GPU: A New Enabling Platform for Real-Time Optimization in Wireless Networks

Yan Huang, Shaoran Li, Yongce Chen, Y. Thomas Hou, Wenjing Lou, James Delfeld, and Vikrama Ditya

Abstract

Optimization methods are a common tool to maximize the performance of wireless networks and systems. When addressing complex optimization problems in wireless networks, a key technical challenge is to find an optimal or near-optimal solution in real-time, especially when such a timing constraint is extremely short. Due to this challenge, there is usually a serious disparity between what a system can achieve optimally (if an optimal solution were found in real time) and what is actually achieved in the field (due to the use of fast heuristics). In this article, we present a novel approach that exploits problem decomposition techniques and the massive parallel processing capability of GPU platforms to address this challenge. Under the new approach, an original complex optimization problem is first decomposed into a large number of small and mutually independent sub-problems. Then the resulting sub-problems are fitted into massively parallel GPU cores and solved simultaneously. The optimal (or near-optimal) solution is chosen among the solutions from all the parallel sub-problems solved by GPU. We use the classic proportional-fair (PF) scheduling problem in 5G cellular networks as a case study to illustrate this approach. Finally, we briefly review recent advances in applying this approach to addressing a wide array of real-time optimization problems in wireless networks.

Introduction

There is a growing demand on wireless networks to offer higher throughput, lower latency, and more connectivity. Despite advances of radio access technologies at the physical (PHY) layer, the performance of wireless networks remains constrained by limited frequency spectrum, unreliable channel conditions, complex interference, and battery. To get the most of current and future generations of wireless networks, it is important to optimally design and utilize network infrastructure and radio resources across multiple domains (frequency, time, spatial, and power).

In this endeavor, optimization methods are often used as a primary tool by researchers and operators. Typically, a resource optimization problem involving design variables at multiple layers such as routing, scheduling, power control, and beamforming, among others, is first formulated as a mathematical program, in the form of an objective function[1] and a set of constraints [1]. Due to the presence of integer decision variables and nonlinear functions in many of these problems, they are typically NP-hard and cannot be solved optimally in real time. So in most cases, one needs to develop a near-optimal solution (provably or heuristic). As expected, the better the performance of the objective value, the higher the complexity of the underlying solution.

In practice, the allowed time scale to compute an optimal (or near-optimal) solution to an optimization problem depends on the nature of the problem and can be very limited. For example, in 4G LTE, the duration of a transmission sub-frame is only 1 ms [2]. Since radio resources in LTE must be allocated for every sub-frame, the scheduling solution for each sub-frame must be determined under 1 ms.[2] But such timing requirement has posed a serious challenge, which we call the real-time challenge of finding an optimal (or near-optimal) solution. Even more so, as wireless communications evolve from 4G LTE to 5G, 6G and beyond, the timing requirement from either the applications or the radio interface appears to become drastically more stringent. For example, under the OFDM numerologies defined in 5G standards [4], the minimum duration for data transmission is merely 125 μs (for time slot based transmission), which is almost one order of magnitude shorter than 4G LTE (1 ms).

Such real-time requirements, although critical for wireless networks in the field, have never been a focus in traditional optimization methods. In fact, the mainstream of research works on wireless network optimizations relies on the *asymptotic complexity* analysis (either in time or space) for designing solution algorithms. But asymptotic complexity analysis of an algorithm is only concerned with the growth of its computational complexity as the input size n increases (i.e., expressed in the *big-O* notation [5]). It is far from adequate to address real-time requirements that are measured in *wall-clock time*, (i.e., in terms of μs or ms). For example, with an input size n = 100 (e.g., the number of users in a macro cell), an algorithm (A1) with complexity $0.1n^3$ may be 10× faster than another algorithm (A2) with complexity $100n^2$. Although with a higher asymptotic complexity $O(n^3)$, A1 is a faster algorithm than A2 as measured in wall-clock time for a given input size n = 100 and may meet real-time requirement in the field. That is, the actual computation time performance of an algorithm for a given problem size could be more important than merely lower asymptotic complexity of the algorithm.

[1] Multiple objectives are possible under multi-criteria optimization.

[2] The computation of scheduling solutions, the delivery of scheduling grants to users, and the actual data transmissions may not happen within the same sub-frame. For example, the time gap between scheduling grant and uplink data transmission is at least 4 ms [3].

Yan Huang, Shaoran Li, Yongce Chen, Y. Thomas Hou (corresponding author), and Wenjing Lou are with Virginia Polytechnic Institute and State University; James Delfeld and Vikrama Ditya are with NVIDIA Corporation.

Digital Object Identifier: 10.1109/MNET.011.2000016

0890-8044/20/$25.00 © 2020 IEEE

Only until very recently did the community get a glimpse of what a possible approach to achieving this might be, that is, the use of GPU in solving optimization problems in real time.

In this article, we use wall-clock time as the ultimate benchmark for timing complexity when developing a solution to an optimization problem. We present a novel GPU-based optimization approach to addressing this real-time challenge. Our proposed approach decomposes a complex optimization problem into a large number of independent and structurally-identical small problems and exploits GPU's massive low-cost parallel processing capability to find an optimal (or near-optimal) solution in real time. There are two basic assumptions for our GPU-based optimization approach. First, the optimization problem should include integer variables, i.e, the problem should belong to mixed-integer linear program (MILP) or mixed-integer nonlinear program (MINLP) (or integer linear program (ILP)/integer nonlinear program (INLP)). This is necessary so that the original problem can be decomposed into a large number of sub-problems. Such decomposition is usually done by fixing values for integer variables. Second, we assume the optimization problem is solved by a central controller in the network that has all the information needed for the optimization problem.

In the rest of this article, we first review existing solution approaches (based on wall-clock performance) and discuss their limitations. Then we present the new GPU-based approach to solving complex optimization problems in real time. Finally, we review recent advances in this direction.

Existing Approaches: A State of the Art

In this section, we classify existing optimization methods in wireless networks based on their wall-clock time performance. In this regard, we classify three approaches:

- Offline optimization methods that can find optimal (or near-optimal) solutions but usually incur excessive computation time.
- Fast heuristic algorithms which have the goal of pursuing good solutions (not necessarily optimal or near-optimal) and only require polynomial-time complexity (but still may not meet real-time requirement).
- Industry-grade real-time solutions that can meet real-time requirement but usually with performance far from optimum.

Offline Optimization Methods

Consider a typical mathematical formulation of an optimization problem from wireless networks that consists of an objective function and a set of constraints depicting its feasible region. When such a problem involves integer variables (commonly encountered in scheduling and routing) and nonlinear functions (e.g., logarithmic function to calculate channel capacity), the process of finding an optimal (or near-optimal) solution can be computationally intensive, particularly when the search space (i.e., feasible region) is large.

One effective approach to addressing such combinatorial optimization problems is to relax the original problem into a linear programming (LP) or nonlinear programming (NLP) problem. The relaxed problems can be solved by the classic simplex algorithm (for LP) or convex optimization methods (for convex NLP or convex hull relaxations). When the solution to the relaxed problem is infeasible to the original problem, one can use a local search to find a feasible solution. Since simplex and convex methods involve a large number of iterations, such an approach usually cannot meet real-time requirement.

For MILPs, cutting plane (CP) is a widely used method to find optimal solutions. CP iteratively solves LP relaxations of an MILP and adds linear "cut" constraints without excluding integer feasible points until an optimal solution is found. Another class of solution methods is branch-and-bound (BB), which can be used for solving both MILPs and MINLPs. BB is a tree-search algorithm that keeps track of upper and lower bounds on the optimal solution and uses these bounds to prune branches of the tree that do not contain an optimal solution. By relaxing the optimality criterion, BB can also be used to find solutions within $1 \pm \varepsilon$ of the optimum for any small positive value ε. Furthermore, a very popular solution approach is branch-and-cut (BC), which combines BB with CP. Basically, BC uses CP in addition to LP relaxations in the bounding steps, and is able to obtain tighter bounds compared to BB. Although BB, CP or BC can eventually find optimal (or near-optimal) solutions, their computation time is long and increases exponentially as problem size increases. For example, a resource scheduling problem in a 4G LTE or 5G NR cell may involve tens of thousands of integer variables and constraints. A solution based on BC may take tens of seconds or even hours to find a near-optimal solution with $\varepsilon = 5$ percent [6]. Such performance in wall-clock time limits these methods to be only useful as offline benchmark solutions.

Fast Heuristics Methods

Since an exact optimization method such as BC is too slow to be useful in the field, researchers are pressed to design fast heuristic algorithms. A heuristic represents a problem-solving scheme that is not guaranteed to be optimal for a given problem objective, but is believed or empirically validated to be able to achieve good performance. Computational time complexity of a heuristic is expected to be in polynomial-time and a common goal is to make its order (measured in $O(\cdot)$) as low as possible.

There are many different types of heuristics for solving optimization problems in wireless networks, such as local search, greedy algorithm, and iterative partial fixing, to name a few. Greedy algorithm is a heuristic scheme that always makes locally optimal choices throughout the iterations. For instance, a greedy scheduling algorithm will assign each resource unit to a user that has the highest data rate on it, regardless of the global optimization objective. Local search is a heuristic method that only explores candidate solutions within a local sub-space instead of the entire problem feasible region. With iterative partial fixing, different subsets of variables are fixed and optimized iteratively (in each iteration optimizing one subset of variables while all other variables are fixed) until there is no significant change of variable values. In general, heuristics are designed

to solve specific problems and may have very diverse features and properties.

Although a heuristic algorithm may be of low polynomial-time complexity, there is no guarantee that it can meet real-time requirement. For example, as shown in [10], many state-of-the-art heuristics for proportional-fair (PF) scheduling are of polynomial-time complexities. But none of them is able to meet the stringent real-time requirement in 5G NR. As a result, the common wisdom held in the research community that a polynomial-time algorithm is sufficient is simply false as there is no guarantee that such an algorithm can meet real-time requirement. In fact, many well-known heuristic solutions in the literature can hardly satisfy real-time requirement (under any known implementation).

Industrial-Grade Real-Time Solutions

Given the limitations of offline optimization methods and fast heuristic algorithms, wireless engineers in the field have to resort to extremely simple solutions that can meet real-time requirement. But unfortunately, many of these solutions offer performance that is far from optimum (in terms of objective value).

An example of industry-grade real-time algorithms is the round-robin (RR) algorithm for resource scheduling in cellular networks [7]. Here, the real-time requirement for allocating 100s of resource blocks (RBs) on a channel to 100s of users is 1 ms in 4G LTE and sub-1 ms in 5G NR. To reduce complexity, RR bundles RBs within each transmission time interval (TTI) into multiple RB groups (RBGs) and assigns these RBGs to users in a cyclic manner regardless of the users' channel conditions. RR is attractive for its simplicity and low computation cost and can be easily implemented to meet real-time requirement.

In summary, algorithms used in the field are designed with real-time consideration as their first and foremost criterion. However, such an algorithm may experience severely compromised performance in terms of optimization objective. As of today, optimal (or near-optimal) solutions to most of the complex optimization problems in wireless networks can hardly be obtained in real time. As a result, there is a serious disparity between what a wireless network may achieve optimally (through the use of offline optimization methods) and what this same network is currently offering in the real world.

GPU-Based Solution Approach: A New Direction

Given the limitations of existing approaches, one may wonder whether or not it is possible to achieve the best of both worlds, that is, solving hard problems with optimal/near-optimal performance (as offline optimization methods) while still meeting real-time requirements (as industry-grade solutions). This objective has remained a holy grail for academia and industry for decades. Only until very recently did the community get a glimpse of what a possible approach to achieving this might be, that is, the use of GPU in solving optimization problems in real time. In this section, we offer an overview of this new approach.

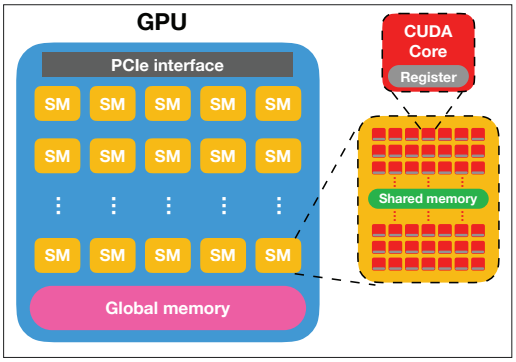

FIGURE 1. An illustration of GPU architecture. This diagram omits many components on a GPU such as L1/L2 caches, memory controllers, warp schedulers and so on.

Basic Idea

The core of this new approach is to decompose an original optimization problem into a large number of small and independent sub-problems that have the same mathematical structure, and then employ a GPU's massive parallel processing cores to solve the sub-problems in parallel. A key in the problem decomposition step is to ensure independence among all sub-problems. This will ensure that they can be solved simultaneously without any inter-process communications, which are costly in time. In addition, each small sub-problem should have the same mathematical structure and problem size so that they can be solved using the same set of instructions that are implemented across all GPU cores in a single-instruction multiple-data (SIMD) manner.[3] Under SIMD, the sub-problems are solved at about the same time on GPU. Then the final output solution (to the original problem) can be found by comparing the objective values achieved among the sub-problem solutions.

The motivation to this approach is twofold. First, the modern GPU platform offers massive parallel processing capability that was once unimaginable and has never been exploited to solve real-time optimization problems. Second, for a complex combinatorial optimization problem, there may exist some approach to decomposing the original problem into a large number of tiny sub-problems where the solution to each sub-problem is simple enough that it can just be handled easily by a GPU core in real-time. Therefore, the success of this approach hinges upon a deep knowledge of the capability and limitation of a GPU architecture and the mathematical techniques to decompose a complex combinatorial optimization problem into tiny sub-problems that can match nicely to a given GPU architecture.

In the rest of this section, we review the state-of-the-art of GPU architecture. We also compare GPU with other computing platforms such as CPU-based parallel computing, FPGA, and ASIC based solutions. In the next section, we use a case study to show how a complex combinatorial optimization problem can be decomposed into numerous sub-problems and how a GPU can be used to find a near-optimal solution in real time.

[3] SIMD means that the same operation instruction is executed by multiple processing elements at different data points [8]. In contrast, multiple-instruction multiple-data (MIMD) machines have processing elements perform different instructions at different data points.

Platform	GPU	CPU	FGPA	ASIC
Key features	> 1000 cores, general-purpose, optimized for SIMD, off-the-shelf, high throughput	~ 10s cores, general-purpose, optimized for MIMD, off-the-shelf, high clock speed	Massive re-programmable logic blocks, general-purpose, optimized for fixed-point operations, off-the-shelf, suitable for low-volume hardware validation	Massive permanent logic gates, manufactured for customized design, application-specific, optimized for fixed-point operations, suitable for high-volume production
SIMD parallelism	High	Low	Medium	Design-dependent
Programming flexibility	High	High	Low	Low
Reprogrammable	Yes	Yes	Yes	No
Floating-point efficiency	High	High	Low	Low
Off-the-shelf	Yes	Yes	Yes	No
Power efficiency	Medium	Medium	High	High

TABLE 1. Comparison of GPU to other parallel computing platforms.

GPU Architecture

The architecture of a GPU is best illustrated with an example. Figure 1 illustrates a GPU architecture from NVIDIA.[4] A GPU is generally composed of multiple streaming multi-processors (SMs), with each SM consisting of a large number of parallel processing cores (a.k.a., CUDA cores). The massive processing cores on a GPU offer enormous capability for large-scale parallel computing. GPU has a hierarchical memory structure as shown in Fig. 1. Specifically, a GPU has its own on-board global memory, which can be accessed by all SMs within the GPU. In addition, each SM has local shared memory that can be used for data communication among processing cores on the SM. Further, there are registers that are only accessible to each processing core. An important feature of GPU memory hierarchy is that each type of memory has very different access speed. Local registers in a processing core are the fastest. Shared memories are also very fast for data sharing within the SM. But access to global memory is much slower than registers and shared memories. The proper usage of a GPU's hierarchical memories is a key to algorithm design and implementation for meeting real-time requirement.

On the software side, the CUDA programming tool [9] is available to program NVIDIA GPUs for parallel computing. It offers highly flexible programming interfaces to directly control the massive CUDA cores. Under CUDA, a kernel (i.e., a function to be executed on GPU) is composed of multiple thread blocks (TBs) and each TB further consists of a large number of concurrent threads (up to 1024). During execution, each TB is run by a single SM, while an SM may be assigned to multiple TBs. All threads in a TB will be scheduled to utilize the CUDA cores within an SM for executing their programmed instructions.

GPU vs. Other Computing Platforms

Given that there are other parallel computing platforms available, a natural question to ask is why a GPU-based approach is more desirable here. To address this question, we have prepared a table for the comparison between GPU and other platforms including CPU, FPGA and ASIC, as shown in Table 1.

As described earlier, the key idea behind our real-time solution design is to first decompose an optimization problem into a large number of structurally-identical sub-problems and then solve the sub-problems through a SIMD computing architecture. Thus an important criterion for selecting a computing platform is how much SIMD parallelism a platform can offer. As shown in Table 1, GPU is much more capable than CPU in terms of large-scale SIMD thanks to its massive parallel CUDA cores, making it a better choice for implementing our solution design.

Although FPGA can also offer a high level of parallelism, its total number of parallel processing units is no match to that from a GPU. In addition, unlike GPU, FPGA is very inefficient for floating-point processing, which is necessary for addressing many optimization problems in wireless networks.

Compared with ASIC, the major advantages of GPU are that it is general-purpose, of low-cost, and available off-the-shelf. In contrast, the development of ASIC for a specific algorithm is expensive (both time and labor). Further, it is difficult to make changes once an ASIC is made, which is undesirable as algorithms are bound to evolve over time.

A Case Study: GPU-Based Real-Time Scheduler for 5G

As a case study to demonstrate how one can decompose a complex combinatorial optimization problem and use GPU cores to solve the resulting sub-problems in parallel, consider the classic PF resource scheduling in 5G NR [10].

The Problem

In 5G NR, the minimum time resolution for resource scheduling is 125 μs under OFDM Numerology 3 [4, 10]. This means that the real-time requirement for finding a scheduling solution should be no more than 125 μs. This is almost an order of magnitude shorter than 4G LTE, which has a 1 ms time interval for making a scheduling decision.

The PF scheduling problem involves the allocation of 100s of resource blocks (RBs) to up to 100 users in a cell, and the selection of one out of 29 modulation and coding schemes (MCSs) for each user. The optimization objective is to maximize the PF among all users, defined as $\Sigma_i \log \tilde{R}_i$, where $\tilde{R}_i$ is the long-term average data rate of user i.

[4] We choose to use NVIDIA GPU in our illustration due to the popularity of NVIDIA's CUDA programming platform [9] and the large user base of NVIDIA GPUs in the industry.

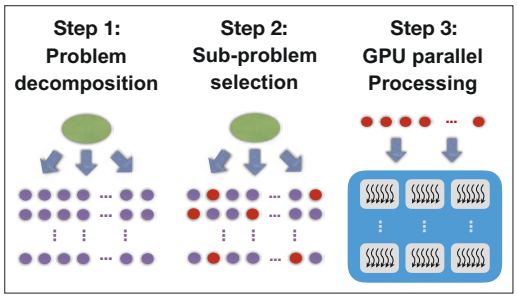

FIGURE 2. A diagram for the GPU-based PF scheduler design. The design includes three steps: 1) decomposition of the original problem into a large number of sub-problems, 2) selection of sub-problems to fit the available GPU cores, and 3) parallel processing of sub-problems on GPU cores.

This PF scheduling problem has been proved to be NP-hard [11], which is attributed to the following complexities. First, each user's channel conditions vary across different RBs due to frequency-selective channel fading. Second, a user must employ the same MCS for all of its allocated RBs to generate the transport block. In 5G NR, 29 MCS levels [12] are defined, where a higher MCS level corresponds to a higher spectral efficiency ((b/s)/Hz) but requires a better channel quality. In particular, an RB can only offer a non-zero achievable data rate when the underlying user channel quality is no worse than the selected MCS's requirement. Otherwise, this RB will not be able to offer any usable throughput if the channel quality cannot match up to the selected MCS. This MCS selection problem is clearly a trade-off between higher throughput per RB and the number of RBs that can indeed contribute to such throughput.

Solution Design

The proposed solution is shown in Fig. 2. The first step is to decompose the original PF optimization problem into a large number of sub-problems. There may exist different ways to achieve this but it is sufficient to illustrate one that works. Specifically, we can decompose the original problem by fixing the choice of MCS for each user. For instance, when there are 100 users in the cell, we have a total of 29^{100} sub-problems, where each sub-problem has a fixed (given) MCS selection for each user. Then for each sub-problem we only need to find the optimal RB allocation among all users, which is a much smaller problem.

Based on this decomposition, the optimal solution to the original PF problem can be determined by solving all sub-problems and then identifying one solution that has the maximum PF objective value. But clearly, 29^{100} is too large to fit into the available CUDA cores on a GPU. So our next step is to determine a smaller set of sub-problems from which it is more likely to find an optimal or near-optimal solution. This is called *intensification* in optimization [13]. Once we identify such a subspace, we can fit the sub-problems in this space into GPU cores.

As a first step to reduce the number of sub-problems, we can consider the most promising choice of MCS levels, instead of enumerating all of them. One plausible choice is to limit the range of MCS to the highest several levels. For example, we could consider only the three highest MCS levels that can offer nonzero data rates on some RBs for each user (e.g., MCS 7, 8 and 9 for user 1, MCS 20, 21, and 22 for user 2, and so forth). Consequently, the number of sub-problems will be reduced to 3^{100}, which is much smaller than 29^{100}.

However, 3^{100} is still too large compared to the number of CUDA cores on a GPU. Next, we perform a random sampling over the search subspace to select a small subset of sub-problems. The size of the subset depends on how many CUDA cores are available. This sampling can be done by randomly choosing the MCS for each user from its highest three feasible MCS levels. After this sampling step, all selected sub-problems will be solved (finding the optimal RB allocation) in parallel on GPU. Then the final output solution is determined by finding the sub-problem solution that achieves the highest PF objective value.

The success of the two steps (intensification and sampling) in reducing the problem space hinges upon the characteristics of the original problem as well as its physical properties. These steps are necessary to fit a large-scale combinatorial optimization problem into GPU cores. Each problem may call for a unique approach to accomplishing this goal. Most importantly, one must justify that they can still get an optimal or near-optimal solution after these steps. For this particular PF scheduling problem, there is some rigorous science (based on probability theory and empirical data) behind the intensification and sampling steps that guarantees one can find a near-optimal solution. For readers who are interested in the details, we refer them to [10].

Performance

We implemented this GPU-based PF scheduler on an NVIDIA Quadro P6000 GPU, which has 30 SMs and 128 CUDA cores per SM (3840 totally). Experimental results with 100 RBs on a 20 MHz channel, 29 MCS levels (following 5G NR [12]), and different numbers of users per cell (25, 50, 75, and 100) are given in Table 2. For finding optimal solutions, we used the same desktop computer that is equipped with the GPU. Instead of using the GPU, we used the default CPU in the desktop, which is an Intel Xeon E5-2687W v4. For the solver, we employed the IBM CPLEX Optimizer (version 12.7.1). Results of normalized objective values were obtained by normalizing our experimental results with respect to the optimal objectives. We see that in all cases the scheduler can find near-optimal solutions and the average computation time is well within the 125 μs real-time requirement.

Recent Developments

The proposed GPU-based solution approach has the potential to solve many complex real-time optimization problems arising from wireless networks. In this section, we describe some recent advances in applying this approach to problems in a wide range of areas, including hybrid beamforming (HB) for millimeter wave (mmWave) massive MIMO, LTE/WiFi coexistence in unlicensed spectrum, and joint scheduling and power control.

Number of users	25	50	75	100
Average time to compute optimum (μs)	3.20×10^6	1.06×10^7	1.82×10^7	3.02×10^7
Average comput. time by GPU (μs)	96.16	94.93	112.60	116.21
Normalized objective by GPU (%)	94.65	98.66	98.76	99.53

TABLE 2. Experimental results of the GPU-based PF scheduler.

HB for mmWave MIMO

HB uses a fewer number of RF chains than the total number of antennas and is considered a practical solution to implement massive MIMO beamforming in mmWave frequency bands. HB consists of both analog and digital beamforming. A critical challenge in HB is to compute digital beamforming weights in real time, where the timing requirement can be as short as 1 ms. However, almost all research efforts on HB remain theoretical and none of the state-of-the-art efforts is able to meet this real-time requirement. Recently, a GPU-based solution was proposed in [14] to compute digital beamforming weights in real time. Specifically, the design reduces the complexity of singular-value-decomposition (SVD) by exploiting the sparsity of channel matrices in the mmWave band. Further, multi-user MIMO beamforming across all RBs on the channel is decomposed into a large number of independent single-user MIMO beamforming sub-problems, which have the same structure and much smaller problem size. Then these single-user bearmforming problems are computed in parallel using GPU cores. Implementation of this approach demonstrated that it can meet the 1 ms real-time requirement, which is a major milestone to demonstrate that the HB technique is feasible for deployment in the field.

LTE/WiFi Coexistence in Unlicensed Spectrum

Carrier sensing adaptive transmission (CSAT) is a promising technique to achieve LTE/WiFi coexistence in the unlicensed band. Under CSAT, an LTE scheduler needs to decide the channel selection for uplink (UL) and downlink (DL) traffic, the division of air time for LTE and WiFi per channel, and resource allocation within LTE's "on" periods. The scheduler must minimize the adverse impact of LTE on WiFi while meeting LTE users' UL and DL rate requirements. Here the real-time requirement is that LTE scheduling time interval is only 1 ms, meaning that an optimal or near-optimal solution must be found within this time interval. None of the existing LTE schedulers designed for licensed spectrum can be easily extended to address this new coexistence problem. In a recent work [6], a GPU-based solution was proposed to address this optimal scheduling problem in real time. The original optimization problem is decomposed into a large number of feasibility-check sub-problems by fixing channel assignment variables and objective values. By evaluating all feasibility checks using GPU cores in parallel, a near-optimal scheduling solution can be determined within the 1 ms timing requirement.

Joint Scheduling and Power Control

In underlay coexistence, secondary users are allowed to be active simultaneously as the primary users as long as their interference to the primary users is under control. This can be achieved through judicious design of joint scheduling and power control of secondary users. In [15], this problem was studied under a scheduling time constraint of 250 μs, which can meet Numerology 0, 1, and 2 in 5G NR. The authors employed chance-constrained programming to address channel uncertainty from the primary users to the secondary users. The proposed solution scheme consists of three steps. The first step is to decompose the original problem into a large number of sub-problems by enumerating all possible scheduling decisions. Then a subset of sub-problems is selected (based on secondary users' channel conditions) to fit the available GPU cores. The last step is to solve selected sub-problems on the GPU based on closed-form starting point initialization and scaling-based local search. The final output solution is determined by finding the best sub-problem solution. Experimental results confirm that the proposed solution achieves 90 percent-optimality on average within 150 μs.

Conclusion and Future Work

A holy grail in wireless networks (as well as in many other systems) is to solve complex optimization problems in real time. In this article, we proposed a novel GPU-based solution design that was shown to be highly effective in addressing this challenge, particularly for complex combinatorial optimization problems involving integer decision variables. The proposed approach hinges upon effective problem decomposition techniques to transform an original combinatorial optimization problem into a large number of tiny sub-problems, each of which is easy to solve and independent from each other. Then by employing the massive SIMD parallel processing capability of the GPU platform and a high-level programming language (e.g., CUDA), we showed that an optimal or near-optimal solution can be obtained in real time. We used a classic PF scheduling problem in 5G NR as a case study to illustrate the efficacy of this approach. Further, we offered a brief review of recent advances in applying such an approach to addressing a number of hard optimization problems in wireless networks. Although this novel approach is still in its infancy, we believe it has the potential to be applied in the real world when there is a need to solve complex optimization problems in real time. As future work, we will explore more advanced techniques to further improve this GPU-based optimization approach. Potential improvements include using machine learning methods to help identify promising search sub-space for sub-problem sampling, employing hybrid CPU/GPU architectures to further enhance the parallelism of the algorithm, and investigating the provable performance gap of this approach to the global optimum.

Acknowledgments

This research was supported in part by NSF grant 1800650. The authors from Virginia Tech thank the NVIDIA AI Lab (NVAIL) in Santa Clara, CA for its unrestricted gift and equipment donation. All opinions expressed in this article are the authors' and do not necessarily reflect the views and opinions of NSF or NVIDIA.

References

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

[2] 3GPP TS 36.211 version 16.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2425.

[3] 3GPP TS 36.213 version 16.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures," available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427.

[4] 3GPP TS 38.211 version 16.0.0, "NR; Physical Channels and Modulation," Available: https://portal.3gpp.org/desktopmodules/ Specifications/SpecificationDetails.aspx?specificationId=3213.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Chapter 1, New York, NY, USA: Freeman, 1990.

[6] Y. Huang *et al.*, "Achieving Fair LTE/Wi-Fi Coexistence with Real-Time Scheduling," *IEEE Trans. Cognitive Commun. and Networking*, vol. 6, no. 1, Mar. 2020, pp. 366–80.

[7] F. Capozzi *et al.*, "Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 2, 2013, pp. 678–700.

[8] K. Hwang and N. Jotwani, *Advanced Computer Architecture*, 3rd ed, Chapter 1, New York, NY, USA: McGraw-Hill Education, 2016.

[9] NVIDIA, "CUDA C++ Programming Guide," available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[10] Y. Huang *et al.*, "GPF: A GPU-Based Design to Achieve ~100 μs Scheduling for 5G NR," *Proc. ACM MobiCom*, Oct. 29-Nov. 2, 2018, New Delhi, India, pp. 207–22.

[11] S. B. Lee *et al.*, "Downlink MIMO with Frequency-Domain Packet Scheduling for 3GPP LTE," *Proc. IEEE INFOCOM*, Apr. 2009, Rio de Janeiro, Brazil, pp. 1269–77.

[12] 3GPP TS 38.214 version 16.0.0, "NR; Physical Layer Procedures for Data," available: https://portal.3gpp.org/desktopmodules/ Specifications/SpecificationDetails.aspx-?specificationId=3216.

[13] E. G. Talbi, *Metaheuristics: From Design to Implementation*, Chapter 2 & 3, Hoboken, NJ, USA: Wiley, 2009.

[14] Y. Chen *et al.*, "Turbo-HB: A Novel Design and Implementation to Achieve Ultra-Fast Hybrid Beamforming" *Proc. IEEE INFOCOM*, July 6–9, 2020, Toronto, Canada.

[15] S. Li *et al.*, "A Real-Time Solution for Underlay Coexistence with Channel Uncertainty" *Proc. IEEE GLOBECOM*, Dec. 2019, Waikoloa, HI, USA.

Biographies

Yan Huang [S'15] received the B.S. and the M.S. degrees in electrical engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2012 and 2015, respectively. He is currently studying toward the Ph.D. degree at Virginia Tech, Blacksburg, VA, USA. His research interests are GPU-based real-time optimizations for wireless networks and machine learning for communications.

Shaoran Li [S'17] received the B.S. degree from Southeast University, Nanjing, China, in 2014 and the M.S. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2017. He is a currently working toward the Ph.D. degree at Virginia Tech, Blacksburg, VA, USA. His research interests include algorithm design and implementation for wireless networks.

Yongce Chen [S'16] received the B.S. and M.S. degrees in electrical engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2013 and 2016, respectively. He is currently working toward the Ph.D. degree at Virginia Tech, Blacksburg, VA, USA. His current research interests include wireless network optimization, MIMO techniques and real-time implementation of wireless systems.

Y. Thomas Hou [F'14] is the Bradley Distinguished Professor of Electrical and Computer Engineering at Virginia Tech, Blacksburg, VA, USA, which he joined in 2002. He received his Ph.D. degree from NYU Tandon School of Engineering in 1998. From 1997 to 2002, he was a member of research staff at Fujitsu Laboratories of America, Sunnyvale, CA, USA. His current research focuses on developing innovative solutions to complex science and engineering problems arising from wireless and mobile networks. He is also interested in wireless security. He has over 300 papers published in IEEE/ACM journals and conferences. His papers were recognized by eight best paper awards from IEEE and ACM. He holds five U.S. patents. He has authored/co-authored two graduate textbooks: *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, 2014) and *Cognitive Radio Communications and Networks: Principles and Practices* (Academic Press/Elsevier, 2009). He was named an IEEE Fellow for contributions to modeling and optimization of wireless networks. He was/is on the editorial boards of a number of IEEE and ACM transactions and journals. He served as Steering Committee Chair of IEEE INFOCOM and was a member of the IEEE Communications Society Board of Governors. He was also a Distinguished Lecturer of the IEEE Communications Society.

Wenjing Lou [F'15] is the W. C. English Endowed Professor of Computer Science at Virginia Tech and a Fellow of the IEEE. Her research interests cover many topics in the cybersecurity field, with her current research interest focusing on blockchain, privacy protection in machine learning systems, and security and privacy problems in Internet of Things (IoT) systems. She is a highly cited researcher by the Web of Science Group. She received the Virginia Tech Alumni Award for Research Excellence in 2018, which is the highest university level faculty research award. She received the INFOCOM Test-of-Time paper award in 2020. She is a TPC Chair for IEEE INFOCOM 2019 and ACM WiSec 2020. She is the Steering Committee Chair of the IEEE CNS conference, steering committee member of IEEE INFOCOM and *IEEE Transactions on Mobile Computing*. She served as a program director at the U.S. National Science Foundation (NSF) from 2014 to 2017.

James Delfeld is a software engineer at NVIDIA working on GPU-enabled virtual RAN. He received a Ph.D. degree in mathematics from the University of Texas at Austin, USA in 2015.

Vikrama Ditya heads the Radio Access Networks compute and standards group at NVIDIA. He holds an M.S. in computer science from India. His areas of interests are distributed computing, scheduling algorithms and telecom/network system architecture.