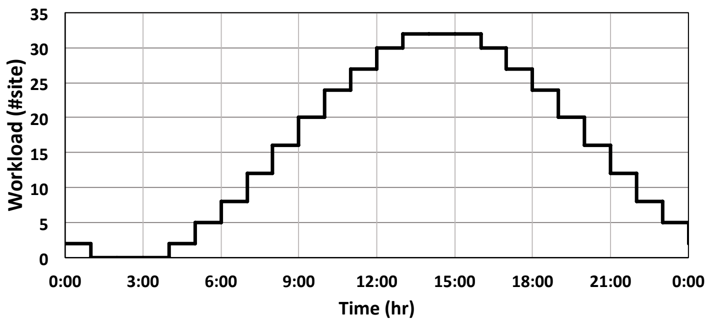


Figure 5.10 C-RAN traffic in 24 hours. The wireless traffic follows the typical periodic night/day pattern, and I use a sinusoidal function to model the C-RAN traffic in a 24-hour period [37].

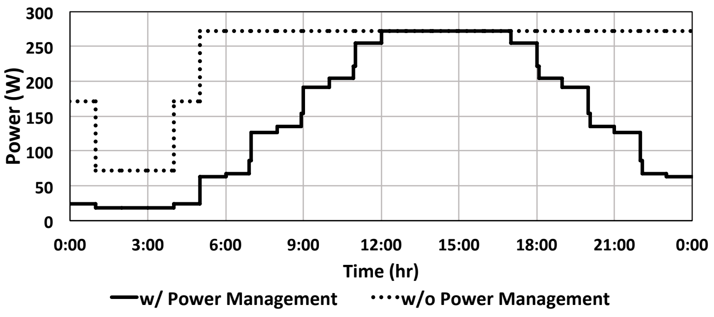


Figure 5.11 Power consumption of a quad-GPU C-RAN datacenter with and without power management in 24 hours. Four GPUs and 2×2+QPSK are used in the study. The values are the minimum power required to meet the throughput and latency requirements, following the traffic pattern in Figure 5.10.

of the GPU number and the frequency. For every traffic case in Figure 5.8, I evaluate all possible combinations of the number of GPUs and the frequency that can meet the performance requirement, and select the one consuming the least power. In this way, I get the global optimal power consumption under the proposed power management technique, and Figure 5.9 shows the results. As we can see, combining GPU shutdown and DVFS achieves more power savings than each technique individually. In addition, using individual GPU DVFS is better than using same DVFS for all GPUs, due to the flexibility of choosing an optimal frequency for each GPU. However, individual GPU DVFS requires load monitoring for each GPU, which is more costly than load monitoring the datacenter as a whole. Overall, GPU shutdown with same DVFS across all GPUs and individual DVFS per GPU save 40% and 42% power compared with no power management respectively, which are 4% and 6% more than GPU shutdown.

5.3.1.3 "Hill-climbing" power management evaluation under practical traffic model

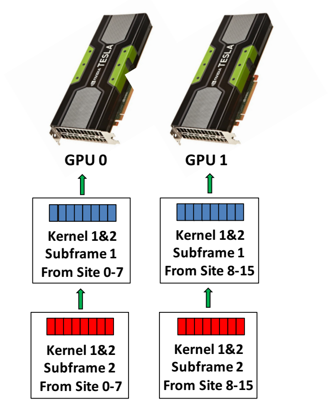
To validate the proposed power management technique, I evaluate its energy savings under a practical C-RAN traffic model. The wireless traffic follows the typical periodic night/day pattern, and I use a sinusoidal model from [37] to build the C-RAN traffic model in this study(Figure 5.10).

Because there is no oracle information about the power of different GPU and frequency combinations available in practice, I use a "hill-climbing" method in the proposed power management technique. I assume that no GPU is active when there is zero traffic. When the traffic starts increasing/decreasing, I first adjust the frequency of all active GPUs to meet the performance requirement. If it cannot, I turn on/off one GPU and adjust frequencies until the performance is met. This "hill-climbing" method is simple and can find a local optimal solution, but it may not find the global optimal solution. For example, if there are two solutions, two GPUs with a high frequency and three GPUs with a low frequency, when starting from one GPU, the “hill-climbing” will stop at the two-GPU solution, and will never reach the three-GPU one which could be the global optimal solution. In addition, I assume an accurate prediction of the traffic change, and conservatively assume that turning on/off GPUs takes 5 minutes. Therefore, in the 5 minutes before/after every traffic change, the GPUs that will be turned on/off are idle, and consume 18 W each.

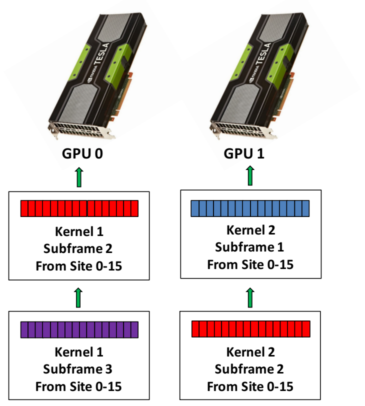
Figure 5.11 shows the power consumption of a quad-GPU C-RAN datacenter with and without the “hill-climbing” power management under the 24-hour traffic model. Based on Figure 5.11, a C-RAN datacenter consumes 6 kWh in 24 hours without power management and 3.6 kWh with power management. Therefore, with power management, a C-RAN datacenter saves 40% of the BBU energy.

5.3.2 Load balancing

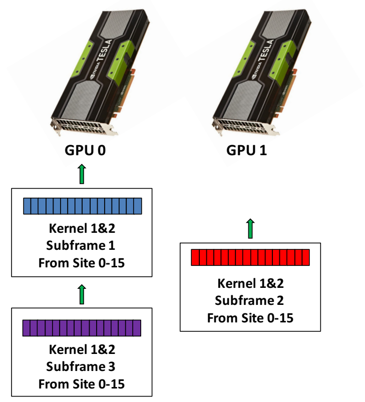
In addition to the power management, we must also think carefully about how to distribute the workload when multiple GPUs are available in the datacenter such that we are maximizing the capacity of the datacenter. Since the RAN traffic is rarely equally distributed between sites (i.e. some sites maybe servicing higher load), determining how many sites should be serviced by each GPU with load balancing techniques is necessary to improve the GPU throughput and utilization. I develop three techniques to achieve different extents of load balancing. Figure 5.12 presents these three techniques: fixed assignment, pipelining kernels and pipelining packets.



(a) Fixed Assignment



(b) Pipelining Kernels



(c) Pipelining Packets

Figure 5.12 C-RAN BBU implementation on multiple GPUs

5.3.2.1 Workload distribution with load balancing

For this section, I assume a simplified datacenter with 2 GPUs and 16 sites with 2×2+QPSK.

Fixed assignment (Figure 5.12a) – The first technique assigns all traffic from a single site to be processed by the same GPU, and all its processing is done on that GPU only. For example, sites 0-7 are assigned to GPU 0, and sites 8-15 are assigned to GPU 1. The fixed assignment is simple, and has a short processing latency because only a subset of sites are processed on each GPU. However, it can only balance load among the group of sites also assigned to the same GPU, which is very limited. When there is imbalance across site groups (e.g. sites 0-7 are sending traffic while sites 8-15 are idle), it suffers from low throughput due to low hardware utilization.

A better technique is to use multiple GPUs to pipeline the processing. There are two types of pipelining techniques: pipelining kernels and pipelining packets.

Pipelining kernels (Figure 5.12b) – In pipelining kernels, kernels are assigned to different GPUs and each GPU processes those kernels for all the sites. In this case, GPUs form a *pipeline of kernels*. For example, GPU 0 processes Kernel 1 and GPU 1 processes Kernel 2 for all 16 sites.

Pipelining packets (Figure 5.12c) – In pipelining packets, each packet has traffic from all sites. All the kernels for that packet are processed by one GPU. The next packet is sent to another GPU. In the example, one subframe[1] is a packet. GPU 0 processes subframes with odd indices from all 16 sites, and GPU 1 processes subframes with even indices.

Compared to the fixed assignment technique, both pipelining techniques balance the workload by aggregating the processing of all sites. This results in better hardware utilization and system throughput. In addition, pipelining kernels allows GPUs to be different from each other and be selected specifically to accelerate kernels allocated to them. However, pipelining techniques suffer from longer la-

[1]A subframe is a sub-component of a radio frame. It lasts 1ms in length.

tency due to aggregated processing of all sites (for both techniques) and inter-GPU communication through PCI-E (for pipelining kernels only).

5.3.2.2 Performance evaluation of distribution techniques

I evaluate the three techniques with balanced and unbalanced loads to identify which achieves the best performance. For the balanced case, all 16 sites are running with full workload; for the unbalanced case, sites 0-7 are running with full workload and sites 8-15 are running with half workload. I measure both the achieved throughput and subframe processing latencies of all three techniques.

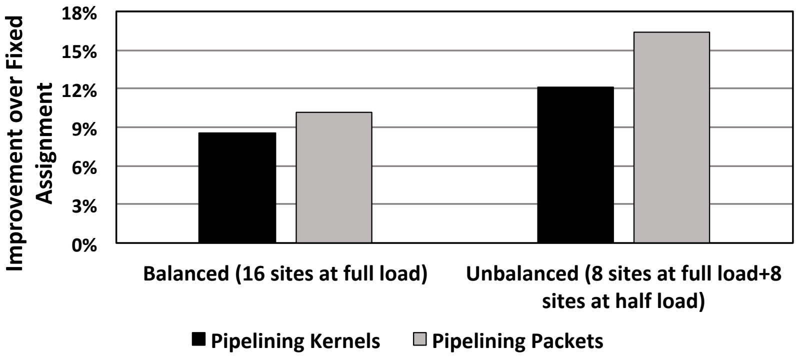


Figure 5.13 Throughput improvements of two pipelining techniques over the fixed assignment. Overall, pipelining packets improves throughput by 10% and 16% under balanced and unbalanced loads respectively, over the fixed assignment.

Figure 5.13 presents the throughput improvements of two pipelining techniques over the fixed assignment. As shown, both pipelining techniques achieve better throughput than the fixed assignment. In addition, pipelining techniques accomplish more throughput improvements in the case of unbalanced loads, indicating that they have better load balancing capabilities. Pipelining packets achieves better

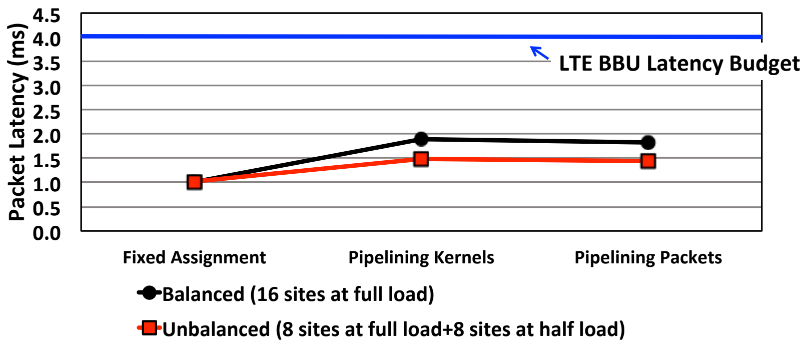


Figure 5.14 Subframe processing latencies of three distribution techniques. The fixed assignment has the shortest processing latency, due to processing a subset of sites on each GPU.

throughput than pipelining kernels, because pipelining kernels suffers from the extra inter-GPU communication. Also, I use homogeneous GPUs in this study, which limits the benefit of supporting heterogeneous GPUs from pipelining kernels. Overall, pipelining packets improves throughput by 10% and 16% under balanced and unbalanced loads, respectively.

Figure 5.14 presents the subframe processing latencies of all three techniques. Both pipelining techniques have longer latencies, due to aggregating processing of all sites to achieve better load balancing. In addition, pipelining kernels has the worst latencies, which are caused by the PCI-E inter-GPU communication. Overall, both pipelining techniques incur 40% more latencies than the fixed assignment under unbalanced workload, which are still in the 4 ms LTE BBU latency budget.

In summary, when multiple GPUs are available in the datacenter, pipelining techniques achieve better throughput than the fixed assignment through load balancing, but at the cost of longer processing latencies. Overall, pipelining packets is the best technique, due to its highest throughput and acceptable processing latency. In addition, pipelining packets allows dynamic adjustment of the number of GPUs, which is required by “hill-climbing” power management.

5.4 Implications for C-RAN datacenter design

Based on the study in the prior sections, I now discuss implications of the C-RAN datacenter design. Section 5.4.1 suggests an alternate solution to addressing the limited throughput of the Turbo decoder. Section 5.4.2 suggests other ways to make use of the idle hardware during non-peak times of the day.

5.4.1 ISA extension support for trellis algorithms

One key observation I made in the study was that due to its heavy computation and high register usage, the Turbo decoder’s achievable throughput was much lower than the PHY layer kernels. Therefore, there is a need to improve the performance of the Turbo decoder in the C-RAN datacenter.

In fact, the Turbo decoder algorithm is a member of trellis algorithms, a family of algorithms widely used in many areas such as coding theory, speech recognition and data compression. Other well-known members of the trellis algorithms include Viterbi algorithm [56] and Baum-Welch algorithm [34]. As trellis algorithms are widely used and share similar computational features, accelerating them on general-purpose platforms is a worthwhile research goal to pursue.

I suggest that GPUs have the trellis accelerator and instruction set extension to improve the performance of the Turbo decoder. Several similar hardware accelerators with instruction set support already existed in commercial CPUs, such as the AES instruction extension [48] in x86 CPUs.

5.4.2 Service at the wireless edge

Although C-RAN has better hardware utilization by centralizing the processing of multiple sites, there are still times that the datacenter utilization is low due to the overall low traffic load from the area. A good example is that there are more tourists on the Manhattan island in the summer than the winter because of the severe weather in winter. To guarantee the quality and the coverage of the service, the C-RAN datacenter needs to be designed to support the peak amount of tourists during the summer. This leads to low utilization of the datacenter in the rest of the year, and opens up opportunities to use the hardware during idle time.

One solution is to run internet services in the C-RAN server, pushing the internet service from the core network to the wireless edge. Since the C-RAN server is closer to the end-users, the latency of a service request can be greatly reduced, improving the quality of the internet service. In addition, some services that have been proven to fit in the GPU-based server [71, 70] can take advantage of local information that is only available in the C-RAN server, also improving the service quality and reducing the complexity of the server system.

5.5 Related Work

Since C-RAN is still a new cloud service, there is little study on the datacenter design for the C-RAN. Authors of [63] evaluated a CPU-based C-RAN BBU pool, and showed that 20 Intel CPUs are required for the C-RAN BBU datacenter. The paper lacks details on the hardware platform used for evaluation, and the deployed algorithms for each kernel, making it difficult for readers to get take-home messages. Their result of having 20 CPUs proves the discovery to some extent that a GPU-based cloud datacenter is more energy and cost efficient than a CPU-based datacenter. Open wireless system cloud [42] is a RAN architecture similar to C-RAN. Authors used an Intel x86 blade server and a Cell/B.E. blade server to evaluate a C-RAN WiMax datacenter. The paper mainly focuses on the radio network design, instead of the cloud datacenter design.

There are several previous works on using GPUs to build software-defined LTE base stations [117, 118, 62, 97, 30, 104]. They discussed the GPU implementations of key signal processing kernels, and evaluated the performance. The main focus of these papers is the GPU implementations of a traditional LTE base station. This work targets on the C-RAN BBU datacenter design, and focuses on improving the performance and energy efficiency through resource management. There are previous works on the C-RAN datacenter resource management [35, 113]. They focus on the high-level management algorithm design, without the implementation details on the actual hardware.

A great number of prior works have been done on the power management in datacenters. Many works examined the power management techniques on processing units, including DVFS [51, 52, 102, 53], processor power gating [84, 90] and using heterogeneous computing components in datacenters [85, 76, 79]. Other works focused on the memory system, such as DRAM [47, 49, 50] and hard disks [38, 66]. There are also several works that examined various approaches to accomplish energy-proportionality in datacenters [89, 107, 88, 72]. And finally, [99] combined many existing power management techniques, and studied the coordinations between these techniques to achieve better energy saving in a datacenter. My proposed power management deploys existing power management techniques, and applies them to handle the C-RAN temporal traffic pattern.

There are papers on accelerating other services on a GPU-based datacenter. [71] and [77] evaluated using GPUs to improve of the throughput of a warehouse scale datacenter for the new emerging voice personal assistant. This work focuses on C-RAN, the emerging wireless application as cloud service, and is orthogonal to these previous works.

5.6 Summary

In this chapter, I discussed designing a datacenter for C-RAN using commodity server platforms. C-RAN was proposed to be a future wireless network that will solve the scalability and cost problems of the traditional RAN. Unfortunately, C-RAN is still a new concept with no clear models for deployment using general-purpose server platforms. With my work, I bring to light the challenges of designing

C-RAN BBUs on general-purpose platforms, and outline design objectives that C-RAN designers should adhere to.

I design the C-RAN datacenter and also investigate how to manage its resources to best fit the needs of C-RAN services. I compare CPU-based servers and GPU-based servers, and find that across the three metrics of performance, energy, and TCO, GPU-based datacenters best meet my design objectives: 1) meeting the throughput requirement of the LTE specification, 2) supporting 20 sites or more and 3) reducing energy by 13$\times$ and TCO by 6$\times$ over CPU-based datacenters (Section 5.2).

To effectively manage the resources of the datacenter, I have two objectives: save power and balance the load. Among the few power management techniques I evaluated, the combination of turning on/off GPUs and DVFS is able to save 40% of the BBU energy without violating throughput and latency requirements. Among the three load balancing techniques I evaluated, pipelining packets using multiple GPUs is the best at managing spatial imbalance in the traffic pattern, achieving 16% throughput improvement (Section 5.3).

CHAPTER VI

Conclusion

Wireless communication has become one of the critical uses of computing resources and technologies. The increasing number of mobile device users result in more energy consumption and cost in the traditional wireless radio access network with worst throughput and quality of service. In addition, emerging wireless protocols and fast wireless technology development make using hardwired ASIC solutions more expensive and complex in wireless base stations. Therefore, as a future radio access network, C-RAN was proposed to solve all these problems. However, the design of a C-RAN datacenter has yet been studied. This dissertation characterizes the wireless signal processing applications, and explores both the software and hardware design of a datacenter using commodity servers for C-RAN BBUs. Resource management techniques are then studied to save power and balance the load in the presence of the temporal or spatial workload imbalance.

6.1 Summary

In this thesis, I study how to design a datacenter for the future C-RAN BBUs. In Chapter III, I presents the design of WiBench, which is used to characterize the wireless signal processing applications. WiBench contains the key signal processing kernels of major wireless protocols. The characterization study of WiBench indicates that processors that can explore DLP and TLP to provide high computing throughputs are the good hardware platforms for a C-RAN datacenter.

In Chapter IV, high performance software implementations of the LTE uplink kernels are explored and discussed. Particularly, I present the parallelization techniques in CUDA for GPUs. These techniques are deployed to make full use of the GPU computing resources, in order to achieve high throughputs. In addition, since the Turbo decoder is a member of the trellis algorithms, I extend its parallelization techniques to study the tradeoffs between throughput, latency and the bit error rate for the trellis algorithms.

Chapter V finally discusses the design of a C-RAN datacenter. As the first and the most important step, I evaluate two major commodity general-purpose servers, multi-core CPUs and GPUs. By comparing them through the performance, energy efficiency and TCO, I come up with the conclusion that GPUs are the better servers be used in C-RAN datacenters. Then I study the resource management schemes in a GPU-based C-RAN datacenter to handle the temporal and spatial workload imbalance in the radio access network.

As C-RAN will be more widely deployed in the future, a high-performance and energy-efficient datacenter design for C-RAN will become more crucial, and there will be more research being done in this area.

6.2 Future Work

There are several opening topics that are potential future research directions to extend this thesis.

6.2.1 Fixed-point implementations of C-RAN BBU

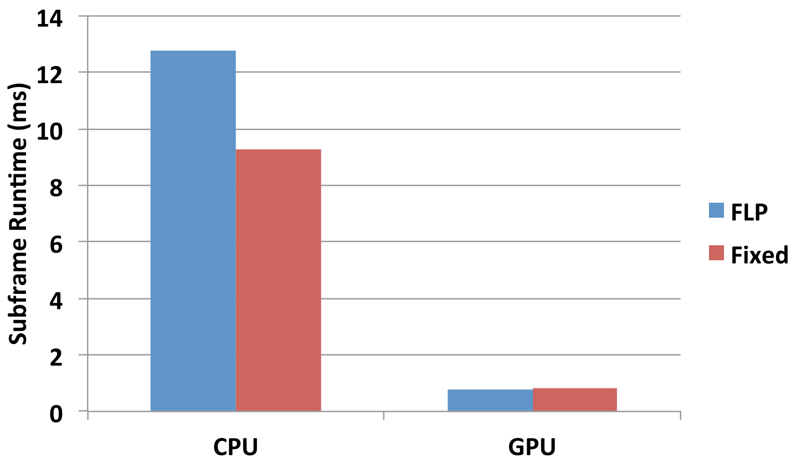


Figure 6.1 Runtimes of 32-bit floating-point and 16-bit fixed-point implementations of demodulation

In this thesis, BBU kernels were implemented based on single precision floating-point data (32 bits) to take advantage of the powerful floating-point engines on general purpose platforms. However, sometimes data with less precision, such as 16-bit fixed-point values, may be enough for wireless signal processing. Therefore, it is interesting as a future work to investigate the influence of using fixed-point data with less precision in C-RAN BBU on performance, power consumption and bit error rate. As an initial study, I implement a 16-bit fixed-point version of demodulation in both C++ and CUDA, and compare its performance and power with the floating-point implementation. Figure 6.1 and 6.2 show the performance and

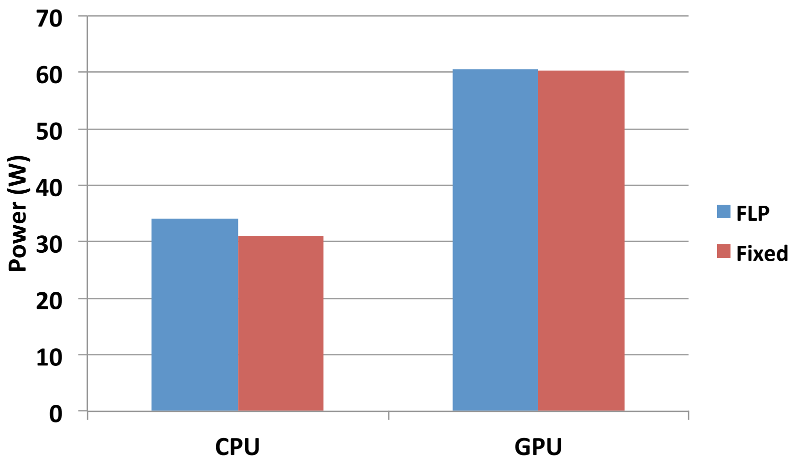


Figure 6.2 Power of 32-bit floating-point and 16-bit fixed-point implementations of demodulation

power comparisons between 32-bit floating-point and 16-bit fixed-point implementations of the demodulation kernel. As we can see, for the CUDA implementation on GPUs, there is hardly any difference between floating-point and fixed-point implementations for both performance and power. However, for the CPU code, although the power consumption is still quite close between two implementations, the fixed-point version has significant better performance. This is due to the fact that the 16-bit fixed-point data can make better use of the AVX extension in an Intel CPU, and double the throughput of each AVX instruction compared with the 32-bit floating-point data. Therefore, in the future work, fixed-point implementations of the BBU system should be investigated to further improve the software in a C-RAN datacenter.

6.2.2 Streaming input data into the GPU memory

When using GPUs for the wireless communication, one key stage is to stream the input raw data into the GPU memory. Traditionally, GPUs have no I/O capability to read data from the external devices, and thus require data to be staged through the CPU memory first. This results in long data movement latency, which harms the quality of service of C-RAN. Therefore, how to stream input data into the GPU memory is important for a GPU-based C-RAN datacenter. There are two possible solutions: one is to use integrated CPU+GPU SoC, and the other is to enable GPUs with I/O to the external devices.

Through using the integrated CPU+GPU SoC, CPUs and GPUs on the same chip will share the main memory and even the last level cache. As a result, the data movement latency between CPUs and GPUs is greatly reduced. In fact, many commercial CPU+GPU SoCs are available already, such as Intel Ivy Bridge and NVIDIA Tegra K1. However, how to manage the CPU and GPU data in the shared memory is still under study, and can greatly affect the performance of GPUs.

Enabling GPU-based I/O to the external devices is another possible solution to solve the input data problem. In this case, many additional software and hardware supports are required, including the operating system, the I/P hub and the modified drivers. NVIDIA Kepler GPUs already have GPUDirect to support the RDMA feature and allow direct access to GPU memory by third-part devices. Therefore, the future work can evaluate the performance of using GPUDirect in a C-RAN datacenter, and design an even better I/O interface for GPUs to support future wireless protocols.

6.2.3 Providing internet service at the wireless edge

As discussed in Chapter V, internet services can benefit significantly from being pushed to C-RAN datacenters, because the round-trip latency of a service request is greatly reduced, and more accurate geographical information is available to provide better customized services. However, since the internet and wireless services will share the hardware resources, good management technologies are required to coordinate the hardware sharing in order to maintain the good quality of both services. In addition, careful decisions need to be made to determine what internet services should be offloaded to the wireless edge, to take advantage of the available geographical information.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] [Online]. Available: http://en.kioskea.net/contents/telephonie-mobile/gsm.php3

[2] [Online]. Available: http://www.scribd.com/doc/19345748/Cellular-Telephone-Networks

[3] Atom (system on chip). [Online]. Available: http://en.wikipedia.org/wiki/Atom_(system_on_chip)

[4] "BDTI^{TM} OFDM Receiver Benchmark." [Online]. Available: http://www.bdti.com/Services/Benchmarks/OFDM

[5] CEVA-XC4000. [Online]. Available: http://www.ceva-dsp.com/CEVA-XC4000.html

[6] "CTIA Annual Wireless Industry Survey." [Online]. Available: http://www.ctia.org/your-wireless-life/how-wireless-works/annual-wireless-industry-survey

[7] "CUDA Toolkit Documentation." [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/

[8] CUFFT User Guide. [Online]. Available: http://docs.nvidia.com/cuda/cufft/index.html

[9] "GNU Radio." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki

[10] List of Intel Atom microprocessors. [Online]. Available: http://en.wikipedia.org/wiki/List_of_Intel_Atom_microprocessors

[11] LTE Channel Card Solutions. [Online]. Available: http://www.altera.com/end-markets/wireless/lte/channel-card/wir-lte-channel-solution.html

[12] "LTE PHY Downlink with Spatial Multiplexing." [Online]. Available: http://www.mathworks.com/help/comm/examples/lte-phy-downlink-with-spatial-multiplexing.html

[13] MATLAB Central–File Exchange. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/

[14] Smartphones with Intel Inside. [Online]. Available: http://www.intel.com/content/www/us/en/smartphones/smartphones.html

[15] "Thinkmate, High Performance Computing," http://www.thinkmate.com/system/hdx-xa12-5260v3, accessed:2015-08-10.

[16] "Thinkmate, High Performance Computing," http://www.thinkmate.com/system/gpx-xt4-2160v3-4gpu, accessed:2015-08-10.

[17] Wiki page: Base station subsystem. [Online]. Available: http://en.wikipedia.org/wiki/Base_station_subsyste

[18] "Wiki papge: Channel state information." [Online]. Available: http://en.wikipedia.org/wiki/Channel_state_information

[19] "Wireless Communications Technologies and Research Trends: Long Term Evolution-Advanced (LTE-A) and Beyond." [Online]. Available: http://www.iaria.org/conferences2013/filesICWMC13/Abdul_Nice_WirelessCommunicationsTechnologies.pdf

[20] (2011) Gigabyte GA-X79-UD7. [Online]. Available: http://www.gigabyte.us/products/product-page.aspx?pid=4047#sp

[21] *LTE Specification*, 3GPP Std. 36.521.

[22] *LTE Specification*, 3GPP Std. 36.312.

[23] *LTE Specification*, 3GPP Std. 36.211.

[24] *LTE Specification*, 3GPP Std. 36.212.

[25] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge, “Scaling towards kilo-core processors with asymmetric high-radix topologies,” in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*. IEEE, 2013, pp. 496–507.

[26] “Product Brief: 4G (LTE/WiMAX) Base Transceiver Station AMC,” Accipiter Systems, 2010.

[27] “Alcatel-Lucent 9926 Digital 2U eNodeB Baseband Unit,” Product Brief, Alcatel-Lucent, 2009.

[28] N. Alliance, “Suggestions on Potential Solutions to C-RAN,” *White Paper, Version*, vol. 4, 2013.

[29] A. S. Al_safi, B. Bazuin, and L. Alhafadhi, “Parallel Computation in Communication and Signal Processing,” *IJRECE*, vol. 3, no. 3, pp. 67–69, 2015.

[30] A. S. Al_safi, B. Bazuin, and L. Alhafadhi, “Parallel Computation in Communication and Signal Processing,” *IJRECE*, vol. 3, no. 3, pp. 67–69, 2015.

[31] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate,” *IEEE Trans. Intell. Transport. Syst.*, vol. 20, pp. 284–287, Mar. 1974.

[32] L. A. Barroso, J. Clidaras, and U. Hölzle, “The datacenter as a computer: An introduction to the design of warehouse-scale machines,” *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[33] L. 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*, vol. 41, pp. 164–171, Feb. 1970.

[34] 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.

[35] S. Bhaumik, S. P. Chandrabose, M. K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, and T. Woo, “CloudIQ: a framework for processing base stations in a data center,” in *Proceedings of the 18th annual*

international conference on Mobile computing and networking. ACM, 2012, pp. 125–136.

[36] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC benchmark suite: Characterization and architectural implications,” in *Proceedings of the 17th International Conference on Parallel architectures and Compilation Techniques*, 2008, pp. 72–81.

[37] A. Bousia, A. Antonopoulos, L. Alonso, and C. Verikoukis, “” Green” distance-aware base station sleeping algorithm in LTE-Advanced,” in *2012 IEEE International Conference on Communications (ICC)*. IEEE, 2012, pp. 1347–1351.

[38] E. V. Carrera, E. Pinheiro, and R. Bianchini, “Conserving disk energy in network servers,” in *Proceedings of the 17th annual international conference on Supercomputing*. ACM, 2003, pp. 86–97.

[39] A. Carroll and G. Heiser, “An Analysis of Power Consumption in a Smartphone,” in *USENUX*, 2010.

[40] P. Chanclou, A. Pizzinat, F. Le Clech, T.-L. Reedeker, Y. Lagadec, F. Saliou, B. Le Guyader, L. Guillo, Q. Deniel, S. Gosselin, S. Le, T. Diallo, R. Brenot, F. Lelarge, L. Marazzi, P. Parolari, M. Martinelli, S. O’Dull, S. Gebrewold, D. Hillerkuss, J. Leuthold, G. Gavioli, and P. Galli, “Optical fiber solution for mobile fronthaul to achieve cloud radio access network,” in *2013 Future Network and Mobile Summit (FutureNetworkSummit)*, July 2013, pp. 1–11.

[41] C. Chen, “C-RAN: the Road Towards Green Radio Access Network,” White Paper, China Mobile, 2011.

[42] J. Chen, X. Chen, J. Liu, and M. Zhao, “Open wireless system cloud: An architecture for future wireless communications system,” *Network and Communication Technologies*, vol. 1, no. 2, p. p28, 2012.

[43] J. Clemons, H. Zhu, S. Savarese, and T. Austin, “MEVBench: A Mobile Computer Vision Benchmarking Suite,” in *IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 91–102.

[44] “SMART 2020: Enabling the Low Carbon Economy in the Information Age,” The Climate Group on behalf of the Global eSustainability

Initiative (GeSI), 2008. Available: http://www.smart2020.org/_assets/les/02_Smart2020Report.pdf

[45] S. Coleri, M. Ergen, A. Puri, and A. Bahai, “Channel Estimation Techniques based on Pilot Arrangement in OFDM Systems,” *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, 2002.

[46] S. Coleri, M. Ergen, A. Puri, and A. Bahai, “Channel estimation techniques based on pilot arrangement in OFDM systems,” *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, 2002.

[47] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, “Hardware and software techniques for controlling dram power modes,” *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1154–1173, 2001.

[48] M. C. Demands, “Intel Security Technology for the Cloud,” 2012.

[49] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, “Memscale: active low-power modes for main memory,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, pp. 225–238, 2011.

[50] B. Diniz, D. Guedes, W. Meira Jr, and R. Bianchini, “Limiting the power consumption of main memory,” *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 290–301, 2007.

[51] M. Elnozahy, M. Kistler, and R. Rajamony, “Energy conservation policies for web servers,” in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems-Volume 4*. USENIX Association, 2003, pp. 8–8.

[52] X. Fan, C. S. Ellis, and A. R. Lebeck, “The synergy between power-aware memory systems and processor voltage scaling,” in *Power-Aware Computer Systems*. Springer, 2005, pp. 164–179.

[53] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[54] L. S. Ferreira, D. Pichon, A. Hatefi, A. Gomes, D. Dimitrova, T. Braun, G. Karagiannis, M. Karimzadeh, M. Branco, and L. M. Correia, "An architecture to offer cloud-based radio access network as a service," in *Networks and Communications (EuCNC), 2014 European Conference on*. IEEE, 2014, pp. 1–5.

[55] G. D. Forney, Jr., "The Viterbi Algorithm," in *Proc. of IEEE*, vol. 61, Mar. 1973, pp. 268–278.

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

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

[58] "Modular AdvancedMC Platform for Broadband/LTE Base Stations," Freescale Semiconductor, 2010.

[59] M. Frigo and S. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 1998, pp. 1381–1384 vol.3.

[60] Frigo, Matteo and Johnson, Steven G., "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[61] R. Gallager, "Low-density Parity-check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[62] Y. Gao, Y. Sun, C. H. Zhou, X. Su, X. B. Xu, and S. D. Zhou, "Accelerating the 3GPP LTE System Level Simulation with NVidia CUDA," in *Applied Mechanics and Materials*, vol. 58. Trans Tech Publ, 2011, pp. 1596–1601.

[63] Guangjie, Li and Senjie, Zhang and Xuebin, Yang and Fanglan, Liao and Tin-fook, Ngai and Zhang, Sunny and Chen, Kuilin, "Architecture of GPP based, scalable, large-scale C-RAN BBU pool," in *2012 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2012, pp. 267–272.

[64] B. Guo, W. Cao, A. Tao, and D. Samardzija, "CPRI compression transport for LTE and LTE-A signal in C-RAN," *2013 8th International Conference*

on Communications and Networking in China (CHINACOM), vol. 0, pp. 843–849, 2012.

[65] X. Guo and P. Song, "Simulink Based LTE System Simulator," M. Sci. thesis, Chalmers University of Technology, Goteborg, Sweden, 2010.

[66] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: dynamic speed control for power management in server class disks," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*. IEEE, 2003, pp. 169–179.

[67] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *IEEE International Workshop on Workload Characterization (WWC-4)*, 2001, pp. 3–14.

[68] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81–90.

[69] Hadzialic, Mesud and Dosenovic, Branko and Dzaferagic, Merim and Musovic, Jasmin, "Cloud-RAN: innovative radio access network architecture," in *2013 55th International Symposium ELMAR*. IEEE, 2013, pp. 115–120.

[70] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015, pp. 27–40.

[71] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang *et al.*, "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers," 2015.

[72] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 186–195.

[73] 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.

[74] “Intel Solutions for the Next Generation Multi-Radio Basestation,” Application Note, Intel, 2006.

[75] “The World in 2013: ICT Facts and Figures,” International Telecommunication Union, 2012.

[76] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, “Web search using mobile cores: quantifying and mitigating the price of efficiency,” *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 314–325, 2010.

[77] J. Kim, J. Chong, and I. R. Lane, “Efficient On-The-Fly Hypothesis Rescoring in a Hybrid GPU/CPU-based Large Vocabulary Continuous Speech Recognition Engine.” in *INTERSPEECH*, 2012.

[78] H. Kroll, S. Zwicky, B. Weber, C. Roth, D. Tschopp, C. Benkeser, A. Burg, and Q. Huang, “An Evolved GSM/EDGE Baseband ASIC Supporting Rx Diversity,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 7, pp. 1690–1701, July 2015.

[79] W. Lang, J. M. Patel, and S. Shankar, “Wimpy Node Clusters: What about non-wimpy workloads?” in *Proceedings of the Sixth International Workshop on Data Management on New Hardware*. ACM, 2010, pp. 47–55.

[80] E. G. Larsson, “MIMO detection methods: How they work,” *IEEE Signal Processing Magazine*, vol. 26, no. 3, pp. 91–95, 2009.

[81] D. Lee, M. Wolf, and H. Kim, “Design space exploration of the Turbo decoding algorithm on GPUs,” in *CASES'10*, 2010, pp. 217–226.

[82] S. Lee, C. Ahn, and S. Choi, “Implementation of Software-based 2X2 MIMO LTE Base station system using GPU,” in *SDR-WInnComm*, 2011.

[83] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “GPUWattch: enabling energy optimizations in GPGPUs,” *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 487–498, 2013.

[84] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *Computer Architecture Letters*, vol. 8, no. 2, pp. 48–51, 2009.

[85] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *35th International Symposium on Computer Architecture, 2008. ISCA'08.* IEEE, 2008, pp. 315–326.

[86] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A High-Performance DSP Architecture for Software-Defined Radio," *IEEE Micro*, vol. 27, no. 1, pp. 114–123, 2007.

[87] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. Flautner, "Design and Implementation of Turbo Decoders for Software Defined Radio," in *IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS)*, 2006, pp. 22–27.

[88] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *ACM Sigplan Notices*, vol. 44, no. 3. ACM, 2009, pp. 205–216.

[89] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *2011 38th Annual International Symposium on Computer Architecture (ISCA).* IEEE, 2011, pp. 319–330.

[90] D. Meisner and T. F. Wenisch, "DreamWeaver: architectural support for deep sleep," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 313–324.

[91] P. Micikevicius, "Multi-GPU Programming," in *GPU Technology Conference*, 2012.

[92] "Long Term Evolution (LTE)," White Paper, Motorola, Apr. 2011.

[93] Y.-H. Nam, L. Liu, Y. Wang, C. Zhang, J. Cho, and J.-K. Han, "Cooperative communication technologies for LTE-advanced," in *2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP).* IEEE, 2010, pp. 5610–5613.

[94] N. S. Network, “Mobile broadband with HSPA and LTE-capacity and cost aspects,” *Nokia Siemens Network, White Paper*, 2010.

[95] “NVIDIA GeForce GTX 680–The fastest, most efficient GPU ever built,” White Paper, NVIDIA.

[96] “Next Generation CUDA Compute Architecture: Fermi,” White Paper, NVIDIA, 2009.

[97] T. Nyländen, J. Janhunen, O. Silvén, and M. Juntti, “A GPU implementation for two MIMO-OFDM detectors,” in *2010 International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2010, pp. 293–300.

[98] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY: McGraw-Hill, 2008.

[99] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No power struggles: Coordinated multi-level power management for the data center,” in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 48–59.

[100] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, “Complete Computer System Simulation: the SimOS Approach,” *IEEE Parallel Distributed Technology: Systems Applications*, vol. 3, no. 4, pp. 34–43, 1995.

[101] M. Sjalander, S. McKee, P. Brauer, D. Engdal, and A. Vajda, “An LTE Uplink Receiver PHY Benchmark and Subframe-based Power Management,” in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2012, pp. 25–34.

[102] D. C. Snowdon, S. Ruocco, and G. Heiser, “Power management and dynamic voltage scaling: Myths and facts,” in *Proceedings of the 2005 workshop on power aware real-time computing*, vol. 12, 2005.

[103] K. Spafford, J. Meredith, and J. S. Vetter, “Quantifying NUMA and contention effects in multi-GPU systems,” in *GPGPU-4 Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, 2011, pp. 11:1 –11:7.

[104] X.-F. Tao, Y.-Z. Hou, K.-D. Wang, H.-Y. He, and Y. J. Guo, "GPP-based soft base station designing and optimization," *Journal of Computer Science and Technology*, vol. 28, no. 3, pp. 420–428, 2013.

[105] S. Ten Brink, J. Speidel, and R.-H. Han, "Iterative demapping for QPSK modulation," *Electronics Letters*, vol. 34, no. 15, pp. 1459–1460, 1998.

[106] "LTE emerges as early leader in 4G technologies," White Paper, Texas Instruments, 2009.

[107] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering Energy Proportionality with Non Energy-Proportional Systems-Optimizing the Ensemble." *HotPower*, vol. 8, pp. 2–2, 2008.

[108] "The Vanu Anywave Base Station Subsystem," Vanu Inc., Apr. 2006.

[109] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[110] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electronics letters*, vol. 36, no. 23, pp. 1937–1939, 2000.

[111] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, and K. Flautner, "From SODA to Scotch: The Evolution of a Wireless Baseband Processor," in *41st IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*, 2008, pp. 152–163.

[112] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime Anywhere Anyway Signal Processing," *IEEE Micro*, vol. 30, no. 1, pp. 81–91, 2010.

[113] J. Wu, Z. Zhang, Y. Hong, and Y. Wen, "Cloud radio access network (C-RAN): a primer," *Network, IEEE*, vol. 29, no. 1, pp. 35–41, 2015.

[114] M. Wu, Y. Sun, G.Wang, and J. Cavallaro, "Implementation of a High Throughput 3GPP Turbo Decoder on GPU," *Journal of Signal Processing Systems*, vol. 65, pp. 171–183, 2011.

[115] "LTE Baseband Targeted Design Platform," Xilinx, 2011.

[116] D. Yoge and N. Chandrachoodan, "GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications," in *25th International Conference on VLSI Design*, Jan. 2012, pp. 149–154.

[117] 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.

[118] 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.