GPF: A GPU-based Design to Achieve ~100 μs Scheduling for 5G NR

Yan Huang Shaoran Li Y. Thomas Hou Wenjing Lou
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA
{huangyan,shaoran,thou,wjlou}@vt.edu

ABSTRACT

5G New Radio (NR) is designed to operate under a broad range of frequency spectrum and to support new applications with extremely low latency. To support its diverse operating conditions, a set of different OFDM numerologies has been defined in the standards body. Under this numerology, it is necessary to perform scheduling with a time resolution of ~100 μs. This requirement poses a new challenge that does not exist in LTE and cannot be supported by any existing LTE schedulers. In this paper, we present the design of GPF – a GPU-based proportional fair (PF) scheduler that can meet the ~100 μs time requirement. The key ideas include decomposing the scheduling problem into a large number of small and independent sub-problems and selecting a subset of sub-problems from the most promising search space to fit into a GPU platform. By implementing GPF on an off-the-shelf Nvidia Quadro P6000 GPU, we show that GPF is able to achieve near-optimal performance while meeting the ~100 μs time requirement. GPF represents the first successful design of a GPU-based PF scheduler that can meet the new time requirement in 5G NR.

CCS CONCEPTS

• **Networks → Mobile Networks**;

KEYWORDS

5G NR, optimization, real-time, GPU, resource scheduling.

1 INTRODUCTION

As the next-generation cellular communication technology, 5G New Radio (NR) aims to cover a wide range of service cases, including broadband human-oriented communications, time-sensitive applications with ultra-low latency, and massive connectivity for Internet of Things [4]. With its broad range of operating frequencies from sub-GHz to 100 GHz [8, 9], the channel coherence time for NR varies greatly.[1] Compared to LTE, which typically operates on bands lower than 3 GHz [15] and with a coherence time over 1 ms, NR is likely to operate on higher frequency range (e.g., 3 to 6 GHz), with much shorter coherence time (e.g., ~200s μs). Further, from application's perspective, 5G NR is expected to support applications with ultra-low latency [12] (e.g., augmented/virtual reality [16, 17], autonomous vehicles [18]), which may require millisecond or even sub-millisecond scale delay or response time.

With such diverse service cases and channel conditions, the air interface design of NR must be much more flexible and scalable than that of LTE [1]. To address such needs, a number of different OFDM numerologies are defined for NR [6], allowing a wide range of frequency and time granularities for data transmission (see Table 1). Instead of a single transmission time interval (TTI) setting of 1 ms as for LTE, NR allows 4 numerologies (0, 1, 2, 3) for data transmission [10],[2] with TTI varying from 1 ms to 125 μs [5]. In particular, numerology 3 allows NR to cope with extremely short coherence time and to meet the stringent requirement in ultra-low latency applications, where the scheduling resolution is ~100 μs.

But the new ~100 μs time requirement also poses a new technical challenge to the design of an NR scheduler. To concretize our discussion, we use the most popular *proportional-fair* (PF) scheduling as an example [25–28]. Within each scheduling time interval (i.e., a TTI), a PF scheduler needs to decide how to allocate frequency-time *resource blocks* (RBs) to users and determine *modulation and coding scheme* (MCS) for each user. The objective of a PF scheduler is to maximize the sum of logarithmic (long-term) average data rates of all users. An important constraint is that each user can only use one MCS (from a set of allowed MCSs) across all RBs

MobiCom '18, October 29-November 2, 2018, New Delhi, India

ACM ISBN 978-1-4503-5903-0/18/10…$15.00
https://doi.org/10.1145/3241539.3241552

[1] A widely-used definition of coherence time is given by $T_c = 9/16\pi f_m$, where $f_m = v/\lambda$ is the maximum Doppler shift, v is the user speed and λ is the carrier wave length [19]. For instance, with v = 120 km/h, on 6 GHz spectrum we have $T_c \approx 260$ μs.

[2] Numerology 4 is used for control signaling.

that are allocated to her. This problem is found to be NP-hard [26–28] and has been widely studied in the literature. Although some of the existing PF schedulers could offer a scheduling solution on a much larger time scale, none of them can offer a solution close to 100 μs. In [25], Kwan *et al.* formulated the PF scheduling problem as an integer linear programing (ILP) and proposed to solve it using branch-and-bound technique, which has exponential computational complexity. Some polynomial-time PF schedulers that were designed using efficient heuristics can be found in [26–28]. We will examine computational complexity and, more importantly, real-time computational time of these schedulers in Section 4. A common feature of these PF schedulers (designed for LTE) is that they are all of sequential designs and need to go through a large number of iterations to determine a solution. Although they may meet the scheduling time requirement for LTE (1 ms), none of them comes close to meet the new ~100 μs time requirement for 5G NR.

In this paper, we present a novel design of a PF scheduler using off-the-shelf GPU to achieve ~100 μs scheduling resolution. We call our design "GPF", which is the abbreviation of GPU-based PF scheduler. Key ideas of GPF are: (i) to decompose the original scheduling problem into a large number of small and independent sub-problems with similar structure, where each sub-problem can be solved within very few number of iterations; (ii) to identify the promising search space through intensification and select a subset of sub-problems to fit into the processing cores of a GPU platform through random sampling. The contributions in this work can be summarized as follows:

- This paper presents the first design of a PF scheduler for 5G NR that can meet ~100 μs time requirement. This design can support NR numerology 0 to 3, which are to be used for data transmission. This is also the first scheduler design (for cellular networks) that exploits GPU platform. In particular, our design only uses a commercial off-the-shelf GPU platform and does not require any expensive custom-designed hardware.
- Our GPU-based design is based on a decomposition of the original optimization problem into a large number of sub-problems through enumerating MCS assignments for all users. We show that for each sub-problem (with a given MCS assignment), the optimal RB allocation problem can be solved exactly and efficiently.
- To reduce the number of sub-problems and fit them into a GPU, we identify the most promising search space among all sub-problems by using an intensification technique. We select a subset of sub-problems in the promising search space through a simple random sampling. We show that such an approach can find a near-optimal (if not optimal) solution.
- We implement GPF on an off-the-shelf Nvidia Quadro P6000 GPU using the CUDA programming model. By optimizing the operations performed on the processing cores of the given GPU, minimizing memory access time on the GPU based on differences in memory types and locations, and reducing iterative operations by exploiting techniques such as parallel reduction, etc., we are able to achieve overall GPF's scheduling time to ~100 μs for a user population size of up to 100 for an NR macro-cell.
- We conduct extensive experiments to investigate the performance of GPF and compare it with three representative PF schedulers (designed for LTE). Experimental results show that GPF can achieve near-optimal performance (per PF criterion) in ~100 μs while the other schedulers require significantly more time (ranging from many times to several orders of magnitude) and none of them comes close to meet the 100 μs time requirement.
- By breaking down the time performance between data movement (host to/from GPU) and computation in GPU, we show that between 50% to 70% (depending on user population size) of the overall time is spent on data movement while less than half of the time is spent on GPU computation. This suggests that our GPF scheduler can achieve even better performance (e.g., < 50 μs) if a customized GPU system (e.g., with enhanced bus interconnection such as the NVLink [46], or integrated host-GPU architecture [47–49]) is used for 5G NR base stations (BSs).

In the literature, there have been a number of studies applying GPUs to networking [29–31] and signal processing for wireless communications [32–34]. The authors of [29] proposed PacketShader, which is a GPU-based software router that utilizes parallelism in packet processing for boosting network throughput. The work in [30] applied GPU to network traffic indexing and is able to achieve an indexing throughput over one million records per second. In [31], the authors designed a packet classifier that is optimized for GPU's memory hierarchy and massive number of cores. All these works focus on network packet processing, which is fundamentally different from the resource scheduling problem that we consider in this paper. Authors of [32] proposed a parallel soft-output MIMO detector for GPU implementation. In [33], the authors designed GPU-based decoders for LDPC codes. The work in [34] addressed the implementation of a fully parallelized LTE Turbo decoder on GPU. These studies address baseband signal processing and their approaches cannot be applied in solving a complex scheduling optimization problem, which is the focus of this paper.

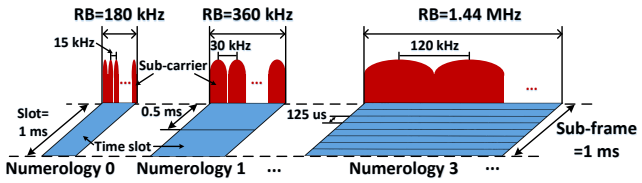


Figure 1: A frame structure of NR.

The rest of the paper is organized as follows. We provide a primer of NR air interface in Section 2. In Section 3, we formulate the classical PF scheduling problem in the context of NR frame structure. In Section 4, we highlight the challenge in designing a scheduler to meet the ~100 μs time requirement. In Section 5, we present our design ideas of GPF. The detailed implementation of GPF on an Nvidia Quadro P6000 GPU is given in Section 6. In Section 7, we conduct experimental study to validate the performance of GPF. Section 8 concludes the paper.

2 A PRIMER ON NR AIR INTERFACE

To meet diverse operating requirements, NR employs a much more flexible and scalable air interface than LTE [1]. A frame structure of NR is illustrated in Fig. 1. In the frequency domain, NR still employs OFDM and the bandwidth of an operating carrier is divided into a number of sub-carriers (SCs). In the time domain, each frame has 10 ms duration and consists of 10 sub-frames (SFs), each with 1 ms duration. An SF may consist of one or multiple time slots. The number of time slots in an SF is defined by OFDM numerologies [6]. Table 1 shows the SC spacing, number of time slots per SF, duration of each time slot and suitable frequency bands under each numerology. Since the number of OFDM symbols per slot is fixed to 14 in NR under different SC spacing [6], the duration of a time slot becomes shorter when SC spacing increases. Since numerology 4 is only used for control signaling [10], we will focus our discussion on numerology 0-3 (for data transmission) in this paper.

At the BS, each scheduling time interval (or scheduling resolution) is called transmission time interval (TTI), and its duration varies from several OFDM symbols (a mini-slot), one slot, to multiple slots. The choice of TTI depends on service and operational requirements [4]. In the frequency domain, the scheduling resolution is one RB, which consists of 12 consecutive SCs grouped together. Within each TTI, the base station (BS) needs to decide how to allocate (schedule) all the RBs for the next TTI to different users. Thus the channel coherence time should span at least two TTIs.

Although one RB within a TTI is allocated to a single user, a user may be allocated with multiple RBs. The next question is what modulation and coding scheme (MCS) to use for each user. For 5G NR, 29 MCSs are available [7], each representing a combination of modulation and coding techniques.[3] For a user allocated with multiple RBs, the BS must use the same MCS across all RBs allocated to this user [7].[4] This requirement is the same for LTE [14]. One reason is that using different MCSs on RBs cannot provide a significant performance gain, but results in additional signaling overhead [20]. For each user, the choice of MCS for its allocated RBs depends on channel conditions. A scheduling decision within each TTI entails joint RB allocation to users and MCS assignment for each user.

Table 1: OFDM numerologies in NR [2, 6].

Numerology	SC Spacing	Slots/SF	Slot Duration	Suitable Bands
0	15 kHz	1	1000 μs	$\leq$ 6 GHz
1	30 kHz	2	500 μs	$\leq$ 6 GHz
2	60 kHz	4	250 μs	$\leq$ 6 GHz
3	120 kHz	8	125 μs	$\leq$ 6 GHz or $\geq$ 24 GHz
4	240 kHz	16	62.5 μs	$\geq$ 24 GHz

3 A FORMULATION OF THE PF SCHEDULING PROBLEM

In this section, we present a formulation of the classical PF scheduling problem under the NR frame structure. Table 2 gives notation that we use in this paper.

3.1 Modeling and Formulation

Consider a 5G NR BS and a set $\mathcal{U}$ of users under its service. For scheduling at the BS, we focus on the downlink (DL) direction (data transmissions from BS to all users) and assume a full-buffer model, i.e., there is always data backlogged at the BS for each user. Denote W as the total bandwidth of the DL channel. Under OFDM, radio resource on this channel is organized as a two-dimensional frequency-time resource grid. In the frequency domain, the channel bandwidth is divided into a set $\mathcal{B}$ of RBs, each with bandwidth $W_0 = W/|\mathcal{B}|$. Due to frequency-selective channel fading, channel condition for a user varies across different RBs. For the same RB, channel conditions from the BS to different users also vary, due to the differences in their geographical locations. In the time domain, we have consecutive TTIs, each with a duration T_0. Scheduling at the BS must be completed within the current TTI (before the start of the next TTI).

[3] More precisely, 31 MCSs are defined, with 2 of them being reserved, leaving 29 MCSs available [7].

[4] In this paper, we consider one codeword per user. The analysis can be extended to cases where a user has two codewords by configuring the same MCS for both codewords.

Table 2: Notation

Symbol	Definition
$\mathcal{B}$	The set of RBs
I	The number of sub-problems solved by a thread block
K	The total number of sub-problems solved in each TTI
$\mathcal{M}$	The set of MCSs
N_c	The window size in number of TTIs used for PF scheduling
$q_u^b(t)$	The highest level of MCS that user u's channel can support on RB b in TTI t
$q_u^{\max}$	The highest level of MCS that user u's channel can support among all RBs
Q_u^d	The set of d MCS levels near $q_u^{\max}$ (inclusive)
Q^d	The Cartesian of sets $Q_1^d, Q_2^d, \cdots, Q_{\lvert\mathcal{U}\rvert}^d$
r^m	The per RB achievable data rate with MCS m
$r_u^{b,m}(t)$	The instantaneous achievable data rate of user u on RB b with MCS m in TTI t
$R_u(t)$	The aggregate achievable data rate of user u in TTI t
$\bar{R}_u$	The long-term average data rate of user u
$\bar{R}_u(t)$	The exponentially smoothed average data rate of user u up to TTI t
T_0	The duration of a TTI
$\mathcal{U}$	The set of users
W	Total DL channel bandwidth
W_0	$= W/\lvert\mathcal{B}\rvert$, bandwidth of an RB
$x_u^b(t)$	A binary variable indicating whether or not RB b is allocated to user u in TTI t
$y_u^m(t)$	A binary variable indicating whether or not MCS m is used for user u in TTI t
$z_u^{b,m}(t)$	An RLT variable in OPT-R

Denote $x_u^b(t) \in \{0,1\}$ as a binary variable indicating whether or not RB $b \in \mathcal{B}$ is allocated to user $u \in \mathcal{U}$ in TTI t, i.e.,

$$x_u^b(t) = \begin{cases} 1, & \text{if RB } b \text{ is allocated to user } u \text{ in TTI } t, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Since each RB can be allocated to at most one user, we have:

$$\sum_{u \in \mathcal{U}} x_u^b(t) \le 1, \quad (b \in \mathcal{B}). \tag{2}$$

At the BS, there is a set $\mathcal{M}$ of MCSs that can be used by the transmitter for each user $u \in \mathcal{U}$ in TTI t. When multiple RBs are allocated to the same user, then the same MCS, denoted by m ($m \in \mathcal{M}$), must be used across all these RBs. Denote $y_u^m(t) \in \{0,1\}$ as a binary variable indicating whether or not MCS $m \in \mathcal{M}$ is used by the BS for user $u \in \mathcal{U}$ in TTI t, i.e.,

$$y_u^m(t) = \begin{cases} 1, & \text{if MCS } m \text{ is used for user } u \text{ in TTI } t, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Since only one MCS from $\mathcal{M}$ can be used by the BS for all RBs allocated to a user $u \in \mathcal{U}$ at t, we have:

$$\sum_{m \in \mathcal{M}} y_u^m(t) \le 1, \quad (u \in \mathcal{U}). \tag{4}$$

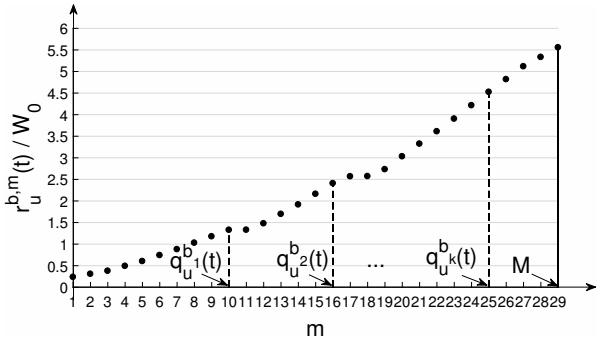


Figure 2: Spectral efficiencies corresponding to different levels of MCS. Data are from Table 5.1.3.1-1 in [7], with MCS levels 17 and 18 exchanged.

For user $u \in \mathcal{U}$ and RB $b \in \mathcal{B}$, the achievable data rate for this RB can be determined by Fig. 2. In this figure, $M = |\mathcal{M}|$ is the maximum number of MCSs allowed in the standard. It represents the most efficient MCS under the best channel condition and thus corresponds to the maximum data rate. For example, for MCSs in 5G NR, M can be 29 and the corresponding data rate per RB is $5.5547W_0$ [7]. Under best channel condition, any $m \le M$ can be supported on this RB for transmission. When channel condition is less than perfect, things become a bit complicated. Denote $q_u^b(t)$ as the highest level of MCS that can be supported by user u's channel on RB b in TTI t. $q_u^b(t)$ is determined by the the channel quality indicator (CQI) that is in the feedback report by user u at TTI $t-1$. Since M is the maximum value for $q_u^b(t)$, we have $q_u^b(t) \le M$. For a given $q_u^b(t)$, any MCS level from $\{1, 2, \cdots, q_u^b(t)\}$ can be supported by user u's channel on RB b in TTI t. On the other hand, if $q_u^b(t) < M$ and the BS chooses a MCS level $m > q_u^b(t)$ for user u (i.e., beyond the highest MCS level on RB b), then the achievable data rate of user u on RB b drops to zero, due to severe bit error [25, 28]. Denote $r_u^{b,m}(t)$ as user u's instantaneous achievable data rate on RB b with MCS m in TTI t. Then we have

$$r_u^{b,m}(t) = \begin{cases} r^m, & \text{If } m \le q_u^b(t), \\ 0, & \text{If } m > q_u^b(t). \end{cases} \tag{5}$$

Recall that for user $u \in \mathcal{U}$, the BS must use the same MCS mode $m \in \mathcal{M}$ across all RBs allocated to this user. As an example (shown in Fig. 2), suppose there are k RBs (denoted by $b_1, b_2, \cdots, b_k$) allocated to user u. Without loss of generality, suppose $q_u^{b_1}(t) < q_u^{b_2}(t) < \cdots < q_u^{b_k}(t) \le M$. Then there is a trade-off between the chosen MCS m and the subset of RBs that contribute nonzero data rates. That is, if $m_1 \le q_u^{b_1}(t)$, then all RBs will contribute data rate r^{m_1}; if $q_u^{b_1}(t) < \cdots < q_u^{b_i}(t) = m_2 < q_u^{b_{i+1}}(t) < \cdots < q_u^{b_k}(t)$, then only RBs $b_i, b_{i+1}, \cdots, b_k$ will contribute data rate r^{m_2}.

Let $R_u(t)$ denote the aggregate achievable data rate of user u in TTI t. Under a given scheduling decision (consisting of RB allocation as specified in (1) and MCS assignment in (3)), $R_u(t)$ can be computed as follows:

$$R_u(t) = \sum_{b\in\mathcal{B}} x_u^b(t) \sum_{m\in\mathcal{M}} y_u^m(t) r_u^{b,m}(t). \tag{6}$$

3.2 PF Objective Function

We now describe the formulation of the PF objective function. Let $\tilde{R}_u$ denote the long-term average data rate of user u (averaged over a sufficiently long time period). A widely-used objective function for PF is $\sum_{u\in\mathcal{U}} \log \tilde{R}_u$ [23, 26]. It represents a trade-off between total throughput and fairness among users. To maximize the PF objective when scheduling for each TTI t, a common approach is to maximize the metric

$$\sum_{u\in\mathcal{U}} \frac{R_u(t)}{\tilde{R}_u(t-1)} \tag{7}$$

during TTI $(t-1)$ and use the outcome of the decision variables for scheduling TTI t [23, 24, 26, 27], where $R_u(t)$ is the scheduled data rate to user u for TTI t (which can be calculated in (6)) and $\tilde{R}_u(t-1)$ is user u's exponentially smoothed average data rate up to TTI $(t-1)$ over a window size of N_c TTIs, and is updated as:

$$\tilde{R}_u(t-1) = \frac{N_c-1}{N_c}\tilde{R}_u(t-2) + \frac{1}{N_c}R_u(t-1)\ . \tag{8}$$

It has been shown that such real-time (per TTI) scheduling algorithm can approach optimal PF objective value asymptotically when $N_c \to \infty$ [23].

In this paper, we adopt this widely-used PF objective function for our scheduler. Putting (6) into (7), we have:

$$\sum_{u\in\mathcal{U}} \frac{R_u(t)}{\tilde{R}_u(t-1)} = \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} x_u^b(t) y_u^m(t). \tag{9}$$

In (9), $r_u^{b,m}(t)$ and $\tilde{R}_u(t-1)$ are input parameters and $x_u^b(t)$ and $y_u^m(t)$ are decision variables.

3.3 Problem Formulation

Based on the above discussions, the PF scheduling optimization problem for TTI t can be formulated as:

OPT-PF

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} x_u^b(t) y_u^m(t)$$

subject to RB allocation constraints: (2),

MCS assignment constraints: (4),

$x_u^b(t) \in \{0,1\}, \quad (u\in\mathcal{U}, b\in\mathcal{B}.)$

$y_u^m(t) \in \{0,1\}. \quad (u\in\mathcal{U}, m\in\mathcal{M}.)$

In OPT-PF, $r_u^{b,m}(t)$ is a constant for a given $u\in\mathcal{U}$, $b\in\mathcal{B}$, $m\in\mathcal{M}$ and $q_u^b(t)$. Recall that $q_u^b(t)$ is a constant and determined by the CQI in user u's feedback report at TTI $(t-1)$, which we assume is available by the design of an NR cellular network. $\tilde{R}_u(t-1)$ is also a constant as it is calculated in TTI $(t-1)$ based on $\tilde{R}_u(t-2)$ (available at TTI $(t-1)$) and $R_u(t-1)$ (the outcome of the scheduling decision at TTI $(t-2)$. The only variables here are $x_u^b(t)$ and $y_u^m(t)$ $(u\in\mathcal{U},\ b\in\mathcal{B},\ m\in\mathcal{M})$, which are binary integer variables. Since we have product terms $x_u^b(t)\cdot y_u^m(t)$ (nonlinear) in the objective function, we employ the *Reformulation-Linearization Technique* (RLT) [35, 36] to linearize the problem.

To do this, define $z_u^{b,m}(t) = x_u^b(t)\cdot y_u^m(t)$ $(u\in\mathcal{U}, b\in\mathcal{B}, m\in\mathcal{M})$. Since both $x_u^b(t)$ and $y_u^m(t)$ are binary variables, $z_u^{b,m}(t)$ is also a binary variable and must satisfy the following RLT constraints:

$$z_u^{b,m}(t) \le x_u^b(t), \quad (u\in\mathcal{U}, b\in\mathcal{B}, m\in\mathcal{M}), \tag{10}$$

$$z_u^{b,m}(t) \le y_u^m(t), \quad (u\in\mathcal{U}, b\in\mathcal{B}, m\in\mathcal{M}). \tag{11}$$

By replacing $x_u^b(t)\cdot y_u^m(t)$ with $z_u^{b,m}(t)$ and adding RLT constraints, we have the following reformulation for OPT-PF, which we denote by OPT-R:

OPT-R

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}(t)}{\tilde{R}_u(t-1)} z_u^{b,m}(t)$$

subject to RB allocation constraints: (2),

MCS assignment constraints: (4),

RLT constraints: (10), (11),

$x_u^b(t) \in \{0,1\}, \quad (u\in\mathcal{U}, b\in\mathcal{B})$

$y_u^m(t) \in \{0,1\}, \quad (u\in\mathcal{U}, m\in\mathcal{M})$

$z_u^{b,m}(t) \in \{0,1\}. \quad (u\in\mathcal{U}, b\in\mathcal{B}, m\in\mathcal{M})$

OPT-R is an ILP since all variables are binary and all constraints are linear. Commercial optimizers such as the IBM CPLEX [41] can be employed to obtain optimal solution to OPT-R (optimal to OPT-PF as well), which will be used as a performance benchmark for the scheduler design. Note that ILP is NP-hard in general and is consistent to the fact that the PF scheduling problem is NP-hard [26–28].

4 THE REAL-TIME CHALLENGE

Although it is possible to design an algorithm to find a near-optimal solution to OPT-R, it remains an open problem to find a near-optimal solution in real-time. By real-time, we mean that one needs to find a scheduling solution for TTI t during TTI $(t-1)$. For 5G NR, we are talking about on the order of ~100 μs for a TTI, which is much shorter than a

scheduling time interval under 4G LTE. This requirement comes from the fact that the shortest slot duration allowed for data transmission in NR is 125 μs under numerology 3 (refer to Sec. 2). To the best of our knowledge, none of existing scheduling algorithms can solve the PF scheduling problem within a time interval of ~100 μs. As such, this is the first design that breaks this technical barrier.

To design a ~100 μs PF scheduler for 5G NR, it is important to first understand why existing LTE schedulers are unable to meet such time requirement. PF schedulers designed for LTE can be classified into two categories: 1) metric-based schemes (typically implemented in industry-grade schedulers) that only address RB allocation [21, 22], and 2) polynomial-time approximation algorithms that address both RB allocation and MCS assignment [26–28].

Basically, simple metric-based schedulers such as those surveyed in [21, 22] allocate RBs to users in each TTI by comparing per-user metrics (e.g., the ratio between instantaneous rate and past average rate) on each RB. These schedulers do not address the assignment of MCS. In a BS, an independent adaptive modulation and coding (AMC) module is in charge of assigning MCS for each user [20]. Therefore, metric-based schedulers cannot be used to solve problem OPT-PF. From optimization's perspective, such a decoupled approach cannot achieve near-optimal performance and will result in a loss of spectral efficiency.

In the literature, there have been a number of polynomial-time heuristics designed for LTE PF scheduling. A common feature of these algorithms is that they are all of sequential designs. Their executions involve a large number of iterations. In this paper, we select several state-of-the-art LTE PF schedulers for performance comparison, including algorithms *Alg1* and *Alg2* from [26], the *Unified Scheduling* algorithm from [27], and the *Greedy* algorithm from [28]. Specifically, *Alg1* and *Alg2* first determine the RB allocation without considering constraints of single MCS per user, and then fix conflicts of multiple MCSs per user by selecting the best MCS for each user given the RB allocation. The computational complexity of *Alg1* and *Alg2* is $O(|\mathcal{U}||\mathcal{B}||\mathcal{M}|)$. The *Unified Scheduling* algorithm selects user with its associated MCS and adjusts RB allocation iteratively, until a maximum number of $\bar{K}$ users are scheduled in a TTI. It has a complexity of $O(\bar{K}|\mathcal{U}||\mathcal{B}||\mathcal{M}|)$. The *Greedy* algorithm employs a similar iterative design and can support scheduling over multiple carriers. It does not restrict the number of scheduled users per TTI and thus has a complexity of $O(|\mathcal{U}|^2|\mathcal{B}||\mathcal{M}|)$ for scheduling on a single carrier.

Alg1 and *Alg2* are the fastest among these sequential schedulers. Consider a practical NR cell with 100 users, 100 RBs, and 29 levels of MCS. The number of iterations that *Alg1* and *Alg2* need to go through is roughly 2.9×10^5. Our implementation of *Alg1* on a desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz) shows that the computational time of *Alg1* is beyond 800 μs (see Sec. 7 for additional experimental results).

The reason why existing PF schedulers cannot meet ~100 μs time requirement is that they are sequential algorithms involving a large number of iterations (as shown above). Although the use of additional CPU cores may help (by utilizing instruction-level parallelism, e.g., pipelining [40]), the potential reduction in computational time remains unclear.

5 A DESIGN OF A REAL-TIME SCHEDULER ON GPU PLATFORM

5.1 Basic Idea and Roadmap

The basic idea in our design is to decompose the original problem OPT-PF into a large number of mutually independent and small sub-problems, with the solution to each sub-problem being a feasible solution to the original problem. Then we can determine the output scheduling solution through solving sub-problems *in parallel* and finding the sub-problem solution that achieves the maximum objective value. We propose to employ GPU to solve these sub-problems, as a GPU typically consists of a large number (1000s) of processing cores and is highly optimized for massive parallel computation.

To make this idea work, we need to address the following two questions.

- (i) How to decompose the original problem so that the obtained sub-problems are independent and can be solved in parallel in real-time (within ~100 μs)?
- (ii) How to fit the large number of sub-problems into a given GPU platform?

The first question is directly tied to the time complexity of our scheduler. To meet the time requirement of ~100 μs, each sub-problem must be solved in 10s of μs while all sub-problems are solved concurrently in parallel. Therefore, it is important that each sub-problem is small in size and requires very few (sequential) iterations to find a solution. Also, it is desirable that all sub-problems have the same structure and require the same number of iterations to find their solutions. We address the first question in Section 5.2.

The second question is to address the space limitation of a GPU platform. If a GPU had an infinite number of cores, then we can solve all sub-problems in parallel and there is no issue. Unfortunately, any GPU has a limited number of cores. Although such number is large for a modern GPU (e.g., 3840 CUDA cores in an Nvdia Quadro P6000 GPU), it is still much smaller than the total number of sub-problems. So we have to select a subset of sub-problems from a promising search space that has a high probability to produce optimal or near-optimal solutions and fit them into the given GPU cores. We address this question in Section 5.3.

In our design of GPF, we do not exploit channel correlations in either time or frequency domains. This is to ensure that GPF works in any (worst) operating conditions under 5G NR.

5.2 Decomposition

There exist a number of decomposition techniques for optimization problems, with each designed for a specific purpose. For example, in branch-and-bound method, a tree-based decomposition is used to break a problem into two sub-problems so as to intensify the search in a smaller search space [37]. In dynamic programming method, decomposition results in sub-problems that still require to be solved recursively [38]. These decompositions cannot be readily parallelized and implemented on GPU.

Our proposed decomposition aims to produce a large number of independent sub-problems with the same structure. Further, each sub-problem is small and simple enough so that GPU cores can complete their computations under 100 μs. In other words, our decomposition is tailored toward GPU architecture (massive number of cores, lower clock frequency per core, very few number of computations for each sub-problem). Such a decomposition can be done by fixing a subset of decision variables via enumerating all possibilities. Then for each sub-problem, we only need to determine the optimal solution to the remaining subset of variables.

To see how this can be done for our optimization problem, consider OPT-PF, i.e., the original problem formulation with two sets of variables x_u^b and y_u^m, $u \in \mathcal{U}$, $b \in \mathcal{B}$, $m \in \mathcal{M}$. For the ease of exposition, we omit the TTI index t. Recall that variables x_u^b's are for RB allocation (i.e., assigning each RB to a user) while y_u^m's are for MCS assignment (i.e., choosing one MCS from $\mathcal{M}$ for each user). So we can decompose either along x-variable or y-variable. If we decompose along x, then we would have $|\mathcal{U}|^{|\mathcal{B}|}$ sub-problems (since there are $|\mathcal{U}|$ ways to assign each RB and we have a total of $|\mathcal{B}|$ RBs). On the other hand, if we decompose along y, then we would have $|\mathcal{M}|^{|\mathcal{U}|}$ sub-problems (since there are $|\mathcal{M}|$ ways to assign MCS for a user and we have a total of $|\mathcal{U}|$ users). In this paper, we choose to decompose along y, considering that the "intensification" technique that we propose to use in Section 5.3 works naturally for such sub-problem structure.

Under a given y-variable assignment $y_u^m = Y_u^m$, $u \in \mathcal{U}$, $m \in \mathcal{M}$, where Y_u^m's are binary constants and satisfy constraint (4), i.e., $\sum_{m\in\mathcal{M}} Y_u^m = 1$, $u \in \mathcal{U}$, OPT-PF degenerates into the following sub-problem:

OPT(Y)

$$\text{maximize} \quad \sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}}{\tilde{R}_u} Y_u^m \cdot x_u^b$$

subject to RB allocation constraints: (1), (2)

In the objective function, for $\sum_{m\in\mathcal{M}} \frac{r_u^{b,m}}{\tilde{R}_u} Y_u^m$, only one term in the summation is nonzero, due to the MCS constraint (4) on Y_u^m. Denote the m for this nonzero Y_u^m by m_u^*. Then the objective function becomes $\sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} \cdot x_u^b$. By interchanging the two summation orders, we have $\sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{B}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} \cdot x_u^b = \sum_{b\in\mathcal{B}}\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} \cdot x_u^b$. OPT(Y) now becomes:

$$\text{maximize} \quad \sum_{b\in\mathcal{B}}\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$$

subject to RB allocation constraints: (1), (2)

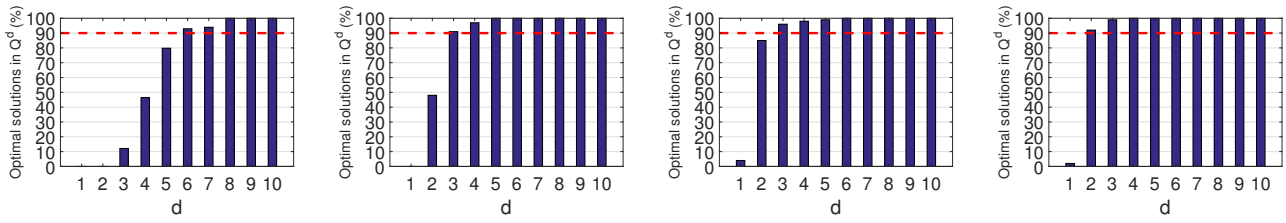
For a given $b \in \mathcal{B}$, there is only one term in the inner summation $\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$ that can be nonzero, due to the RB allocation constraint (2). So $\sum_{u\in\mathcal{U}} \frac{r_u^{b,m_u^*}}{\tilde{R}_u} x_u^b$ is maximized when the x_u^b corresponding to the largest $\frac{r_u^{b,m_u^*}}{\tilde{R}_u}$ across all users is set to 1 while others are set to 0. Physically, this means that the optimal RB allocation (under a given y-variable assignment) is achieved when each RB is allocated to a user that achieves the largest instantaneous rate normalized by its average rate.

We have just shown how to solve each sub-problem involving x-variable (RB allocation) under a given y-variable (MCS) assignment. The computational complexity of each sub-problem is $|\mathcal{B}||\mathcal{U}|$, if we solve it sequentially. Note that the sub-problem structure also allows us to perform optimal RB allocation in parallel for all RBs. In this case, the time complexity of a sub-problem can be reduced to $|\mathcal{U}|$ iterations that are used to search for the most suitable user for each RB (refer to Sec. 6).

5.3 Selection of Sub-Problems

After problem decomposition by enumerating all possible settings of y-variable, we have a total of $|\mathcal{M}|^{|\mathcal{U}|}$ sub-problems. This is too large to fit into the processing cores of a GPU. In this step, we will select a subset of K sub-problems through intensification and sampling and only search for the best solution among these K sub-problems, where the value K depends on the number of GPU cores that we have.

Intensification The first step in our selection of sub-problems is based on the intensification and diversification techniques from optimization (see, e.g., [39]). The basic idea is to break up the search space into promising and less promising subspaces in terms of finding optimal solution and devote search efforts mostly on the promising subspace (intensification). Even though there is a small probability that the optimal solution may lie in the less promising subspace, we can still be assured of getting high quality near-optimal solutions in the promising subspace. So the first question to



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 50$ **(c)** $|\mathcal{U}| = 75$ **(d)** $|\mathcal{U}| = 100$

Figure 3: Percentage of optimal solutions found in Q^d as a function of d under different user population sizes.

address is: what is the promising search subspace (among all possible y-variable settings) for optimal solution?

Recall that each user has $|\mathcal{M}|$ levels of MCS to choose from, with a higher level of MCS offering greater achievable data rate but also requiring a better channel quality. Recall for each RB $b \in \mathcal{B}$, q_u^b is the highest level of MCS that can be supported by user u's channel. Since q_u^b differs for different $b \in \mathcal{B}$, let's denote $q_u^{\max} = \max_{b\in\mathcal{B}} q_u^b$ as the highest level of MCS that user u's channel can support among all RBs. Then for each user $u \in \mathcal{U}$, it is safe to remove all MCS assignments with $m > q_u^{\max}$ (since such MCS assignments will have zero data rate for user u on all RBs) and we would not lose the optimal solution.

Among the remaining MCS settings for user u, i.e., $\{1, 2, \cdots, q_u^{\max}\}$, it is intuitive that the search space with MCS settings close to $q_u^{\max}$ is the most promising subspace for user u. To validate this idea, we conduct a numerical experiment using the CPLEX solver to solve OPT-R (not in real time) and examine the probability of success in finding the optimal solution as a function of the number of MCS levels near $q_u^{\max}$ (inclusive) for each user $u \in \mathcal{U}$. Specifically, denote

$$Q_u^d = \{m | \max\{1, q_u^{\max} - d + 1\} \le m \le q_u^{\max}\} \subset \mathcal{M} \quad (12)$$

as the set of d MCS levels near $q_u^{\max}$ (inclusive), where $d \in \mathbb{N}^*$ denotes the number of descending MCS levels from $q_u^{\max}$. For example, when $d = 1$, we have $Q_u^1 = \{m | m = q_u^{\max}\}$ for user u, meaning that user u will only choose its highest allowed MCS level $q_u^{\max}$; when $d = 2$, we have $Q_u^2 = \{m | q_u^{\max} - 1 \le m \le q_u^{\max}\}$ for user u, meaning that user u's MCS can be chosen between $q_u^{\max} - 1$ and $q_u^{\max}$. Across all $|\mathcal{U}|$ users, we define

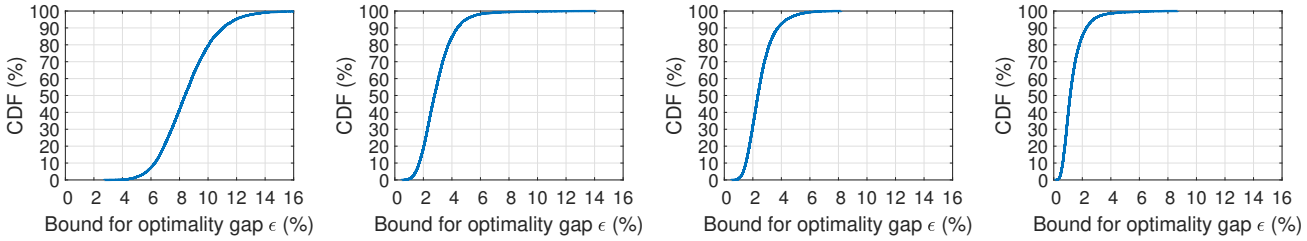
$$Q^d = Q_1^d \times \cdots \times Q_{|\mathcal{U}|}^d \subset \mathcal{M}^{|\mathcal{U}|} \quad (13)$$

as the Cartesian of sets $Q_1^d, Q_2^d, \cdots, Q_{|\mathcal{U}|}^d$. Clearly, Q^d contains MCS assignment vectors for all users where the MCS assigned for each user u is within its corresponding set Q_u^d.

In our experiment, we consider a BS with 100 RBs and a number of users ranging from 25, 50, 75, and 100. A set of 29 MCSs (see Fig. 2) can be used for each user. For a given number of users, we run experiments for 100 TTIs ($t = 1, 2, \cdots, 100$) with $N_c = 100$. For generality, we consider the scenario where there is no correlation in frequency, i.e., channel conditions (q_u^b's) vary independently across RBs for each user. Detailed experimental settings can be found in Section 7. Fig. 3 shows the percentage of optimal solutions in Q^d as a function of d under different user population sizes. For example, when $|\mathcal{U}| = 25$, 93% optimal solutions are within Q^6; when $|\mathcal{U}| = 75$, 96% optimal solutions are within Q^3.

Now we turn the table around and are interested in what d we should choose to meet a target probability of success in finding the optimal solution. Then Fig. 3 suggests that for a given success probability (say 90%), the value of d required to achieve this success probability decreases with the user population size ($d = 6$ for $|\mathcal{U}| = 25$, $d = 3$ for $|\mathcal{U}| = 50$ and 75, and $d = 2$ for $|\mathcal{U}| = 100$). This is intuitive, as for the same number of RBs, the greater the number of users we have, the fewer the number of RBs to be allocated to each user, leading to the need of fewer levels of MCS for selection. More importantly, Fig. 3 shows that for a target success probability (90%), we only need to set d to a small number and a corresponding small search space Q^d would be sufficient to achieve this probability.

For a target success probability, the optimal setting of d depends not only on the user population size $|\mathcal{U}|$ but also on users' channel conditions. For instance, as shown in Sec. 7.3, when there are frequency correlations among RBs, the optimal d may change. In a practical NR cell, optimal d under each user population size $|\mathcal{U}|$ should be set adaptively in an online manner to keep up with the changing channel conditions. Specifically, the BS frequently computes optimal solution to OPT-R based on users' CQI reports, and records the smallest d that contains the optimal solution along with the $|\mathcal{U}|$ at that time. Such computation can be done only for selected TTIs and there is no strict real-time requirement. Optimal values of d under different $|\mathcal{U}|$'s are re-calculated periodically based on recorded results through the statistical approach described above, and are maintained in a lookup table stored in the BS's memory. During run-time, the BS sets d adaptively based on the number of active users in the cell by simply looking up the table.



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 50$ **(c)** $|\mathcal{U}| = 75$ **(d)** $|\mathcal{U}| = 100$

Figure 4: CDFs of gaps (in percentage) between objectives of sub-problem solutions and the optimum.

Sampling After the intensification step, the number of sub-problems is still too large to fit into the processing cores of a GPU. For example, with Q^2 and 100 users, we have 2^{100} sub-problems. In the second step, we employ random sampling based on certain probability distribution to select K sub-problems.

The probability distribution used for random sampling can be set based on specific needs. In this work, we employ a simple uniform distribution as an example. Specifically, after narrowing down the promising subspace Q^d, for each of the K sub-problems that we choose, we set the MCS for each user u from Q_u^d following a uniform distribution (with equal probability). This is equivalent to sampling from Q^d with a uniform distribution. This construction (equivalent to sampling) of K sub-problems can be executed in parallel on GPU cores (refer to Sec. 6). After the K sub-problems are constructed, they can be solved in parallel and the best solution (with the maximum objective value) among them is chosen as the output scheduling solution for the next TTI.

One question to ask is whether such sampling of K sub-problems from the promising search subspace can ensure finding an optimal or near-optimal solution. In the following, we show that solutions of the K sub-problems (samples) will almost surely contain at least one near-optimal solution to OPT-PF (e.g., > 95% of optimum).

The science behind this claim is as follows. Denote a as the gap (in percentage) of a sample's objective value from the optimum. For a target bound for optimality gap ϵ, denote $p_{1-\epsilon}$ as the probability that a sample is $(1-\epsilon)$-optimal, i.e., $p_{1-\epsilon} = P(a \le \epsilon)$. The probability $p_{1-\epsilon}$ is the same for all K samples since they are taken from the same search subspace independently and following the same uniform distribution. Denote $P_{1-\epsilon}$ as the probability that at least one sample (among the K samples) is $(1-\epsilon)$-optimal. Since all samples are mutually independent, we have:

$$P_{1-\epsilon} = 1 - (1 - p_{1-\epsilon})^K. \tag{14}$$

Therefore, to have $P_{1-\epsilon} \ge 99.99\%$, i.e., with at least 99.99% probability of having one sample achieving $(1-\epsilon)$-optimal among the K samples, we must have

$$p_{1-\epsilon} \ge 1 - \sqrt[K]{1 - 99.99\%}. \tag{15}$$

As will be discussed in Sec. 6, the Nvidia Quadro P6000 GPU we use in implementation can solve $K = 300$ sub-problems independently in parallel. Thus from (15), we should have $p_{1-\epsilon} \ge 3.02\%$ to ensure $P_{1-\epsilon} \ge 99.99\%$. That is, as long as each sample has a small probability ($\ge 3.02\%$) of achieving $(1-\epsilon)$-optimal, then the probability that at least one sample (among 300 samples) achieves $(1-\epsilon)$-optimal is almost surely ($\ge 99.99\%$).

We now investigate the value of $p_{1-\epsilon}$ from our samples through experiments. The experimental settings are the same as those for Fig. 3, i.e., $|\mathcal{B}| = 100$, $|\mathcal{U}| \in \{25, 50, 75, 100\}$, and $|\mathcal{M}| = 29$. The parameter d is set to 6, 3, 3, and 2 for $|\mathcal{U}|$ =25, 50, 75, and 100, respectively. We run experiments for 100 TTIs with $N_c = 100$. For each TTI, we generate 100 samples from Q^d under each $|\mathcal{U}|$, and record gaps (a's) of their objective values from the optimum. Thus for each $|\mathcal{U}|$, we have 10,000 samples (over 100 TTIs), each with a corresponding a. Cumulative distribution functions (CDFs) for $a \le \epsilon$ for different $|\mathcal{U}|$'s are shown in Fig. 4. Coordinates of each point on these CDFs represent a given ϵ and its corresponding (empirical) probability $P(a \le \epsilon)$, i.e., $p_{1-\epsilon}$. To have $p_{1-\epsilon} \ge 3.02\%$, the value of ϵ must be at least 5.35%, 1.34%, 1.24%, 0.47% for $|\mathcal{U}|$ =25, 50, 75, and 100, respectively. That is, with 99.99% probability, at least one of the $K = 300$ samples selected in each TTI achieves up to 94.65%-, 98.66%-, 98.76%- and 99.53%-optimal for $|\mathcal{U}|$ =25, 50, 75, and 100, respectively. These experimental results show that random sampling based on intensification can indeed find near-optimal solution to problem OPT-PF.

When we construct the K sub-problems through random sampling, it is possible that we may get some identical samples. But it is easy to calculate that such probability is extremely small (as each sample consists of $|\mathcal{U}|$ MCS assignments). In fact, even if there are identical samples, it will not affect much on the final near-optimal performance because we still have a large number of samples to work with.

6 IMPLEMENTATION

In this section, we describe our implementation of the design ideas in Section 5 on commercial off-the-shelf GPU platform.

6.1 Why GPU

For the purpose of implementing our proposed solution in the previous section, there are a number of advantages of GPU over FPGA and ASIC. From hardware's perspective, GPU is much more flexible. By design, GPU is a general-purpose computing platform optimized for large-scale parallel computation. It can be used to implement different scheduling algorithms without any change on hardware. In contrast, FPGA is not optimized for massive parallel computation, while ASIC is made for a specific algorithm and cannot be changed or updated after the hardware is made. From software's perspective, commercial GPUs (e.g., those from Nvidia) come with highly programmable tool such as CUDA, which is capable of programming the behavior of each processing core. On the other hand, it is much more complicated to program the same set of functions in FPGA. Finally, in terms of cost and design cycle, the GPU platform that we use is off-the-shelf, which is readily available and at low cost (for NR BS). In contrast, the cost for making an ASIC could be orders of magnitude higher than GPU and it will take a considerable amount of time to develop.

6.2 Fitting Sub-Problems into a GPU

We use an off-the-shelf Nvidia Quadro P6000 GPU [42] and the CUDA programming platform [43]. This GPU consists of 30 streaming multi-processors (SMs). Each SM consists of 128 small processing cores (CUDA cores). These cores are capable of performing concurrent computation tasks involving arithmetic and logic operations.

Under CUDA, the K sub-problems considered by the scheduler per TTI are handled by a grid of thread blocks. An illustration of this implementation is given in Fig. 5. Since our Nvidia GPU has 30 SMs, we limit each SM to handle one thread block so as to avoid sequential execution of multiple thread blocks on an SM. Since the processing of each sub-problem requires $\max\{|\mathcal{B}|, |\mathcal{U}|\}$ threads (see Steps 1 and 2 in Fig. 5, more on this later) and a thread block can have a maximum of 1024 threads, the number of sub-problems that can be solved by each thread block is

$$I = \min\left\{\left\lfloor \frac{1024}{|\mathcal{B}|} \right\rfloor, \left\lfloor \frac{1024}{|\mathcal{U}|} \right\rfloor\right\}, \tag{16}$$

(refer to Sec. 6.3, Step 2). Thus, the total number of sub-problems that we can fit into an Nvidia Quadro P6000 GPU for parallel computation is $K = 30 \cdot I$. For example, for $|\mathcal{B}| = 100$ RBs and $|\mathcal{U}| = 100$ users, the GPU can solve $K = 300$ sub-problems in parallel.

6.3 Tasks and Processing in GPU

To find an optimal or near-optimal solution on GPU, we need to spend time for three tasks: (i) transfer the input data from the host (BS) memory to GPU's global memory; (ii) generate and solve $K = 30 \cdot I$ sub-problems with 30 thread blocks (one thread block per SM); (iii) transfer the final solution back to the host memory. In the rest of this section, we give details for each task.

Transferring Input Data to GPU Based on our discussion in Sec. 5.3, we only transfer input data associated with the promising search subspace $\mathcal{Q}^{d^*}$, where d^* depends on the user population size $|\mathcal{U}|$ and the target success probability of finding an optimal solution in $\mathcal{Q}^{d^*}$. For example, to achieve 0.9 success probability, we should set $d^* = 3$ if $|\mathcal{U}| = 50$ and $d^* = 2$ if $|\mathcal{U}| = 100$ (refer to Sec. 5.3). For each user u, only d^* MCS levels in $\mathcal{Q}_u^{d^*}$ will be considered in the search space. Note that even if with up to 0.1 probability (if we consider 0.9 success probability) we may miss the optimal solution in $\mathcal{Q}^{d^*}$, we can still find extremely good near-optimal solutions in $\mathcal{Q}^{d^*}$, which is shown in the last section. The input data that we need to transfer from the host (BS) memory to the GPU global memory include $r_u^{b,m}$'s (for $m \in \mathcal{Q}_u^{d^*}, u \in \mathcal{U}, b \in \mathcal{B}$) and $\tilde{R}_u$'s (for $u \in \mathcal{U}$). For example, with 100 users and 100 RBs, we have $d^* = 2$. Then the size of transferred data is equal to 80 KB for $r_u^{b,m}$'s plus 0.4 KB for $\tilde{R}_u$'s (with float data-type).

Generating and Solving K Sub-problems Within each SM, I ($K/30$) sub-problems are to be generated and solved with one thread block. Then the best solution among the I sub-problems is selected and sent to the GPU global memory. This is followed by a round of selection of best solution from the 30 SMs (with a new thread block). Fig. 5 shows the five steps in our design to complete this task. We describe each step as follows. Steps 1 to 4 are completed by each of the 30 thread blocks (SMs) in parallel. Step 5 follows after the completion of Step 4 across all 30 thread blocks and is done with a new thread block.

- *Step 1 (Generating Sub-Problems)* Each of the 30 thread blocks needs to first generate I sub-problems, where I is defined in (16). For each sub-problem, a MCS level for each user $u \in \mathcal{U}$ is randomly and uniformly chosen from the set $\mathcal{Q}_u^{d^*}$. Doing this in parallel requires $|\mathcal{U}|$ threads for each sub-problem. Thus, to parallelize this step for all I sub-problems, we need to use $I \cdot |\mathcal{U}| \leq 1024$ threads. Threads should be synchronized after this step to ensure that all sub-problems are successfully generated.
- *Step 2 (Solving Sub-Problems)* For each of the I sub-problems (i.e., with y-variable fixed), optimal RB allocation (x-variable) can be determined by solving OPT(Y)

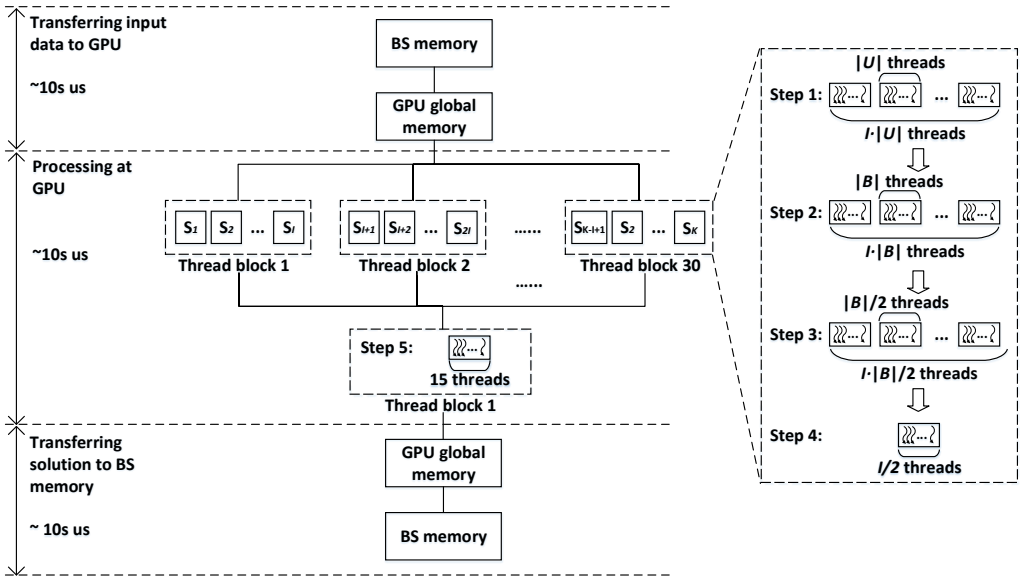


Figure 5: Major tasks and steps in our implementation on GPU.

as described in Sec. 5.2. For each sub-problem, the allocation of each RB $b \in \mathcal{B}$ to a user is done in parallel with $|\mathcal{B}|$ threads. With I sub-problems per block, we need $I \cdot |\mathcal{B}| \leq 1024$ threads for parallelizing this step.[5] Each thread needs to have input data for all users for comparison. Due to the small size of shared memory in an SM (only 96 KB per SM for Nvidia Quadro P6000 GPU), we cannot store the input data for all $|\mathcal{U}|$ users in an SM's shared memory (a part of the shared memory is reserved for other intermediate data). On the other hand, if we let the thread read out data for each user separately from the GPU global memory, it will result in $|\mathcal{U}|$ times of access to the global memory. In a GPU, access to the global memory is much slower than that to the shared memory in an SM. To address this problem, we put users in $\mathcal{U}$ into several sub-groups such that the input data for each sub-group of users can be read out from the global memory in one access and fit into an SM's shared memory. This will result in a major reduction in the number of times that is required for accessing global memory in this step. Once we have the input data for the sub-group of users in the shared memory, we let the thread find the most suitable user for the given RB within this sub-group. By performing these operations for each sub-group of users, a thread will find the optimal RB allocation for the sub-problem. A synchronization of all threads in a thread block is necessary after this step.

- *Step 3 (Calculating Objective Values).* Given the optimal RB allocation for each sub-problem in Step 2, we need to calculate the objective value under the current solution to each sub-problem. The calculation of objective value involves summation of $|\mathcal{B}|$ terms. To reduce the number of iterations in completing this summation, we employ *parallel reduction* technique [44]. Figure 6 illustrates this technique. We use $|\mathcal{B}|/2$ threads in parallel for each sub-problem and it only requires $\log_2(|\mathcal{B}|)$ iterations to complete the summation of $|\mathcal{B}|$ terms. A key in parallel reduction in shared memory is to make sure that threads are reading memory based on consecutive addressing. For I sub-problems, we need $I \cdot |\mathcal{B}|/2 \leq 1024$ threads for this step. Again, threads must be synchronized after this step is completed.
- *Step 4 (Finding the Best Solution in a Thread Block)* At the end of Step 3, we have I objective values in an SM corresponding to the I sub-problems. In this step, we need to find the best solution (with the maximum objective value) among the I sub-problem solutions. This is done through comparison, which again can be realized by parallel reduction. We need $I/2$ threads to parallelize this comparison. Once it's complete, we write the best solution along with its objective value into the GPU's global memory.
- *Step 5 (Finding the Best Solution Across All Thread Blocks).* After Steps 1 to 4 are completed by the 30 thread blocks

[5] Since $I \cdot |\mathcal{U}| \leq 1024$ and $I \cdot |\mathcal{B}| \leq 1024$, the maximum value that I can take is $\min\left\{\left\lfloor \frac{1024}{|\mathcal{B}|} \right\rfloor, \left\lfloor \frac{1024}{|\mathcal{U}|} \right\rfloor\right\}$, which is what we have in Eq. (16).

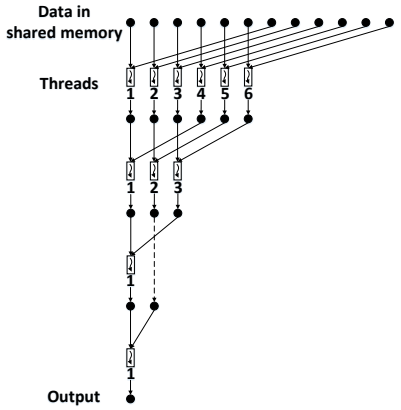


Figure 6: Parallel reduction in shared memory.

(SMs), we have 30 solutions (and their objective values) stored in the GPU's global memory, each corresponding to the best solution from its respective thread block. Then we create a new thread block (with 15 threads) to find the "ultimate" best from these 30 "intermediate" best solutions. Again, this step can be done through parallel reduction.

Transferring Output Solution to Host After we find the best solution in Step 5, we transfer this solution from the GPU's global memory back to the host's memory.

7 EXPERIMENTAL VALIDATION

7.1 Experiment Platform

Our experiment is done on a Dell desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz) and an Nvidia Quadro P6000 GPU. Data communications between CPU and GPU go through a PCIe 3.0 X16 slot with default configuration. Implementation on GPU is based on the Nvidia CUDA (version 9.1) programming platform. For performance comparison, we employ IBM CPLEX Optimizer (version 12.7.1) [41] to find optimal solution to OPT-R.

7.2 Settings

We consider an NR cell with a BS and a number of users. The user population size $|\mathcal{U}|$ is chosen from $\{25, 50, 75, 100\}$.[6] The number of available RBs is $|\mathcal{B}| = 100$. Assume that a set of $|\mathcal{M}| = 29$ MCSs shown in Fig. 2 is available to each user. Numerology 3 (refer to Table 1) of NR is considered, where the duration of a TTI is 125 μs. The PF parameter N_c is set to 100 TTIs. The full-buffer traffic model is employed.

For wireless channels, we consider the block-fading channel model for both frequency and time [45]. User mobility is captured by independent variations of channel conditions ($q_u^b(t)$'s) across TTIs. To model large-scale fading effect, the highest feasible MCS level across all RBs is higher for users that are closer to the BS and is lower for cell-edge users. For frequency-selective fading effect, we first consider the worst-case scenario where parameters $q_u^b(t)$'s across all RBs are uncorrelated and randomly generated for each user. Such setting is useful to examine the robustness of GPF under the extreme operating conditions. Then we consider the scenario where there is correlation in frequency.

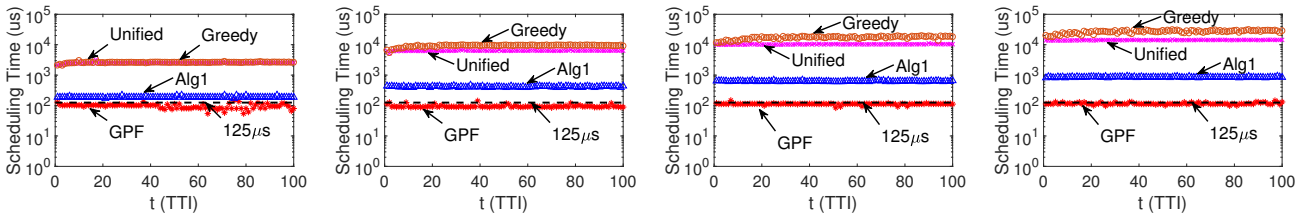
[6] $|\mathcal{U}| = 100$ covers most typical deployment scenarios considered by 3GPP (e.g., indoor hotspot, dense urban, rural, etc.) [4].

7.3 Results

In addition to the optimal solution obtained by CPLEX, we also include state-of-the-art PF schedulers *Alg1* from [26], the *Unified* algorithm from [27], and the *Greedy* algorithm from [28] in our performance comparison. We set the maximum number of scheduled users per TTI to 20 for the *Unified* algorithm. To optimize the performance of these algorithms, we implement them on CPU since they are designed for sequential execution.

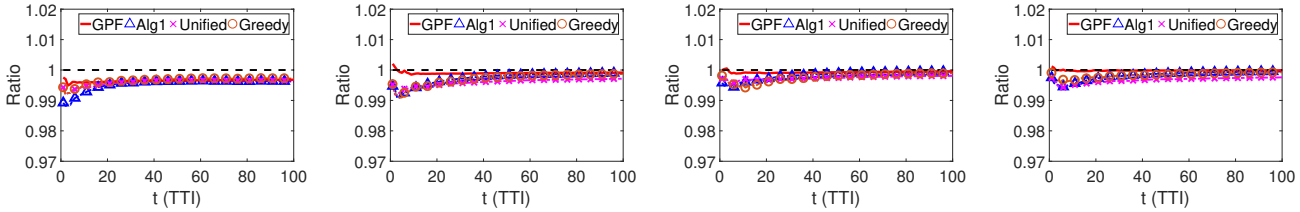
We first verify that GPF can meet the time requirement of ~100 μs, which is the major purpose of this work. We consider the worst-case scenario where there is no frequency correlation, i.e., $q_u^b(t)$'s change independently across RBs. Based on the results from Sec. 5.3, we set the parameter d^* for controlling the sampling subspace $\boldsymbol{Q}^{d^*}$ to 6, 3, 3 and 2 for $|\mathcal{U}|$=25, 50, 75 and 100, respectively. Results of scheduling time for 100 consecutive TTIs are shown in Fig. 7. Computation time of CPLEX is not shown in the figure since it is too large to fit in the scale of the figure. The average computation time of CPLEX is 3.20 s, 10.62 s, 18.17 s, and 30.23 s for $|\mathcal{U}|$ =25, 50, 75, and 100, respectively. We can see that under all $|\mathcal{U}|$'s, the scheduling time of GPF is within 125 μs in most cases. Specifically, mean values and standard deviations of scheduling time are 96.16 μs and 16.60 for $|\mathcal{U}| = 25$, 94.93 μs and 9.36 for $|\mathcal{U}| = 50$, 112.60 μs and 6.47 for $|\mathcal{U}| = 75$, and 116.21 μs and 8.22 for $|\mathcal{U}| = 100$. On the other hand, *Alg1*, which is the fastest among the state-of-the-art schedulers used in comparison, has a mean computation time of 189.7 μs for $|\mathcal{U}| = 25$, 416.6 μs for $|\mathcal{U}| = 50$, 630.8 μs for $|\mathcal{U}| = 75$, and 855.7 μs for $|\mathcal{U}| = 100$.

In Fig. 7, we notice that there are a few instances where GPF's scheduling time is beyond 125 μs. To understand what the scheduling time may entail, we investigate, through experiments, time spent at different execution stages in GPF, including transferring data from host to GPU, processing at GPU, and transferring solution from GPU back to host (refer to Sec. 6). Mean values and standard deviations of processing time in different stages with different user population sizes (each for 1000 TTIs) are shown in Table 3. The GPF computation time corresponds to GPU time overhead entry



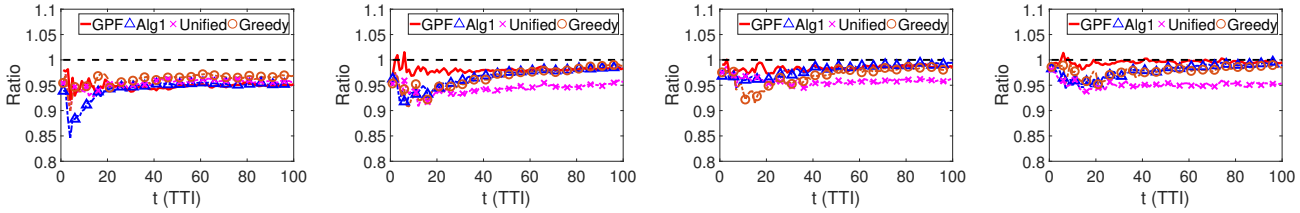
(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 50$ **(c)** $|\mathcal{U}| = 75$ **(d)** $|\mathcal{U}| = 100$

Figure 7: Scheduling time comparison between GPF and state-of-the-art PF schedulers. X axis corresponds to time (in unit of TTI).



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 50$ **(c)** $|\mathcal{U}| = 75$ **(d)** $|\mathcal{U}| = 100$

Figure 8: PF criterion comparison between GPF and state-of-the-art PF schedulers.



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 50$ **(c)** $|\mathcal{U}| = 75$ **(d)** $|\mathcal{U}| = 100$

Figure 9: Sum average cell throughput comparison between GPF and state-of-the-art PF schedulers.

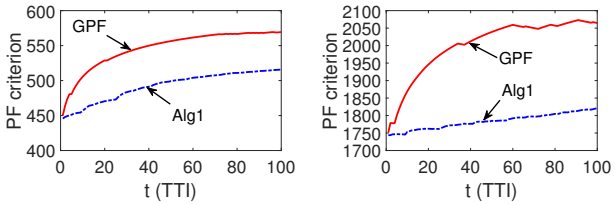
in Table 3. We find that the time spent for computing GPF scheduling solution is much shorter than 100 μs. It is in fact far below our design objective of 100 μs. On the other hand, time spent on data transferring between host and GPU takes a major share of total time overhead. Such data transferring operations take more than 60% of the total scheduling time. This is due to our use of off-the-shelf GPU without any customized design. Thus we conclude that the bottleneck of GPF execution time is on the communication between host and GPU. This overhead can be easily reduced by employing a (customized) integrated host-GPU system [47–49].

Next, we examine the performance of GPF using conventional metrics for PF schedulers, including: (i) the PF criterion $\sum_{u\in\mathcal{U}} \log_2(\tilde{R}_u(t))$ (the primary performance objective of a PF scheduler), and (ii) the sum average cell throughput $\sum_{u\in\mathcal{U}} \tilde{R}_u(t)$ (representing the spectral efficiency). PF criteria and sum throughput performance for 100 consecutive TTIs

Table 3: Breakdown of GPF's execution time consumed in different stages. Data format: (mean (μs), standard deviation).

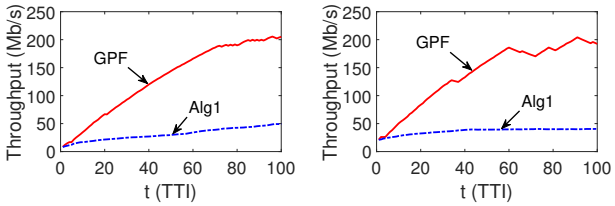
	$\|\mathcal{U}\| = 25$	$\|\mathcal{U}\| = 50$	$\|\mathcal{U}\| = 75$	$\|\mathcal{U}\| = 100$
H-to-G	(18.88, 4.62)	(18.23, 5.69)	(26.58, 3.82)	(25.27, 7.10)
GPU	(26.40, 2.74)	(26.83, 3.86)	(38.95, 1.46)	(48.00, 1.60)
G-to-H	(43.27, 11.36)	(51.06, 14.26)	(50.16, 5.97)	(46.85, 10.14)
Total	(88.55, 12.50)	(96.12, 14.73)	(115.70, 7.01)	(120.12, 12.34)

are shown in Fig. 8 and Fig. 9, respectively. In these figures, we take the ratio between the metric (PF or throughput) achieved by a scheduler and that achieved by optimal solution from CPLEX. Note that there are instances where the ratio is greater than one because CPLEX's solution is optimal with respect to the per-TTI objective (7), but not the long term PF criterion that we consider for comparison. Clearly,



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 100$

Figure 10: PF criterion comparison between GPF and state-of-the-art LTE PF scheduler.



(a) $|\mathcal{U}| = 25$ **(b)** $|\mathcal{U}| = 100$

Figure 11: Sum average cell throughput comparison between GPF and state-of-the-art LTE PF scheduler.

GPF achieves near-optimal performance and is no worse than all three LTE PF schedulers in all cases. GPF performs particularly well when the user population size is larger than or equal to 50.

We have also run experiments for scenarios with frequency correlation, where $q_u^b(t)$'s are identical within the same group of consecutive RBs and change (randomly) for a different group. Results with coherence bandwidth equal to 2 and 5 RBs (not shown here due to space limitation) indicate that optimal d's change with frequency correlations.[7] Specifically, when coherence bandwidth covers 2 RBs, optimal d's for $|\mathcal{U}|$ =25, 50, 75 and 100 are 5, 3, 3 and 2, respectively; when coherence bandwidth covers 5 RBs, optimal d's are 4, 3, 3 and 2, respectively. With adjusted settings of d, GPF achieves similar real-time and near-optimal performance as those shown in Figs. 7, 8 and 9.

Based on our experimental results, we conclude that GPF is able to meet NR's time requirement of ~100 μs for scheduling while achieving near-optimal performance.

7.4 Why Not Re-use LTE Scheduler

In LTE, the scheduling resolution is 1 ms since the duration of a TTI is fixed to 1 ms. It means that an LTE scheduler updates solution every 1 ms. To investigate whether or not LTE scheduler can be re-used for NR, we conduct an experiment with the following setting. Assume that the coherence time covers two slot durations under numerology 3 (likely to occur at high spectrum). We compare two schemes:

- *Scheme 1:* Update the scheduling solution every 8 slots (since 1 ms/125 μs = 8) by an LTE scheduler;
- *Scheme 2:* In each slot (125 μs), use GPF to compute the solution.[8]

We adopt *Alg1* for the LTE scheduler since it is able to find solution in 1 ms and is the fastest among the state-of-the-art PF schedulers (see Fig. 7). Results of two schemes for 100 consecutive TTIs under $|\mathcal{U}|$ = 25 and 100 are shown in Fig. 10 and Fig. 11. We can see that for both the PF criterion (in log scale) and the sum average cell throughput, GPF significantly outperforms *Alg1*, which means that existing PF schedulers perform poorly for 5G NR and thus cannot be re-used.

8 CONCLUSIONS

This paper presents the first successful design of a PF scheduler that can meet the ~100 μs time requirement for 5G NR. Our GPU-based design is based on a novel decomposition of the original optimization problem into a large number of small sub-problems (each requiring very few number of iterations to solve). By employing intensification and random sampling techniques, we are able to select a subset of sub-problems to match a given number of processing cores in a GPU for independent and parallel processing. We implement our GPF on an off-the-shelf Nvidia Quadro P6000 GPU using the CUDA programming model. Through extensive experiments and comparison study, we show that GPF can achieve near-optimal performance and meet the ~100 μs time requirement while none of existing state-of-the-art PF schedulers (designed for LTE) can do so. By analyzing time overhead information from experiments, we find that GPF has the potential to achieve even better timing performance if a custom-designed host-GPU system is employed. Our research opens the door for incorporating GPU computing into 5G BSs for the next generation cellular networks.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their constructive comments and suggestions, which have improved the quality and presentation of this paper. The authors also thank the anonymous shepherd for overseeing the revision process and approving the final version of the paper. This research was supported in part by NSF under grants 1800650, 1642873, 1617634, 1343222 and ONR DURIP grant N00014-15-1-2926.

[7] A widely-used definition of coherence bandwidth is given by $B_c \approx 1/5\sigma_\tau$, where σ_τ denotes the root mean square (RMS) delay spread [19]. For instance, following 3GPP's channel modeling [11], we have mean $\sigma_\tau \approx 90$ ns for the UMa LOS scenario on 6 GHz spectrum, and thus the coherence bandwidth $B_c \approx 2.22$ MHz, which covers 2 RBs under numerology 3.

[8] In the rare case when the total scheduling time of GPF is greater than a slot duration, reuse the previous solution.

REFERENCES

[1] Ericsson Technology Review, "5G new radio: Designing for the future." Available: https://www.ericsson.com/en/ericsson-technology-review/archive/2017/designing-for-the-future-the-5g-nr-physical-layer

[2] Qualcomm, "Making 5G NR a commercial reality." Available: https://www.qualcomm.com/media/documents/files/making-5g-nr-a-commercial-reality.pdf

[3] Z. E. Ankarali, B. Peköz, and H. Arslan, "Flexible radio access beyond 5G: A future projection on waveform, numerology, and frame design principles,?" *IEEE Access,* vol. 5, pp. 18295-18309, May 2017.

[4] 3GPP TR 38.913 version 14.3.0, "Study on scenarios and requirements for next generation access technologies." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996

[5] 3GPP TR 38.804 version 14.0.0, "Study on New Radio access technology; Radio interface protocol aspects." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3070

[6] 3GPP TS 38.211 version 15.0.0, "NR; Physical channels and modulation." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[7] 3GPP TS 38.214 version 15.0.0, "NR; Physical layer procedures for data." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216

[8] 3GPP TS 38.101-1 version 15.0.0, "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3201

[9] 3GPP TS 38.101-2 version 15.0.0, "NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3284

[10] 3GPP TS 38.300 version 15.0.0, "NR; NR and NG-RAN overall description." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3191

[11] 3GPP TR 38.901 version 15.0.0, "Study on channel model for frequencies from 0.5 to 100 GHz." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3173

[12] 3GPP TR 22.891 version 14.2.0, "Feasibility study on new services and markets technology enablers; Stage 1." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2897

[13] 3GPP TS 36.211 version 15.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2425

[14] 3GPP TS 36.213 version 15.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427

[15] 3GPP TS 36.101 version 15.1.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception." Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2411

[16] D. Schmalstieg and T. Höllerer, *Augmented Reality: Principles and Practice.* Reading, MA, USA: Addison-Wesley, 2016.

[17] G. Burdea, and P. Coiffet, *Virtual Reality Technology, 2nd ed.* New York, NY, USA: Wiley, 2003.

[18] L. Kong, M. K. Khan, F. Wu, G. Chen, and P. Zeng, "Millimeter-wave wireless communications for IoT-cloud supported autonomous vehicles: Overview, design, and challenges," *IEEE Commun. Mag.,* vol. 55, no. 1, pp. 62-68, Jan. 2017.

[19] T. S. Rappaport, *Wireless Communications: Principles and Practice.* Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[20] S. Sesia, I. Toufik, and M. Baker, *LTE-The UMTS Long Term Evolution: From Theory to Practice.* New York, NY, USA: Wiley, 2009.

[21] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE Commun. Surveys Tutorials,* vol. 15, no. 2, pp. 678-700, 2013.

[22] O. Grondalen, A. Zanella, K. Mahmood, M. Carpin, J. Rasool, and O. Osterbo, "Scheduling policies in time and frequency domains for LTE downlink channel: a performance comparison," *IEEE Trans. Veh. Technol.,* vol. 66, no. 4, pp. 3345-3360, Apr. 2017.

[23] A. Stolyar, "On the asymptotic optimality of the gradient scheduling algorithm for multi-user throughput allocation," *Operations Research,* vol. 53, pp. 12-25, 2005.

[24] D. Tse, "Multiuser diversity in wireless networks: smart scheduling, dumb antennas and epidemic communication," in *IMA Workshop on Wireless Networks,* 2001. Available: https://web.stanford.edu/~dntse/papers/ima810.pdf

[25] R. Kwan, C. Leung, and J. Zhang, "Proportional fair multiuser scheduling in LTE," *IEEE Signal Process. Lett.,* vol. 16, pp. 461-464, June 2009.

[26] S. B. Lee, S. Choudhury, A. Khoshnevis, S. Xu, and S. Lu, "Downlink MIMO with frequency-domain packet scheduling for 3GPP LTE," in *Proc. IEEE INFOCOM,* pp. 1269-1277, Apr. 2009, Rio de Janeiro, Brazil.

[27] H. Zhang, N. Prasad, and S. Rangarajan, "MIMO downlink scheduling in LTE systems," in *Proc. IEEE INFOCOM,* pp. 2936-2940, Mar. 2012, Orlando, FL, USA.

[28] H. S. Liao, P. Y. Chen, and W. T. Chen, "An efficient downlink radio resource allocation with carrier aggregation in LTE-Advanced networks," *IEEE Trans. Mobile Computing,* vol. 13, no. 10, pp. 2229-2239, Oct. 2014.

[29] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *Proc. ACM SIGCOMM,* pp. 195-206, Aug. 2010, New Delhi, India.

[30] F. Fusco, M. Vlachos, X. Dimitropoulos, and L. Deri, "Indexing million of packets per second using GPUs," in *Proc. ACM SIGCOMM-IMC,* pp. 327-332, Oct. 2013, Barcelona, Spain.

[31] M. Varvello, R. Laufer, F. Zhang, and T. V. Lakshman, "Multilayer packet classification with graphics processing units," *IEEE Trans. Networking,* vol. 24, no. 5, pp. 2728-2741, Oct. 2016.

[32] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A. M. Vidal, "Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector," *IEEE Trans. Veh. Technol.,* vol. 61, no. 8, pp. 3796-3800, Oct. 2012.

[33] Y. Zhao and F. Lau, "Implementation of decoders for LDPC block codes and LDPC convolutional codes based on GPUs," *IEEE Trans. Parallel Distrib. Syst.,* vol. 25, no. 3, pp. 663-672, Mar. 2014.

[34] A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Implementation of a fully-parallel turbo decoder on a general-purpose graphics processing unit," *IEEE Access,* vol. 4, pp. 5624-5639, Jun. 2016.

[35] H. D. Sherali and W. P. Adams, *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems,* Chapter 8. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[36] Y. T. Hou, Y. Shi, and H. D. Sherali, *Applied Optimization Methods for Wireless Networks,* Chapter 6. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[37] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization.* New York, NY, USA: John Wiley & Sons, 1999.

[38] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Belmont, MA, USA: Athena Scientific, 1995.

[39] E. G. Talbi, *Metaheuristics: From Design to Implementation.* Hoboken, NJ, USA: Wiley, 2009.

[40] M. R. Zargham, *Computer Architecture: Single and Parallel Systems.* Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[41] IBM ILOG CPLEX Optimizer, software available at http://www01.ibm.com/software/integration/optimization/cplex-optimizer
[42] Nvidia, "Data sheet: Quadro P6000." Available: https://images.nvidia.com/content/pdf/quadro/data-sheets/192152-NV-DS-Quadro-P6000-US-12Sept-NV-FNL-WEB.pdf
[43] Nvidia, "CUDA C programming guide v9.1." Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[44] Nvidia, "Optimizing parallel reduction in CUDA." Available: https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf
[45] W. Yang, G. Durisi, and E. Riegler, "On the capacity of large-MIMO block-fading channels," *IEEE J. Sel. Areas Commun.,* vol. 31, no. 2, pp. 117-132, Feb. 2013.
[46] Nvidia, "Nvidia Tesla P100 – The most advanced data center accelerator ever built." Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf
[47] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Trans. Parallel Distrib. Syst.,* vol. 28, no. 3, pp. 905-918, Mar. 2017.
[48] M. Daga, M. Nutter, and M. Meswani, "Efficient breadth-first search on a heterogeneous processor," in *Proc. IEEE Int. Conf. Big Data,* pp. 373-382, Oct. 2014, Washington DC, USA.
[49] Intel, "The compute architecture of Intel Processor Graphics Gen7.5." Available: https://software.intel.com/sites/default/files/managed/4f/e0/Compute_Architecture_of_Intel_Processor_Graphics_Gen7dot5_Aug4_2014.pdf