Efficient FFT Computation in IFDMA Transceivers

Yuyang Du, Graduate Student Member, IEEE, Soung Chang Liew, Fellow, IEEE, and Yulin Shao, Member, IEEE

Abstract—Interleaved Frequency Division Multiple Access (IFDMA) has the salient advantage of lower Peak-to-Average Power Ratio (PAPR) than its competitors like Orthogonal FDMA (OFDMA). A recent research effort of ours put forth a new IFDMA transceiver design significantly less complex than conventional IFDMA transceivers. The new IFDMA transceiver design reduces the complexity by exploiting a certain correspondence between the IFDMA signal processing and the Cooley-Tukey IFFT/FFT algorithmic structure so that IFDMA streams can be inserted/extracted at different stages of an IFFT/FFT module according to the sizes of the streams. Although our prior work has laid down the theoretical foundation for the new IFDMA transceiver's structure, the practical realization of the transceiver on specific hardware with resource constraints has not been carefully investigated. This paper is an attempt to fill the gap. Specifically, this paper puts forth a heuristic algorithm called multi-priority scheduling (MPS) to schedule the execution of the butterfly computations in the IFDMA transceiver with the constraint of a limited number of hardware processors. The resulting FFT computation, referred to as MPS-FFT, has a much lower computation time than conventional FFT computation when applied to the IFDMA signal processing. Importantly, we derive a lower bound for the optimal IFDMA FFT computation time to benchmark MPS-FFT. Our experimental results indicate that when the number of hardware processors is a power of two: 1) MPS-FFT has near-optimal computation time; 2) MPS-FFT incurs less than 44.13% of the computation time of the conventional pipelined FFT.

Index Terms—IFDMA, FFT, precedence graph, task scheduling.

I. INTRODUCTION

INTERLEAVED Frequency Division Multiple Access (IFDMA) is a broadband signal modulation and multiple-access technology for advanced wireless communications systems. Compared with other similar technologies [1], [2], such as Orthogonal FDMA (OFDMA), a salient feature of IFDMA is that its signal has a low Peak-to-Average Power Ratio (PAPR) [3]. The low PAPR of IFDMA brings about low signal distortion and high power efficiency when amplified by a power amplifier [4], making it a promising "green technology" for future wireless communications systems in which energy efficiency is a concern [5]. For the widespread deployment of IFDMA, however, the complexity of IFDMA transceivers needs to be minimized.

Our prior works in [6], [7] put forth a new class of IFDMA transceivers that is significantly less complex than conventional IFDMA transceivers. In essence, the design in [6], [7] allows a single IFFT/FFT module to perform the multiplexing/demultiplexing of multiple IFDMA data streams of different sizes, obviating the need to have IFFT/FFT modules of different sizes to cater to different IFDMA data streams. The key is to insert (for multiplexing at the transmitter) and extract (for demultiplexing at the receiver) the IFDMA data streams at different stages of the IFFT/FFT module according to their sizes.

In this paper, we refer to our IFDMA transceiver design in [6], [7] as "compact IFDMA" to capture the fact that it performs IFDMA functions in a compact and efficient manner. We refer to the specially designed IFFT/FFT in compact IFDMA as "compact IFDMA IFFT/FFT".

Although [6], [7] have laid down the theoretical foundation for compact IFDMA, its implementation in specific hardware (e.g., ASIC and FPGA) under resource constraints presents several technical challenges yet to be addressed. In [6], [7], we assumed perfect parallelism in the compact IFDMA FFT computation (i.e., unlimited number of processors). However, when the computation is to be implemented in an ASIC or FPGA, having abundant hardware to process the butterfly computations in parallel may not be possible, particularly when the number of subcarriers is large. Saving the “die area” is a vital issue in chip design [8], and the engineering design may be forced to use fewer processors for butterfly computations. Consequently, the amount of parallelism that can be achieved is limited, and scheduling the butterfly computation processes in compact IFDMA IFFT/FFT to minimize the computation latency is an issue.

Although mature hardware implementations of conventional IFFT/FFT are available [9], [10], [11], they are not well matched to the compact IFDMA IFFT/FFT structure. As will be verified by the experimental results in Section V, conventional IFFT/FFT implementations incur a much longer computation latency than our specially designed compact IFDMA IFFT/FFT.

To achieve low latency, our IFDMA IFFT/FFT implementation schedules the butterfly computations within an IFFT/FFT network by taking into account 1) the precedence relationships...
among the butterfly computations; and 2) the fact that not all butterfly computations within a full IFFT/FFT network need to be executed in the multiplexing and demultiplexing of IFDMA data streams.

This paper focuses on the design of compact IFDMA FFT and skips the design of IFFT. The Cooley-Tukey FFT and IFFT have similar decomposition structures, and the design of compact IFDMA IFFT is similar to that of FFT. Interested readers are referred to Section IV and V of [6] for details on the slight differences between compact IFDMA FFT and IFFT.

A. Quick Review and Motivation

We first briefly review how the compact IFDMA receiver works. Consider a scenario in which there are three users, A, B, and C, in an eight-subcarrier IFDMA system. Users A, B, and C require four subcarriers, three subcarriers, and one subcarrier, respectively. With the resource allocation scheme expounded in [7], a four-subcarrier IFDMA stream A1 is assigned to user A; a two-subcarrier IFDMA stream B1 plus a single-subcarrier IFDMA stream B2 are assigned to user B; and a single-subcarrier IFDMA stream C1 is assigned to user C. Fig. 1 shows the receiver’s FFT [6] responsible for extracting/demultiplexing the respective IFDMA streams from the IFDMA signals.

As illustrated in Fig. 1, not all butterfly computations need to be executed. For example, data stream A1 can be extracted after the first-stage butterfly computations associated with it. The required data are extracted by four tap buses, bypassing the subsequent butterfly computations. Black arrows indicate the tap busses in Fig. 1 (details on the tap bus design can be found in Fig. 7 of [6]). Besides A1, Fig. 1 also shows the early extraction of B1. In general, only the IFDMA streams with one subcarrier need to go through the full FFT operations from the first stage to the last stage. In Fig. 1, all the butterfly computations outside the blue dashed line can be pruned.

In general, the butterfly computations that can be omitted depend on the users’ subcarrier allocation. Fig. 2 shows another example, in which two users (A and B) are each allocated four subcarriers. During the operation of an IFDMA system, users may come and go, and their subcarrier demands may vary from time to time. This calls for a quick scheduling algorithm to determine the order of the butterfly computations in accordance with the active IFDMA subcarrier allocation at any moment in time, with the target of minimizing the computation latency.

Conventional FFT implementations, however, are too rigid to serve the scheduling purpose in compact IFDMA FFT. Three conventional FFT implementations are the serial FFT implementation [9], the pipelined FFT implementation [10], and the parallel FFT implementation [11]. In these implementations, butterfly computations are executed in a fixed and rigid manner, i.e., from left to right, and from top to bottom. Although these fixed schedules work well in conventional FFTs that execute all butterfly computations, the pruned FFT structure resulting from compact IFDMA is less regular because the numbers of butterfly computations needed at different stages are different depending on the subcarrier allocations to the IFDMA data streams. The implication is that to fully exploit the potential efficiency that arises from the pruned butterfly computations, the remaining unpruned butterfly computations need to be scheduled carefully to reduce the time needed to finish the job. Otherwise, in the worst case, the time required would still be the same as that in the conventional FFT due to the precedence constraints among the remaining unpruned butterfly computations.

This paper centers on a flexible computation scheduling algorithm tailored for IFDMA FFT. In general, task scheduling refers to the scheduling of a set of tasks with precedence relationships among them. A task cannot proceed before the inputs of the task are available, and the inputs of one task may be outputs of some preceding tasks. If we treat each butterfly computation as a task, it is then obvious that there are precedence relationships among the butterfly computations in the structure of the FFT. The precedence relationships can be captured by a directed acyclic graph (DAG) in which the vertexes are the tasks, and the directed edges indicate the precedence relationships between tasks. We refer to this special precedence graph as the FFT precedence graph.

In FPGA implementation, for example, we could design a customized butterfly computation processor and have a fixed number of such processors within the FPGA. The goal of the scheduling algorithm is to find an execution order for the tasks that can minimize the overall IFDMA FFT computation, subject to constraints of the precedence relationships and the number of processors.

The general scheduling problem is NP-complete. However, we will provide evidence that our scheduling algorithm for IFDMA FFT is near-optimal. The algorithm 1) works
well with arbitrary numbers of processors; 2) works well for random subcarrier allocations; and 3) has much lower computation time than conventional FFT implementations.

B. Related Works

Task scheduling subject to precedence constraints is tough in general. Karp first studied this problem in his seminal paper [12], where the task scheduling problem is reduced to a 3-satisfiability (3SAT) problem and proved to be one of the first 21 NP-complete problems. Subsequently, [13] found that task scheduling to minimize the computation time subject to a fixed number of processors and a uniform task processing time is also NP-complete. Our scheduling problem, unlike the general scheduling problem, is specific to the FFT precedence graph resulting from the IFDMA pruned FFT.

The first category of prior works concerns task scheduling in conventional unpruned FFT with the objective of computation speedup over specialized hardware. For example, [14] improved the butterfly task scheduling in a very-long-instruction-word (VLIW) digital signal processors (DSP) chip using a software pipelining technique called modulo scheduling. This scheduling algorithm exploits the instruction-level parallelism (ILP) feature in the VLIW DSP platform to schedule multiple loop iterations in an overlapping manner [15]. In [16], [17], the authors put forth the celebrated Fastest Fourier Transform in the West (FFTW) C library for the FFT implementation on multiprocessor computers. Other works like UHFFFT [18] and SPIRAL [19] also contributed to the FFT implementation on multiprocessor computers and DSP chips. What distinguishes our work from the above works is that our design considers pruned FFT arising from compact IFDMA.

Another category of prior works concerns task scheduling in pruned FFT [20], [21], [22], [23], [24], but not that arising from compact IFDMA. A key difference that distinguishes our work from these prior works is that for IFDMA, we want selected intermediate data at various stages inside the FFT flow graph. These intermediate data extractions, and the grouping of the intermediate data, bring forth a completely new structure for the pruned FFT.

Three types of conventional FFT pruning schemes were studied in previous works. The first is input-pruned FFT [20], where some computations can be eliminated when the FFT inputs have many zeros. The second is output-pruned FFT [21], where some computations can be eliminated when some of the FFT outputs are not needed. The third combines the aforementioned input pruning and output pruning [22].

In short, the above FFT pruning works take away the inputs with no data and take away the outputs not required. In [23], the authors studied the scheduling problem when many FFT inputs are zero. In [24], the authors investigated the efficient task scheduling in OFDMA applications, where only a small number of the FFT outputs are wanted by a user.

In our FFT implementation for IFDMA application, however, we want to extract selected data at various stages within the FFT flow graph. In the context of the complete FFT computation, these are intermediate computation results rather than the final outputs. For example, in Fig. 1, four intermediate outputs of the first-stage butterfly computations are wanted for IFDMA stream A1. That leads to a very different structure than pruned graphs studied by prior works. The new FFT pruning structure calls for new scheduling algorithms that can exploit the particular structure induced by IFDMA.

C. Contributions and Findings

We summarize the contributions of this paper in the following. We put forth an algorithm, referred to as multi-priority scheduling (MPS), to find a optimal task execution schedule for compact IFDMA FFT in terms of computation time. To establish the near optimality of MPS, we derive an analytical lower bound for the computation time and show that MPS can almost always find schedules that meet the lower bound. We refer to the FFT computation scheduled by MPS as MPS-FFT. We conducted large-scale experiments to comprehensively study the performance of MPS-FFT with up to 1024 subcarriers. Two key experimental results of MPS-FFT are as follows:

1) When the number of processors is a power of two, MPS-FFT subject to a randomly selected IFDMA subcarrier allocation can reach the computation-time lower bound with probability approaching one. Quantitatively, the probability is larger than \( c_0 \cdot \sqrt{0.05} \), where \( c_0 \) is more than 2.5 million.

2) When the number of processors is arbitrary, MPS-FFT subject to a randomly selected IFDMA subcarrier allocation can reach the computation-time lower bound with probability of at least 98.70%. In the few cases that fail to reach the lower bound, the computation time is at most 6% larger than the lower bound.

Our experiments also demonstrate that MPS-FFT has a much shorter computation time than the conventional FFT schemes do. The speedup achieved by MPS-FFT can be attributed to two factors: (i) the pruning scheme of MPS-FFT reduces more than 11.21% of the butterfly computations on average; (ii) MPS-FFT of 64-point or larger sizes have a more than 98.42% processor utilization rate when the number of processors is a power-of-two, almost twice higher than conventional FFT implementations with the same number of processors. Thanks to the above two benefits, the computation time of MPS-FFT with a number of power-of-two processors reduces the computation time of conventional FFTs with the same number of processors by 59.86% on average.

II. PROBLEM FORMULATION AND DEFINITIONS

A. Problem Formulation

We consider an \( N \)-point FFT in a compact IFDMA receiver. In the FFT network, there are \( n = \log_2 N \) FFT stages, with each stage having \( 2^{n-1} \) butterfly tasks.

For IFDMA subcarrier allocation, it is convenient to first map subcarriers to “bins” through a bit-reversal mapping so that the subcarrier indexes, after bit reversal, become the bin indexes, and subcarrier allocation becomes bin allocation (see Section III of [7] on bit-reverse index mapping of subcarriers.
to bins). This mapping process allows an IFDMA stream to be allocated consecutive bins that correspond to regularly-interpersed subcarriers, as is needed for the IFDMA stream.

A bin allocation can be expressed by an ordered list \( \{S_0, S_1, \ldots, S_{R-1}\} \), where \( R \) is the number of IFDMA streams, and \( S_r \) (\( r = 0, \ldots, R-1 \)) is the set of bins allocated to IFDMA stream \( r \). Fig. 3 gives an example of the construction of the FFT precedence graph given an IFDMA bin allocation. Fig. 3(a) shows a specific bin allocation for a system with 16 subcarriers and 11 IFDMA streams. IFDMA stream 0 is allocated the bins \( S_0 = \{0,1,2,3\} \), IFDMA stream 1 is allocated the bins \( S_1 = \{4,5\} \), and so on.

Fig. 3(b) shows the corresponding IFDMA FFT network for the receiver, where the green boxes are the places from which the data of the IFDMA streams are extracted. Fig. 3(c) shows the corresponding FFT precedence graph. A vertex in the precedence graph corresponds to a \( 2 \times 2 \) butterfly computation in the FFT network.

We next explain how the precedence graph in Fig. 3(c) is obtained. Consider stage \( i \) (\( i = 0,1,\ldots, \lfloor \log_2 N-1 \rfloor \)) of the FFT network. We denote task \( j \) (\( j = 0,1,\ldots, N/2-1 \)) and FFT output \( k \) (\( k = 0,1,\ldots, N-1 \)) at stage \( i \) by \( v_{i,j} \) and \( d_{i,k} \), respectively. With respect to Fig. 3(b), the butterfly computation in blue corresponds to task 6 (enumerating from top to bottom) at stage 2 (enumerating from left to right). Thus, the task is written as \( v_{2,6} \), and the outputs of this task are written as \( d_{2,12} \) and \( d_{2,14} \). The index \( k \) in \( d_{i,k} \) is labeled from top to bottom, with \( k = 0 \) corresponding to a rectangular box at the top and \( k = 15 \) corresponding to a rectangular box at the bottom.

Let \( D \) denote the set of desired FFT outputs of all IFDMA streams, i.e., green boxes in Fig. 3(b). Further, let \( V \) denote the set of tasks that need to be executed to obtain these desired FFT outputs. The precedence graph in Fig. 3(c) is obtained as follows.

In Fig. 3(b), to obtain output \( d_{3,8} \) required by IFDMA stream \( S_3 \), task \( v_{3,4} \) needs to be completed. However, before task \( v_{3,4} \) can be executed, tasks \( v_{2,4} \) and \( v_{2,5} \) need to be executed first, because their outputs \( d_{2,8} \) and \( d_{2,9} \) are inputs to \( v_{3,4} \). Further, before \( v_{2,4} \) and \( v_{2,5} \) can be executed, some tasks in the prior stage need to be executed, and so on and so forth.

In Fig. 3(b), we mark all of the butterflies that need to be executed to obtain \( d_{3,8} \) in pink. The vertices in Fig. 3(c) that correspond to these butterfly tasks are also pink. By tracing backward from the desired IFDMA stream outputs, we can obtain all the tasks that need to be executed and the precedence relationships between them to arrive at the overall precedence graph.

We note that every task except those at stage 0 requires the outputs of two specific tasks in the prior stage as its inputs, and the task cannot be executed unless these two preceding tasks are completed first. We define the concept of parent vertex and child vertex as follows:

**Definition 1 (Parent Vertex and Child Vertex):** If \( v_{i+1,j_0} \) cannot be executed unless \( v_{i,j_1} \) and \( v_{i,j_2} \) are completed, we refer to \( v_{i+1,j_0} \) as a child of \( v_{i,j_1} \) and \( v_{i,j_2} \). Tasks \( v_{i,j_1} \) and \( v_{i,j_2} \) are referred to as parents of \( v_{i+1,j_0} \).

Formally, we define the FFT precedence graph together with its associated notations as follows:

**Definition 2 (FFT Precedence Graph):** A precedence graph \( G(V, E) \) describes the task dependencies between a set of tasks, where

1. Vertex \( v_{i,j} \in V \) represents task \((i,j)\) within \( V \).
2. There is an edge \((v_{i,j_1} \rightarrow v_{i+1,j_2}) \in E \) if task \( v_{i,j_1} \) must be executed before task \( v_{i+1,j_2} \) can be executed, i.e., \( v_{i,j_1} \) is a parent of \( v_{i+1,j_2} \).

Since butterfly tasks are executed by processors with the same capacity, the processing times of tasks are the same (i.e., each vertex in the precedence graph takes the same amount of time to execute). We refer to the execution time of a task as one time slot.

With the above backdrop, we define the compact IFDMA FFT scheduling problem as follows:

**Definition 3 (The FFT Scheduling Problem in Compact IFDMA):** Given

1. an FFT precedence graph \( G(V, E) \) associated with a set of IFDMA streams together with their bin allocations, and
2. \( M \) processors, each of which can complete one task in exactly one time slot,
   
   determine a task-execution schedule \( X = (\chi_0, \ldots, \chi_{T-1}) \), where \( \chi_t \) is the set of tasks to be executed in time slot \( t \), and \( T \) is the completion time of the collection of tasks, such that,
   
   1. the precedence relationships are satisfied;
   2. there are no more than \( M \) tasks in each of \( \chi_t \), and \( \bigcup_t \chi_t \) consists of all the tasks in the precedence graph;
   3. \( T \) is minimized.

Henceforth, unless stated otherwise, we refer to the above FFT scheduling problem in compact IFDMA systems simply as the “scheduling problem”. We refer to an algorithm to solve the scheduling problem as a “scheduling algorithm”.

**Remark 1:** The schedule \( X = (\chi_0, \ldots, \chi_{T-1}) \), once determined, may be reused for a duration of time. The reason is as follows. In many applications, the set of IFDMA streams remains the same for a duration of time. As long as the bin allocations do not change, the subcarriers used by each of the streams remain the same. Each stream uses the same subcarriers to transmit its successive IFDMA symbols. An IFDMA FFT needs to be computed for each symbol period for the collection of IFDMA streams. However, the precedence graph remains the same for the successive FFTs. Thus, once we obtain schedule \( X \), the same schedule can be used to perform the
**IFDMA FFT for the successive IFDMA symbols. In particular, there is no need to compute X and build new FFT precedence graphs repeatedly. If the set of IFDMA streams changes (e.g., an IFDMA stream ends and leaves, or a new IFDMA stream arrives to join the existing IFDMA stream), a new schedule X will need to be computed.**

**B. Feasibility of a Task-Execution Schedule**

We next describe a step-by-step procedure to check whether a given task-execution schedule \( X = \{\chi_0, \ldots, \chi_{T-1}\} \) is feasible according to criteria 1) and 2) in Definition 3. Our algorithm (to be described later) follows this step-by-step procedure to construct the schedule to make sure it adheres to the feasibility criteria in each step of the way. In essence, the step-by-step procedure starts with \( \chi_0 \) and proceeds progressively to \( \chi_1, \chi_2, \ldots \), updating the precedence graph along the way to make sure that the precedence relationships are obeyed each step of the way.

Consider the beginning of time slot \( t \), the vertexes in \( \chi_t \) must have no parents and there are no more than \( M \) tasks in \( \chi_t \). If this is not satisfied, then the schedule \( X \) is not feasible. If \( \chi_t \) passes the test, we then remove the vertexes in \( \chi_t \) together with their output edges from the precedence graph. The updated precedence graph is used to check the feasibility of \( \chi_{t+1} \). We say that \( X \) is a feasible schedule if and only if the precedence graph is nil after vertex removal in time slot \( T - 1 \).

**C. Optimality of a Heuristic Scheduling Algorithm**

Given a feasible schedule \( X \), the next question one may ask is whether the schedule is optimal in that the overall computation time \( T \) is minimized. Recall from the introduction that the scheduling problems associated with the general non-FFT precedence graph have been proven to be NP-complete [12], [13]. Designing good heuristic algorithms for them with provable bounds is challenging.

The FFT precedence graphs, however, are specific precedence graphs with a regular structure. Section III gives a heuristic algorithm that solves the FFT scheduling problem to produce a schedule \( X = \{\chi_0, \ldots, \chi_{T-1}\} \). Section IV gives a lower bound on \( T \) (denoted by \( T^L \)) for a given FFT precedence graph. Section V shows by experiments that our scheduling algorithm, MPS, when given a randomly selected subcarrier allocation instance (note: each subcarrier allocation instance can be mapped to a corresponding FFT precedence graph), can find a schedule that can reach the computation-time lower bound with very high probability.

We characterize the optimality of a scheduling algorithm by a tuple \((\eta_{n,M}, \gamma_{n,M})\) defined as follows. For a given FFT scheduling problem with \( 2^n \) subcarriers and \( M \) processors, if a scheduling algorithm produces a feasible schedule \( X \) with \( T = T^L \), then the schedule is optimal. Among all subcarrier allocation instances tested, we denote the percentage of schedules produced by a heuristic algorithm that satisfies \( T = T^L \) by \( \eta_{n,M} \). Specifically,

\[
\eta_{n,M} = E\left[Pr(T = T^L)\right]
\]

(1)

where \( Pr(\cdot) \) is the probability of an event, and \( E[\cdot] \) is the expectation of a random variable. The probability of finding the optimal schedule for a random subcarrier allocation instance is no less than \( \eta_{n,M} \) (note that \( T > T^L \) does not mean the schedule is not optimal, since \( T^L \) is just a lower bound).

For the remaining \( 1 - \eta_{n,M} \) fraction of the schedules with \( T > T^L \), the gap between \( T \) and the best possible computation time \( T^* \) is \( T - T^* \leq T - T^L \). The percentage gap between \( T \) and \( T^* \) is \( (T - T^*)/T^* < (T - T^L)/T^L \). We define

\[
\gamma_{n,M} = E\left[\frac{(T - T^L)}{T^L}\right]
\]

(2)

to be the upper bound of the expected percentage gap, i.e., the average percentage gap between \( T \) and \( T^* \) is no more than \( \gamma_{n,M} \).

The input to a scheduling algorithm is an FFT precedence graph. As far as scheduling algorithms are concerned, FFT precedence graphs that are isomorphic to each other are equivalent. If a scheduling algorithm finds the optimal schedule for an FFT precedence graph, the optimal scheduling also applies to an isomorphic FFT precedence graph after an isomorphic transformation. Appendix A.1 of our technical report [25] delves into the isomorphism of FFT precedence graphs.

When running experiments over scheduling algorithms by subjecting them to different FFT precedence graphs, it would be desirable to remove isomorphism so that graphs with many isomorphic instances are not over-represented.

We denote the complete set of FFT precedence graphs by \( F_n \), wherein all elements are non-isomorphic. The cardinality of \( F_n \) is written as \( f_n \). We refer to the bin allocations that lead to isomorphic FFT precedence graphs as isomorphic bin allocations. Among isomorphic bin allocations, we only select one to put into \( F_n \). We refer the reader to Appendix A.1 of our technical report [25] on how we do so.

Appendix A.2 of our technical report [25] explains the principle to generate the complete set of non-isomorphic instances \( F_n \) for various \( n \). When \( f_n \) is not large, it is not an issue to use \( F_n \), the complete set, as the test set to conduct experiments and obtain \( \eta_{n,M} \) and \( \gamma_{n,M} \). However, as shown in Appendix A.2 of [25], \( f_n \) becomes prohibitively large when \( n \) is more than 6. Table I lists the \( f_n \) values for various \( n \), which are also derived in Appendix A.2.

### Table I

| \( n \) | FFT Size \((N)\) | Number of Non-isomorphic Instances \((f_n)\) |
|-------|----------------|----------------------------------|
| 1     | 2              | 2                                |
| 2     | 4              | 4                                |
| 3     | 8              | 11                               |
| 4     | 16             | 67                               |
| 5     | 32             | 2279                             |
| 6     | 64             | 2598061                          |
| 7     | 128            | 3.3750E + 12                    |
| 8     | 256            | 5.6952E + 24                    |
| 9     | 512            | 1.6218E + 49                    |
| 10    | 1024           | 1.3151E + 98                    |
As shown, when \( n \geq 7 \), the number of instances in \( F_n \) is so large that it is impractical to generate all the instances and use the complete set of non-isomorphic instances as the test set. Hence, for \( n \geq 7 \), instead of the full set \( F_n \), we randomly generate a subset of \( F_n \) (referred to as \( F'_n \)) to serve as our test set. Appendix A.3 of [25] explains how we randomly generate the subset of non-isomorphic instances for \( n \geq 7 \).

For \( n \geq 7 \), we can perform tests on \( F'_n \) and characterize \((\eta_{n,M}, \gamma_{n,M})\) statistically. We first consider \( \eta_{n,M} \). For a 2\(^n\)-point IFDMA FFT with \( M \) available processors, we randomly sample \( \xi_0 \) instances with replacement from \( F'_n \). Define \( Y_{\xi_1} \) to be the event that \( \xi_1 \) of the \( \xi_0 \) instances are instances for which the heuristic algorithm can find a solution reaching the lower bound. The conditional probability of \( Y_{\xi_1} \), given \( \eta_{n,M} \) is

\[
P(Y_{\xi_1}|\eta_{n,M}) = \left( \frac{\xi_0}{\xi_1} \right) \eta_{\xi_1}(1 - \eta_{\xi_1})^{\xi_0 - \xi_1}.
\]  

(3)

We are interested in the \textit{a posteriori} probability density function (PDF) \( p(\eta_{n,M}|Y_{\xi_1}) \). We have, by Bayes’ rule,

\[
p(\eta_{n,M}|Y_{\xi_1}) = \frac{p(\eta_{n,M}) \cdot P(Y_{\xi_1}|\eta_{n,M})}{P(Y_{\xi_1})}.
\]  

(4)

However, we do not know the \textit{a priori} PDF \( p(\eta_{n,M}) \). As a conservative measure, we can assume the worst case of having no knowledge on \( p(\eta_{n,M}) \) and let \( p(\eta_{n,M}) = 1, \forall \eta \in [0,1] \). Thus,

\[
p(\eta_{n,M}|Y_{\xi_1}) = \frac{P(Y_{\xi_1}|\eta_{n,M})}{P(Y_{\xi_1})}.
\]  

(5)

As a probability density, \( p(\eta_{n,M}|Y_{\xi_1}) \) must integrate to 1, i.e., \( \int p(\eta_{n,M}|Y_{\xi_1})d\eta_{n,M} = 1 \). This gives,

\[
\int \frac{P(Y_{\xi_1}|\eta_{n,M})}{P(Y_{\xi_1})}d\eta_{n,M} = 1
\]

\( \Leftrightarrow P(Y_{\xi_1}) = \int P(Y_{\xi_1}|\eta_{n,M})d\eta_{n,M}. \)

(6)

Thus, we have

\[
p(\eta_{n,M}|Y_{\xi_1}) = \frac{P(Y_{\xi_1}|\eta_{n,M})}{\int_0^1 P(Y_{\xi_1}|\eta_{n,M})d\eta_{n,M}}
\]

\[
= \frac{\eta_{\xi_1}(1 - \eta_{\xi_1})^{\xi_0 - \xi_1}}{\int_0^1 \eta_{\xi_1}(1 - \eta_{\xi_1})^{\xi_0 - \xi_1}d\eta_{n,M}}
\]

\[
= \frac{\eta_{\xi_1}(1 - \eta_{\xi_1})^{\xi_0 - \xi_1}}{B(\xi_1 + 1, \xi_0 - \xi_1 + 1)}
\]

(7)

where \( B(m, n) = \int_0^1 x^{m-1}(1-x)^{n-1}dx \) is the well-known Beta function.\(^1\)

Let \( \alpha \) be the target confidence, let \( \eta_{\xi_1, M}^L \) be the lower bound of \( \eta_{n,M} \), and let \( \int_{\eta_{\xi_1, M}^L}^{1} p(\eta_{n,M}|Y_{\xi_1})d\eta_{n,M} = \alpha \). We then have

\[
\int_{\eta_{\xi_1, M}^L}^{1} \left( \frac{\eta_{\xi_1}^{\xi_1}(1 - \eta_{\xi_1})^{\xi_0 - \xi_1}}{B(\xi_1 + 1, \xi_0 - \xi_1 + 1)} \right) d\eta_{n,M} = \alpha
\]

\( \Leftrightarrow I_{\eta_{M}, \lambda}(\xi_1 + 1, \xi_0 - \xi_1 + 1) - 1 = -\alpha, \)

(8)

\(^1\)The following mathematical properties of the Beta function, regularized Beta function, and incomplete Beta function can be found in Section 6.2 and Section 26.5 of [26].

\[
I_{\eta_{M}, \lambda}(\xi_1 + 1, \xi_0 - \xi_1 + 1) = \frac{B(\eta_{\xi_1, M}^{\xi_1} + 1, \xi_0 - \xi_1 + 1)}{B(\xi_1 + 1, \xi_0 - \xi_1 + 1)}
\]

(9)

is the regularized Beta function, and

\[
B(\eta_{\xi_1, M}^{\xi_1}, \xi_1 + 1, \xi_0 - \xi_1 + 1)
\]

\[= \int_0^{\eta_{\xi_1, M}^{\xi_1}} x^{\xi_1+1}(1-x)^{\xi_0-\xi_1+1}dx \]

(10)

is the incomplete beta function. With partial integration, \( I_{\eta_{M}, \lambda}(\xi_1 + 1, \xi_0 - \xi_1 + 1) \) can be further written as,

\[
I_{\eta_{M}, \lambda}(\xi_1 + 1, \xi_0 - \xi_1 + 1)
\]

\[= \sum_{j=0}^{\xi_0+1} \frac{(\xi_0 + 1)!}{(\xi_0 + 1 - j)!} (\eta_{\xi_1, M}^{\xi_1})^j (1 - \eta_{\xi_1, M}^{\xi_1})^{\xi_0+1-j}. \]

(11)

We obtain \( \eta_{\xi_1, M}^{L} \) by applying (11) to the incomplete Beta function in (8).

We next consider how to obtain the \( \alpha \)-percentile confidence interval of \( \gamma_{n,M} \) for \( n \geq 7 \). For the \( \xi_0 - \xi_1 \) non-lower-bound-reaching schedules (if any), we denote the mean and variance of the observed percentage gap \( (T - T^L)/T^L \) by \( \mu \) and \( \sigma^2 \), respectively. With the central-limit theorem [27], we construct a random variable \( Z \):

\[
Z = \frac{\mu - \gamma_{n,M}}{\sigma/\xi_0 - \xi_1}.
\]

(12)

With confidence target \( \alpha \) and critical z-score \( z_{1-\alpha/2} \), we have

\[
P(-z_{1-\alpha/2} \leq Z \leq z_{1-\alpha/2}) = \alpha.
\]

(13)

Applying (12) to (13), we then say,

\[
\mu - z_{1-\alpha/2} \cdot \frac{\sigma}{\sqrt{\xi_0 - \xi_1}} \leq \gamma_{n,M} \leq \mu + z_{1-\alpha/2} \cdot \frac{\sigma}{\sqrt{\xi_0 - \xi_1}}.
\]

(14)

with confidence \( \alpha \).

III. THE MULTI-PRIORITY SCHEDULING ALGORITHM

This section delves into the structural properties of the FFT precedence graph. Leveraging the structural properties, we put forth a heuristic algorithm for the FFT scheduling problem.

A. Structural Properties of FFT Precedence Graphs

Several important structural properties of FFT precedence graphs are given below:

- **Property 1** (Number of Parents): Vertices at stage 0 have no parent. Vertices at other stages have two parents located at the preceding stage.

- **Property 2** (Number of Children): Vertices at stage \( \log_2 N - 1 \) (i.e., last stage) have no children. Vertices at other stages can have one, two, or no children. In Fig. 3(c), for example, \( v_{3,4} \) at the last stage has no children, while \( v_{1,0} \) has one child, \( v_{1,4} \) has two children, and \( v_{2,2} \) has no children.
exploits the structure of the FFT precedence graph. Fig. 5
we refer to vertex A as a selected companion of vertex B.
companion relationship in Definition 4, on the other hand, are
for execution and then removed from the FFT precedence
is executed. This is because if a parent vertex is selected
initial FFT precedence graph before the scheduling algorithm
Fig. 4. Legitimate parent-child structures within the FFT precedence graph.

- **Property 3 (Parents):** Consider vertex \( v_{i,j} \) at stage \( i (1 \leq i \leq \log_2 N - 1) \). Its two parents at stage \( i - 1 \) can be
  identified as follows:
  1) If \( j \mod (2^{\log_2 N - i}) < 2^{\log_2 N - i - 1} \), the two parents are
     \( v_{i-1,j} \) and \( v_{i-1,j+1} \).
  2) If \( j \mod (2^{\log_2 N - i}) \geq 2^{\log_2 N - i - 1} \), the two parents are
     \( v_{i-1,j-2^{\log_2 N - i - 1}} \) and \( v_{i-1,j} \).

- **Property 4 (Children):** Consider \( v_{i,j} \) at stage \( i (0 \leq i \leq \log_2 N - 2) \). Its children at stage \( i + 1 \) (if any) can be
  identified as follows:
  1) If \( j \mod (2^{\log_2 N - i - 1}) < 2^{\log_2 N - i - 2} \), the potential
     children are \( v_{i+1,j} \) and \( v_{i+1,j+1} \).
  2) If \( j \mod (2^{\log_2 N - i - 1}) \geq 2^{\log_2 N - i - 2} \), the potential
     children are \( v_{i+1,j-2^{\log_2 N - i - 2}} \) and \( v_{i+1,j} \).

Note that one or both of the potential children may be absent in the FFT precedence graph. The above properties lead to a
obvious observation that the FFT butterfly structure in turn
induces a certain partial butterfly pattern in the precedence
graph. This is, if \( v_{i,j} \) has two children, there must be another
vertex \( v_{i,j+1} \) sharing the same two children with \( v_{i,j} \). If
\( v_{i,j} \) has only one child, obviously, there is another parent
at stage \( i \) sharing the same child with \( v_{i,j} \).

With the above observations, we give the definitions of
“vertex pair” and “the companion of a vertex” below.

**Definition 4 (Vertex Pair and the Companion of a Vertex):**
If a parent vertex shares children with another parent vertex,
we refer to the two parent vertexes as a vertex pair. Each of
the parent vertexes is the companion of the other.

Fig. 4 illustrates the legitimate parent-child structures within
the FFT precedence graph. A vertex can have zero, one or
two children. Fig. 4(a) shows the case of two vertexes sharing
two common children. Fig. 4(b) and (c) show the cases of
two vertexes, each having one child, and that they share this
common child.

Note that the structures in Fig. 4 are only applicable to the
initial FFT precedence graph before the scheduling algorithm
is executed. This is because if a parent vertex is selected
for execution and then removed from the FFT precedence
graph, the structure of the graph changes. Vertex pairs and
the companion relationship in Definition 4, on the other hand, are
applicable to the precedence graph throughout the execution
of the scheduling algorithm. If vertex A is the companion of
vertex B, and it is selected for execution before vertex B,
we refer to vertex A as a selected companion of vertex B.

**B. The Heuristic Scheduling Algorithm**

We now construct a heuristic scheduling algorithm that
exploits the structure of the FFT precedence graph. Fig. 5
presents an example of the evolution of the FFT precedence
graph during the execution of our heuristic algorithm. The
example assumes the same IFDMA bin allocation as in
Fig. 5(a) of Section II. The vertexes are marked in different
colors to facilitate later explanation of our heuristic algorithm.
Here, we assume that the number of processors is three.
A scheduling algorithm begins with the initial precedence
graph shown in Fig. 5(a) and ends with the final precedence
graph shown in Fig. 5(i). Each of the subfigures from Fig. 5(b)
to Fig. 5(i) shows an intermediate FFT precedence graph at
the end of a time slot. When a vertex is removed from the
FFT precedence graph, we mark it as a dashed ellipse, the
removed edges are simply not shown.

We refer to our scheduling algorithm as multi-priority
scheduling (MPS). As the name suggests, MPS associates
each vertex (task) with a priority vector \((P_1, P_2, \ldots, P_H)\)
and a pair of priorities,\(P_i\) and \(P_{i+1}\) of tasks A and task B are the same,
but \(P_i\) of task A is higher than that of task B, then task A is of
higher priority than task B.

Throughout this paper, we study the MPS algorithm for
which the priority vector \((P_1, P_2, P_3, P_4)\) has four elements.
This algorithm was applied in the example of Fig. 5. Let us
now specify the elements in \((P_1, P_2, P_3, P_4)\) and explain how
they are used to yield the scheduling results in Fig. 5.

First, \(P_1\) of a vertex is the number of generations of descen-
dants it has, wherein “descendants” refers to its children, its
children’s children, and so on. Fig. 5(a) illustrates how we
count the number of generations. We mark the vertexes in
Fig. 5(a) with four different colors according to how many
generations of descendants they have. Vertexes with no child
are colored in white. Their parents have one generation of
descendants, and they are colored in light green. Vertexes in
green have two generations of descendants, and vertexes in
dark green have three generations of descendants.

The intuition for setting \(P_1\) as above is as follows. Recall
that the children of a vertex must be executed in a different
time slot than the vertex. Thus, the number of generations of
descendants of a vertex corresponds to the number of extra
time slots that are needed in addition to the time slot used
to execute the vertex. It will be advantageous to select the
vertex with more generations of descendants to free up the
dependencies of its descendants on it.

Let us look at Fig. 5. The precedence graph after time
slot 2 is shown in Fig. 5(c). Three tasks can be selected for
execution in time slot 3. Among the eight executable vertexes
\((v_{0,6}, v_{0,7}, v_{1,0}, \ldots, v_{1,5})\) in Fig. 5(c), MPS selects all of
the vertexes with high \(P_1\) value \((P_1(v_{0,6}) = P_1(v_{0,7}) = 3)\) and one
of the vertexes with medium \(P_1\) value \((P_1(v_{1,4}) = 2)\). Our
algorithm does not select the light green vertexes in time
slot 3 because their \(P_1\) is lower than those in green and dark
green.

Next, we present the specification of \(P_2\). We divide the
vertexes into three types. Childless type refers to vertexes with
no children. Paired type refers to vertexes for which both the
vertex itself and its companion have not yet been selected for
execution (see Definition 4 for the concept of companion).
Singleton type refers to unselected vertexes whose companion
has already been selected for execution in a prior time slot. Accordingly, $P_2$ is set to 0, 1, and 2, respectively.

The intuition for setting $P_2$ as above is as follows. Since the selection of childless vertices brings no new executable vertex in the next time slot, we give it the lowest $P_2$. For paired vertices, it takes two processors to execute them together in the same time slot and obtain new executable vertex(es) (i.e., the children) in the next slot. For singleton vertices, it takes only one processor to obtain the same number of new executable vertex(es) in the next slot. So, we give the middle and the highest $P_2$ to paired vertex and singleton vertex, respectively. Note that $P_2$ is a myopic measure of a vertex according to the row of the precedence graph at which it is located. Specifically, $P_2 = (N/2 - 1) - j$, where $j$ is the row that $v_{i,j}$ lies in.

In Fig. 5, the precedence graph at the end of time slot 7 is shown in Fig. 5(h). There are five executable vertices with the same $P_1$, $P_2$ and $P_3$ values. Among them, $v_{2,2}$ is located at the highest row, with its $P_3$ equal to 5. Hence, the selection order of the next time slot (time slot 8) is $v_{2,2} \rightarrow v_{2,3} \rightarrow v_{3,5}$. Instead of comparing the priority vectors, for realization, it will be more convenient to map the priority vectors to scalars so that we can compare scalars when prioritizing the associated tasks. Specifically, we map a priority vector $(P_1(v_{i,j}), P_2(v_{i,j}), P_3(v_{i,j}), P_4(v_{i,j}))$ to a priority scalar $P(v_{i,j})$ as follows:

$$P(v_{i,j}) = \sum_{h=1}^{H=4} \left( \frac{N}{2} \right)^{H-h} P_h(v_{i,j}).$$

Note that for our $H = 4$ MPS, the values of $P_1, P_3, P_4$ are fixed at the beginning of the scheduling and will never change as the algorithm proceeds. However, $P_2$ of a vertex $v_{i,j}$ may change dynamically, according to whether the companion vertex of $v_{i,j}$ has already been selected for execution.

Recall that a paired-type vertex shares one or two common children with another companion vertex. If a vertex has children in the initial precedence graph, it has an unselected companion vertex. Hence, there are only two types of vertexes in the initial FFT precedence graph: the childless type and the paired type. With the execution of our MPS algorithm, a paired vertex $v_{i,j}$ will change to a singleton vertex once its companion has been selected for execution. As a result, $P_2$ of a vertex can be 0 or 1 in the initial FFT precedence graph.
Algorithm 1 MPS Algorithm ($H = 4$)

Input: IFDMA bin allocation \{S₀, S₁, ..., S₆₋₁\} and M
Output: Task schedule $\mathbf{X} = \{\chi₀, \chi₁, ..., \chi₇₋₁\}$ and $T$

1: $t ← 0$ // Algorithm initialization
2: $(\Omega, G(V, E), (P₁, P₂, P₃, P₄)) ←$ initialize func.
3: while $G(V, E) \neq \emptyset$ do // Vertex selection
4: \hspace{1em} $u ← 0$
5: \hspace{1em} while $(u < M$ and $\Omega \neq \emptyset$) do // Vertex selection
6: \hspace{2em} $v_{i,j} ← \arg \max_{v \in \Omega} P(v)$
7: \hspace{2em} $\chi_i, \text{push}(v_{i,j})$
8: \hspace{2em} $\Omega, \text{pop}(v_{i,j})$
9: \hspace{2em} if $v_{i,j}$ has companion $v'_{i,j}$ and $v''_{i,j} ∈ \Omega$ then // If $v_{i,j}$ is selected, raise $P₂$ of $v'_{i,j}$
10: \hspace{3em} $P₂(v'_{i,j}) ← 2$
11: \hspace{3em} $P(v''_{i,j}) ←$ priority scalar calculation func.
12: \hspace{2em} end if
13: \hspace{1em} $u ← u + 1$
14: \hspace{1em} end while // End of vertex selection in one time slot
15: \hspace{1em} $V, \text{pop}(\text{elements in } \chi_i)$
16: \hspace{1em} $E, \text{pop}(\text{output edges of the elements in } \chi_i)$
17: \hspace{1em} $\Omega, \text{push}(\text{new executable vertexes})$
18: \hspace{1em} $t ← t + 1$
19: end while // End of a time slot

(dependents on whether it has children). Then, with the execution of MPS, the $P₂$ of a vertex can be adjusted to 2 accordingly.

Formally, the above description of the assignment of priority vectors, and the serialization for the computation of the priority scalar, can be summarized as follows: For $v_{i,j} ∈ V$, we denote the set of $v_{i,j}$’s children by $K(v_{i,j})$ and the cardinality of $K(v_{i,j})$ by $k_{i,j}$, we then have

1) $P₁$ assignment: if $k_{i,j} = 0$, then $P₁(v_{i,j}) = 0$; else $P₁(v_{i,j}) = \max_{v_{i,j+1} \in K(v_{i,j})} P₁(v_{i,j+1}) + 1$.
2) $P₂$ assignment (initial value): For $v_{i,j}$, if $k_{i,j} = 0$, then $P₂(v_{i,j}) = 0$; else $P₂(v_{i,j}) = 1$.
3) $P₂$ assignment (dynamically adjusted value): If $v_{i,j}$’s companion is newly selected for execution, let $P₂(v_{i,j}) = 2$.
4) $P₃$ assignment: For $v_{i,j}$, $P₃(v_{i,j}) = k_{i,j}$.
5) $P₄$ assignment: For $v_{i,j}$, $P₄(v_{i,j}) = N/2 - 1 - j$.
6) Priority scalar calculation: $P(v_{i,j}) = \sum_{h=1}^{H₄} \left(\frac{N}{2}\right)^{-h} P_h(v_{i,j})$.

We now denote the set of executable vertexes by $\Omega$ and use pseudo-code to describe the $H = 4$ MPS algorithm as follows. Note that Algorithm 1 is the main function of the MPS algorithm, Algorithm 2 is the initialization subfunction called at the beginning of Algorithm 1, Algorithm 3 is the priority scalar calculation subfunction described by (15), which is also called by Algorithm 1.

IV. IFDMA FFT Computation-Time Lower Bound

This section derives a general lower bound $T^L$ on the IFDMA FFT computation time. Any feasible schedule $\mathbf{X} = \{\chi₀, \chi₁, ..., \chi₇₋₁\}$ constrained by the FFT precedence graph and the number of available processors must have computation time $T ≥ T^L$.

Algorithm 2 Priority Scalar Calculation

Input: IFDMA bin allocation \{S₀, S₁, ..., S₆₋₁\}
Output: Initialized (1)FFT precedence graph; (2)executable vertexes set $Ω$; (3)priority vectors $(P₁, P₂, P₃, P₄)$

1: $G(V, E) ←$ build FFT precedence graph with\{S₀, ..., S₆₋₁\}
// See Section II-A on graph building
2: for $i = \log₂N - 1; i ≥ 0; i--$ do
3: \hspace{1em} for $j = 0; j ≤ \log₂N - 1; j++$ do
4: \hspace{2em} if $v_{i,j} ∈ V$ then
5: \hspace{3em} $K(v_{i,j}) ←$ child of $v_{i,j}$
6: \hspace{3em} $k_{i,j} ←$ cardinality of $K(v_{i,j})$
7: \hspace{2em} else
8: \hspace{3em} $P₁(v_{i,j}) ← 0$
9: \hspace{3em} $P₂(v_{i,j}) ← 0$
10: \hspace{2em} end if
11: \hspace{2em} $P₁(v_{i,j}) ← 1 + \max_{v_{i,j+1,x} ∈ K(v_{i,j})} P₁(v_{i,j+1,x})$
12: \hspace{2em} $P₂(v_{i,j}) ← 1$
13: \hspace{1em} end if
14: \hspace{1em} $P₃(v_{i,j}) ← k_{i,j}$
15: \hspace{1em} $P₄(v_{i,j}) ← N/2 - 1 - j$
16: \hspace{1em} end if
17: \hspace{1em} if $i = 0$ then
18: \hspace{2em} $Ω, \text{push}(v_{i,j})$
19: \hspace{2em} end if
20: \hspace{1em} end for
21: end for

Algorithm 3 Priority Scalar Calculation

Input: Priority vector $(P₁(v_{i,j}), P₂(v_{i,j}), P₃(v_{i,j}), P₄(v_{i,j}))$
Output: Priority scalar $P(v_{i,j})$

1: $P(v_{i,j}) = \sum_{h=1}^{H₄} \left(\frac{N}{2}\right)^{-h} P_h(v_{i,j})$ // see eq. (15)

A trivial lower bound is,

$$T^L = \left\lceil \log_{\frac{N}{2}} \left(\sum_{i=0}^{N-1} Q_i \right) / M \right\rceil,$$

where $[\cdot]$ is the round-up operation, and $Q_i$ is the number of vertexes at stage $i$. This is, however, a rather loose lower bound.

In the following, we prove a tight lower bound for $T$. Our argument divides the vertexes into "trunk vertexes" and "branch vertexes", defined as follows:

**Definition 5 (Trunk Vertex and Branch Vertex):** In an initial FFT precedence graph with $β$ FFT stages, a vertex $v_{i,j}$ satisfies,

$$P₁(v_{i,j}) = β - 1 - i,$$

we refer to it as a trunk vertex; otherwise, we refer to it as a branch vertex.

Recall that in Section III, we define $P₁(v_{i,j})$ to be the number of generations of descendants vertex $v_{i,j}$ has. Since vertex $v_{i,j}$ is located at stage $i$, and the maximum number of

2Since many bin allocations do not require tasks in later FFT stages, we have $β ≤ \log₂N$ in general.
remaining stages after stage $i$ is $\beta - 1 - i$, in general we have $P_1(v_{i,j}) \leq \beta - 1 - i$. Hence, trunk vertexes at stage $i$ are vertexes at stage $i$ that have the maximum possible $P_1$.

Fig. 6 gives an illustration. Vertexes in blue are the trunk vertexes, and vertexes in red are the branch vertexes. The rationale for our nomenclature is as follows. We picture vertexes, and vertexes in red are the branch vertexes. The

Lemma 1: A trunk vertex $v_{i,j}$ with $i \in \{0, 1, \ldots, \beta - 2\}$, that is not located at the last stage (i.e., stage $\beta - 1$) has at least one child that is also a trunk vertex.

Proof: Since $v_{i,j}$ is a trunk vertex, we have

$$P_1(v_{i,j}) = 1 + \max_{v_{i+1,x} \in K(v_{i,j})} P_1(v_{i+1,x}) = \beta - i - 1,$$

where $K(v_{i,j})$ is the children set of $v_{i,j}$.

From (18), we know that at least one vertex $v_{i+1,x}$ satisfies,

$$P_1(v_{i+1,x}) = 1 + \max_{v_{i+1,x} \in K(v_{i,j})} P_1(v_{i+1,x}) = \beta - (i + 1) - 1,$$

i.e., $v_{i+1,x}$ is a trunk vertex.

Let $T_{tr}(i)$, $i = 0, 1, \ldots, \beta - 2$, be the time slot by which all trunk vertexes at stage $i$ are executed.

Lemma 2: $T_{tr}(i + 1) \geq T_{tr}(i) + 1$.

Proof: In general, some trunk vertexes at stage $i$ could be executed at time slots earlier than $T_{tr}(i)$. But there must be at least one trunk vertex $v_{i,j}$ at stage $i$ that is executed in time slot $T_{tr}(i)$. From Lemma 1, trunk vertex $v_{i,j}$ must have one child trunk vertex, say vertex $v_{i+1,x}$, at stage $i + 1$ that depends on $v_{i,j}$. The earliest time slot by which $v_{i+1,x}$ can be executed is $T_{tr}(i) + 1$. Thus, $T_{tr}(i + 1) \geq T_{tr}(i) + 1$.

Let $R_i$ be the number of trunk vertexes at stage $i$. Some stages have more than $M$ trunk vertexes, and some stages have no more than $M$ trunk vertexes. Let $U$ be the set of stages with more than $M$ trunk vertexes, and $W$ be the set of stages with no more than $M$ trunk vertexes.

Lemma 3: In an FFT precedence graph, the stages in $U$ precedes the stages in $W$.

Proof: A pair of trunk parent vertexes at stage $i$ have no more than two child trunk vertexes at stage $i + 1$. Each child trunk vertexes, on the other hand, must have two parent trunk vertexes. Hence, $R_i$, the number trunk vertexes at stage $i$, can be no smaller than $R_{i+1}$, the number of trunk vertexes at stage $i + 1$. Thus, the stages in $U$ precede the stages in $W$.

We denote the cardinality of the set $W$ by $m$ in the following. We also refer to the computation time required to execute all trunk vertexes as $T_{tr}$, and we denote the lower bound of $T_{tr}$ by $T_{tr}^L$. With the above lemmas, we obtain $T_{tr}^L$ as follows.

Lemma 4: $T_{tr} \geq T_{tr}^L \triangleq [\frac{\beta - m - 1}{M} R_i] + m$.

Proof: From Lemma 3, the first $\beta - m$ stages have more than $M$ in each stage, and their execution can finish no earlier than,

$$T_{tr} (\beta - m - 1) \geq \left[ \sum_{i=0}^{\beta - m - 1} R_i \right] \frac{1}{M} + m. \tag{20}$$

Applying Lemma 2 to stage $\beta - m$, we have

$$T_{tr} (\beta - m) \geq T_{tr} (\beta - m - 1) + 1 \geq \left[ \sum_{i=0}^{\beta - m - 1} R_i \right] \frac{1}{M} + m. \tag{21}$$

Similarly, for stage $\beta - m + 1$, we have $T_{tr} (\beta - m + 1) \geq T_{tr} (\beta - m) + 1$. And so on and so forth. The last trunk vertexes at the stage $\beta - 1$ can therefore finish execution no earlier than $\left[ \frac{\beta - m - 1}{M} R_i \right] + m$. Thus, we have

$$T_{tr} \geq T_{tr}^L \triangleq \left[ \sum_{i=0}^{\beta - m - 1} R_i \right] \frac{1}{M} + m. \tag{22}$$

The $T_{tr}^L$ in (22) would be a lower bound for the overall problem if there were no branch vertexes. We next take into consideration the computation time needed by the branch vertexes. When packing the vertexes into the schedule implicit in our procedure for obtaining $T_{tr}^L$ above, there could be time slots in which there are fewer than $M$ vertexes being executed. As far as a lower bound for the overall computation time is concerned, we could imagine that we could pack branch vertexes into such time slots for execution without constraints of the precedence graph. In other words, we imagine that the branch vertexes could fully exploit the unused capacities of the time slots allocated to the trunk vertexes. The lower bound obtained as such may not be achievable, but it is still valid as far as a lower bound.

The total number of branch vertexes is $\sum_{i=0}^{\beta - 1} (Q_i - R_i)$. If the unusued capacities is not enough for executing all branch vertexes, i.e., $M \cdot T_{tr}^L - \sum_{i=0}^{\beta - 1} R_i < \sum_{i=0}^{\beta - 1} (Q_i - R_i)$, we need extra time slots after $T_{tr}^L$ to execute the remaining branch vertexes. We refer to the computation time for executing the remaining branch vertexes as $T_{br}$, and we denote the lower bound of $T_{br}$ by $T_{br}^L$.

Lemma 5: $T_{br} \geq T_{br}^L \triangleq [l/M]$, where $l = \max \left\{ 0, \sum_{i=0}^{\beta - 1} Q_i - M \left[ \sum_{i=0}^{\beta - m - 1} R_i \right] \frac{1}{M} + m \right\}$.

Proof: There are $M \cdot T_{tr}^L$ processor capacities in the first $T_{tr}^L$ time slots. During the $T_{tr}^L$ time slots, the unused capacities
are given by,
\[ E = M \cdot T_{tr}^L - \sum_{i=0}^{\beta-1} R_i. \] (23)

After packing branch vertexes into \( T_{tr}^L \) time slots to used up the capacities \( E \), the number of remaining branch vertexes not packed into the \( T_{tr}^L \) time slots is,
\[
\begin{align*}
l &= \max \left\{ 0, \sum_{i=0}^{\beta-1} (Q_i - R_i) - E \right\} \\
&= \max \left\{ 0, \sum_{i=0}^{\beta-1} Q_i - M \cdot T_{tr}^L \right\} \\
&= \max \left\{ 0, \sum_{i=0}^{\beta-1} Q_i - M \left[ \sum_{i=0}^{\beta-1} R_i/M \right] + Mn \right\}. 
\end{align*}
\]

Thus, \( T_{br} \) is lower bounded by,
\[ T_{br} \geq T_{br}^L \triangleq \left\lfloor \frac{l}{M} \right\rfloor. \] (25)

**Theorem 1:** \( T \geq T_L \triangleq \left[ \frac{\beta^{m-1}}{\prod_{i=0}^{m-1} R_i} \right] M + \left\lfloor 1/M \right\rfloor + m. \)

**Proof:** Since \( T \geq T_{tr}^L + T_{br} \), we prove Theorem 1 with Lemma 4 and Lemma 5.

---

### V. Experiments and Analysis

The first subsection presents experimental results showing that the schedule found by the MPS algorithm is close-to-optimal in terms of the FFT computation time. The second subsection presents experimental results showing that MPS-FFT has a shorter computation time than conventional FFT.

In Section II, we proved that when the number of subcarriers is large, the number of non-isomorphic instances for IFDMA can be huge. It is impractical to test all possible subcarrier allocation instances. Hence, in the following two subsections, we analyze the performance of MPS-FFT statistically when the number of subcarriers is large. Specifically, when \( n \geq 7 \), we randomly sample \( f_6 = 2598061 \) of \( f_n \) non-isomorphic instances for testing, where \( f_n \) is the total number of non-isomorphic instances when there are \( 2^n \) subcarriers. For systems of 8 to 64 subcarriers (i.e., \( 3 \leq n \leq 6 \)), we test all non-isomorphic instances. The two-point \((n = 1)\) and four-point \((n = 2)\) FFT modules are trivial and are omitted in the experiments.

#### A. Optimality of MPS

This subsection focuses on the optimality of MPS. We assume a fully packed IFDMA system, in which all subcarriers are used. Recall from Section II that \( \eta_{n,M} \) denotes the percentage of MPS-FFT schedules that are optimal when the number of subcarriers is \( 2^n \) and the number of processors is \( M \). If the computation time does not reach the lower bound, we use \( \gamma_{n,M} \) to denote the average percentage gap between the observed computation time and the theoretical lower bound. As we have explained at the beginning of this section, if \( n \geq 7 \), we follow the analysis in Section II-C and statistically obtain \( \eta \) and \( \gamma \) with a confidence level of \( \alpha = 0.95 \).

**Case 1 (Arbitrary Number of Processors):** Let us first assume an arbitrary number of processors between one and \( 2^{n-1} \), the maximum processors needed. We denote the average value of \( \eta_{n,M} \) and \( \gamma_{n,M} \) over \( M \) by \( \eta_n \) and \( \gamma_n \), respectively. The experimental results of \( \eta_n \) and \( \gamma_n \) are given in Table II.

As can be seen from Table II, if we give MPS a random subcarrier allocation instance, MPS can find a schedule reaching the computation-time lower bound with high probability. Specifically, \( \eta_n \) is no less than 98.70%. Additionally, if a task-execution schedule does not reach the lower bound, it is still acceptable because the computation time is very close to the lower bound, i.e., \( \gamma_n \) is no more than 6.48%.

**Case 2 (Power-of-Two Number of Processors):** Note that we assume an arbitrary number of processors in Table II. We find from our experiments that for the case of \( M \) equal to a power-of-two integer, all our tested task-execution schedules reach the lower bound and are optimal. For \( n < 7 \), our algorithm is optimal in terms of the computation speed, as we tested all instances, and all the schedules reached the lower bound. For \( n \geq 7 \), the \( \xi_0 - \xi_1 = 0 \), (11) in Section II can be further simplified as follows:

\[
\begin{align*}
I_{n,L} (\xi_1 + 1, \xi_0 - \xi_1 + 1) \\
&= \frac{(\xi_0 + 1)!}{j!(\xi_0 + 1 - j)!} \left( \eta_{n,L}^{\xi_1 + 1 - j} (1 - \eta_{n,L}^{\xi_0}) \right) \\
&= \left( \eta_{n,L}^{\xi_0} \right)^{\xi_1 + 1}.
\end{align*}
\] (26)

Applying \( \xi_0 - \xi_1 = 0 \) to (8) and (11) in Section II, we have
\[ \eta_n = (1 - \alpha)^{1/\xi_0}. \] (27)

Therefore, when \( n \geq 7 \), we have \( \eta_n \geq (1 - \alpha)^{1/\xi_0} = 0.95 \), where \( \alpha = 0.95 \) is the confidence level, and \( \xi_0 = f_6 = 2598061 \). Additionally, we know that \( \gamma_n = 0 \) from the definition of \( \gamma_{n,M} \) in (2). The experimental results of \( \eta_n \) and \( \gamma_n \) are given in Table III.

The gap between \( \xi_0 \sqrt{0.05} \) and one is relatively small because \( \xi_0 \) is large enough, i.e., the lower bound of \( \eta_n \) is relatively close to one. Thus, we can say that our algorithm has

---

**Table II:** \( \eta_n \) and \( \gamma_n \) for the Scheduling Problems in a \( 2^n \)-Point IFDMA FFT (The Case of an Arbitrary Number of Processors)

| \( n \) | \( \eta_n \) | \( \gamma_n \) |
|-------|-----------|-----------|
| 3     | 1.0000    | 0.0000    |
| 4     | 0.9985    | 0.0250    |
| 5     | 0.9984    | 0.0460    |
| 6     | 0.9981    | 0.0648    |
| 7     | \( \geq 0.9972 \) | \( \leq 0.0391 \) |
| 8     | \( \geq 0.9942 \) | \( \leq 0.0146 \) |
| 9     | \( \geq 0.9991 \) | \( \leq 0.0121 \) |
| 10    | \( \geq 0.9870 \) | \( \leq 0.0104 \) |

---

**Table III:** \( \eta_n \) and \( \gamma_n \) for the Scheduling Problems in a \( 2^n \)-Point IFDMA FFT (The Case of a Power-of-Two Number of Processors)

| \( n \) | \( \eta_n \) | \( \gamma_n \) |
|-------|-----------|-----------|
| 3     | 1.0000    | 0.0000    |
| 4     | 0.9985    | 0.0250    |
| 5     | 0.9984    | 0.0460    |
| 6     | 0.9981    | 0.0648    |
| 7     | \( \geq 0.9972 \) | \( \leq 0.0391 \) |
| 8     | \( \geq 0.9942 \) | \( \leq 0.0146 \) |
| 9     | \( \geq 0.9991 \) | \( \leq 0.0121 \) |
| 10    | \( \geq 0.9870 \) | \( \leq 0.0104 \) |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE III

| n   | \( \eta_n \) | \( \gamma_n \) |
|-----|----------|----------|
| 3/4/5/6 | 1.0000  | 0.0000  |
| 7/8/9/10 | \( \leq \eta_n \)  | 0.0000  |

close-to-optimal performance when the number of processors is power-of-two.

B. Applying MPS-FFT in Compact IFDMA Transceivers

This subsection analyzes the effect of applying MPS-FFT to compact IFDMA transceivers. To distinguish the transceiver design in this paper from the design in our previous paper [6], let us refer to the compact IFDMA transceiver in our previous work and the one in the current paper as compact IFDMA transceiver v1 and v2, respectively.

As has been explained at the beginning of this section, for \( n \geq 7 \), we analyze the computation complexity/hardware utilization/computation time of MPS-FFT with a subset of \( f_n \) randomly sampled non-isomorphic subcarrier allocation instances rather than the full set of instances. We show in Appendix A.3 of our technical report [25] that our subset generation policy selects all the instances with equal probability. Hence, the analysis of computation complexity/hardware utilization/computation time is unbiased in terms of instance selection.

1) Computation Complexity: The computation complexity improvement of compact IFDMA transceiver v1 over conventional IFDMA transceivers has already been studied in Section IV and Section V of our previous work [6]. For readers’ convenience, we briefly summarize the results in Table IV. In the table, (i), (ii), and (iii) refer to conventional IFDMA transceivers with time-domain realization, conventional IFDMA transceivers with frequency-domain realization, and compact IFDMA transceivers v1, respectively. The computation complexity is measured in terms of the number of butterfly computations needed in a transmitter/receiver. As can be seen from Table IV, transceiver v1 reduces the complexity of a conventional IFDMA transceiver by at least a factor of \( \log_2 N \).

The current paper further reduces complexity by pruning the FFT flow graph. Fig. 7 below compares the computation complexities in compact IFDMA transceiver v2 and IFDMA transceiver v1. As the figure shows, transceiver v2 achieves at least 11.21% reduction in FFT computation complexity over transceiver v1.

2) Hardware Utilization and Computation Time: Although pruning the FFT flow graph reduces the computation complexity, it was only the first step. A more challenging study of compact IFDMA transceiver v2 is to increase the hardware utilization in a pruned FFT with a limited number of processors.

As we have explained, the goal of our scheduling algorithm is to produce a schedule with as short a computation time as possible for any given IFDMA subcarrier allocation. Since the number of hardware processors is limited, MPS achieves that goal by boosting the hardware utilization of the processors during the FFT processing. In the FFT precedence graph, one butterfly computation cannot be executed unless its two parents have been executed. In some rounds of butterfly computation during the FFT processing, it is possible that some processors have no work to do if the schedule is not well constructed. This is because there may not be enough executable computations for every processor to get a job in this round if the schedule is poorly constructed. The “idle time” of processors reduces the hardware utilization. Additionally, conventional FFT may execute unnecessary butterfly computations (i.e., those eliminated in our pruned FFT).

In essence, we define hardware utilization as \( \sum Q_i/M_T \), where \( Q_i \) is the number of necessary computation tasks in FFT stage \( i \), \( M_T \) is the overall computation time required by the schedule, and \( M_T \) is the total number of available hardware capacity over that time.

Fig. 8 benchmarks compact IFDMA transceiver v1 and v2 in terms of hardware utilization. The FFT module used in transceiver v2 is MPS-FFT designed in this paper. As for the FFT used in compact IFDMA transceiver v1, our prior work [6] investigated the computation complexity but did not consider the implementation issue. It just assumed a conventional FFT but did not assign a specific realization scheme. Basically, there are three types of conventional FFT implementations: the serial structure [9], the pipelined structure [10], and the parallel structure [11]. Among the three types, the pipelined FFT is the most widely used in practice, e.g., the same realization can be found in Xilinx’s high-efficiency FFT IP core for FPGA [28]. As for the other two types of FFT realizations, serial FFTs are too slow for the FFT processing in communication systems; parallel FFTs consume too much hardware resource and are not suitable for communication systems with many subcarriers.

Here IFDMA transceiver v1 adopts the pipelined FFT implementation, while IFDMA transceiver v2 adopts this paper’s MPS-FFT.

In Fig. 8, we consider IFDMA transceivers with \( N \) subcarriers and plot the figure with \( \log_2 N \) on the x-axis. Note that the number of processors in an \( N \)-point conventional pipelined FFT is fixed to be \( \log_2 N \). For a fair comparison,
we let the number of hardware processors in both transceiver v1 and transceiver v2 be $M = \log_2 N$.

As can be seen from the dashed line in Fig. 8, the hardware utilization of a conventional pipelined FFT is no larger than 50%. MPS-FFT, on the other hand, greatly increases the hardware utilization. Specifically, the hardware utilization rates of MPS-FFT with 64 or more points are at least 98.42%. And the utilization rate increases as number of subcarriers increases, reaching 99.82% in the 1024-point FFT case.

Table V compares compact IFDMA transceivers v1 and v2 in terms of computation time. We can see that compact IFDMA transceiver v2 achieves much lower FFT computation time than compact IFDMA transceiver v1 does. Specifically, the computation time can be reduced by at least 55.87%, and we can calculate that the average computation time reduction rate is 59.85%.

### VI. CONCLUSION

This paper studied the FFT implementation of an efficient IFDMA transceiver, referred to as compact IFDMA, put forth by a recent investigation. For compact IFDMA, not all butterfly computations inside the full FFT network need to be executed, and the necessary computations vary with different subcarrier allocations. When applied to IFDMA, conventional FFT implementation is resource-wasteful because it does not exploit this specific property of IFDMA signal processing.

This paper focused on FFT implementations tailored for IFDMA. We put forth a flexible heuristic algorithm to schedule the butterfly computations in IFDMA FFT, referred to as multi-priority scheduling (MPS). Compared with conventional FFT implementations, the FFT computation schedule obtained by MPS, referred to as MPS-FFT, has two advantages: 1) MPS-FFT reduces computation requirements. For example, for 1024-subcarrier IFDMA, MPS-FFT can bypass at least 11.21% of the computation tasks in FFT. 2) MPS-FFT utilizes hardware efficiently. For example, for 1024-subcarrier IFDMA, the processor utilization rate in MPS-FFT is 99.82%, much higher than the processor utilization rate in conventional pipeline FFT.

When the number of available processors is a power of two, MPS-FFT has near-optimal computation-time performance. Quantitatively, MPS-FFT subject to a randomly selected IFDMA subcarrier allocation can reach the computation time lower bound with a probability larger than $\xi_0 \sqrt{0.05}$, where $\xi_0$ is more than 2.5 million in our experiment. Furthermore, MPS-FFT incurs less than 44.13% of the computation time of the conventional pipelined FFT.

### REFERENCES

[1] S. Han, Y.-C. Liang, B.-H. Soong, and S. Li, “Dynamic broadband spectrum refarming for OFDMA cellular systems,” IEEE Trans. Wireless Commun., vol. 15, no. 9, pp. 6203–6214, Sep. 2016.
[2] H. G. Myung, J. Lim, and D. J. Goodman, “Single carrier FDMA for uplink wireless transmission,” IEEE Veh. Technol. Mag., vol. 1, no. 3, pp. 30–38, Sep. 2006.
[3] Y. Zhu and K. B. Letaief, “CFO estimation and compensation in SC-IFDMA systems,” IEEE Trans. Wireless Commun., vol. 9, no. 10, pp. 3200–3213, Oct. 2010.
[4] Y. Du, J. Chen, Y. Lei, and X. Hao, “Performance analysis of nonlinear spatial modulation multiple-input multiple-output systems,” Digit. Signal Process., vol. 115, Aug. 2021, Art. no. 103064.
[5] S. Zhang, S. Xu, G. Y. Li, and E. Ayanoglu, “First 20 years of green radios,” IEEE Trans. Green Commun. Netw., vol. 4, no. 1, pp. 1–15, Mar. 2020.
[6] S. C. Liew and Y. Shao, “New transceiver designs for interleaved frequency-division multiple access,” IEEE Trans. Wireless Commun., vol. 19, no. 12, pp. 7765–7778, Dec. 2020.
[7] Y. Shao and S. C. Liew, “Flexible subcarrier allocation for interleaved frequency division multiple access,” IEEE Trans. Wireless Commun., vol. 19, no. 11, pp. 7139–7152, Nov. 2020.
[8] C. Li, X. Wang, L. Cai, and Z. Gu, “Heterogeneous routing: A methodology for PPA tradeoff in network-on-chips,” in Proc. IEEE 3rd Int. Conf. Integ. Circuits Microsyst. (ICICM), Nov. 2018, pp. 357–361.
[9] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, “A generalized mixed-radix algorithm for memory-based FFT processors,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 57, no. 1, pp. 26–30, Jan. 2010.
[10] M. Garrido, “A survey on pipelined FFT hardware architectures,” J. Signal Process. Syst., vol. 94, no. 11, pp. 1345–1364, 2021.
[11] V. T. Ma, “A VLSI-oriented parallel FFT algorithm,” IEEE Trans. Signal Process., vol. 44, no. 2, pp. 445–448, Feb. 1996.
[12] R. M. Karp, Reducibility Among Combinatorial Problems. Berlin, Germany: Springer, 1972, pp. 85–103.
[13] J. K. Lenstra and A. H. G. R. Kan, “Complexity of scheduling under precedence constraints,” Oper. Res., vol. 26, no. 1, pp. 22–35, Feb. 1978.
[14] M. Bahtat, S. Belkouch, P. Elleaume, and P. L. Gall, “Instruction scheduling heuristic for an efficient FFT in VLIW processors with balanced resource usage,” EURASIP J. Adv. Signal Process., vol. 2016, no. 1, pp. 1–21, Dec. 2016.
[15] R. B. Ramakrishna. (1995). Iterative Modulo Scheduling. [Online]. Available: https://www.hpl.hp.com/techreports/94/HPL-94-115.pdf
[16] M. Frigo, “A fast Fourier transform compiler,” in Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement., May 1999, pp. 169–180.
[17] M. Frigo and S. G. Johnson, “FFTW: An adaptive software architecture for the FFT,” in Proc. Int. Conf. Acoust., Speech, Signal Process., vol. 3, May 1998, pp. 1381–1384.
[18] A. M. Blake, I. H. Witten, and M. J. Cree, “The fastest Fourier transform in the South,” IEEE Trans. Signal Process., vol. 61, no. 19, pp. 4707–4716, Oct. 2013.
[19] M. Püschel et al., “SPIRAL: Code generation for DSP transforms,” Proc. IEEE, vol. 93, no. 2, pp. 232–275, Feb. 2005.
[20] D. Skinner, “Pruning the decimation in-time FFT algorithm,” IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-24, no. 2, pp. 193–194, Apr. 1976.
[21] J. Markel, “FFT pruning,” IEEE Trans. Audio Electroacoustics, vol. AE-19, no. 4, pp. 305–311, Dec. 1971.
[22] T. Sreenivas and P. Rao, “FFT algorithm for both input and output pruning,” IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-27, no. 3, pp. 291–292, Jun. 1979.
[23] R. Airoldi, O. Anjum, F. Garzia, A. Wyglinski, and J. Nurmi, “Energy-efficient fast Fourier transforms for cognitive radio systems,” IEEE Micro, vol. 30, no. 6, pp. 66–76, Nov. 2010.
[24] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, “An energy-efficient partial FFT processor for the OFDMA communication system,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 57, no. 2, pp. 136–140, Feb. 2010.
[25] Y. Du, S. C. Liew, and Y. Shao, “Efficient FFT computation in IFDMA transceivers,” 2022, arXiv:2203.09932.
[26] M. Abramowitz, I. A. Stegun, and R. H. Romer, Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables. New York, NY, USA: Dover, 1988.
[27] R. Durrett, Probability: Theory and Examples. Cambridge, U.K.: Cambridge Univ. Press, 2019.
[28] Xilinx. (1995). Fast Fourier Transform V9.1 LogiCORE IP Product Guide. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_1/pg109-xfft.pdf

Yuyang Du (Graduate Student Member, IEEE) received the B.S. degree from the Department of Electronics, Peking University, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, SAR, China. From July 2019 to December 2020, he was a Communication Engineer with the 2012 Laboratory, Huawei Technologies, where he engaged in the development of 5G modem chips. His research interests include FPGA development, URLLC, and nonlinear power amplifiers.

Soung Chang Liew (Fellow, IEEE) received the S.B., S.M., E.E., and Ph.D. degrees from the Massachusetts Institute of Technology. From 1984 to 1988, he was with the MIT Laboratory for Information and Decision Systems, where he investigated fiber-optic communications networks. From March 1988 to July 1993, he was with Bellcore (now Telcordia), where he engaged in broadband network research. Since 1993, he has been a Professor with the Department of Information Engineering, The Chinese University of Hong Kong (CUHK), where he is currently the Co-Director of the Institute of Network Coding. His research interests include wireless networks, the Internet of Things, intelligent transport systems, machines learning, internet protocols, multimedia communications, and packet switch design. He was a recipient of the first Vice Chancellor Exemplary Teaching Award in 2000 and the Research Excellence Award in 2013 with CUHK. He is a fellow of IET, Hong Kong Institution of Engineers, and Hong Kong Academy of Engineering Sciences.

Yulin Shao (Member, IEEE) received the B.S. and M.S. degrees (Hons.) in communications and information engineering from Xidian University, China, in 2013 and 2016, respectively, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2020. He was a Research Assistant with the Institute of Network Coding from March 2015 to August 2016, a Visiting Scholar with the Research Laboratory of Electronics, Massachusetts Institute of Technology, from September 2018 to March 2019, and a Research Associate with the Department of Electrical and Electronic Engineering, Imperial College London, from January 2021 to November 2022. He is currently a Lecturer in engineering with the Department of Engineering, University of Exeter, and a Visiting Researcher with the Department of Electrical and Electronic Engineering, Imperial College London. His research interests include wireless communications and networking, signal processing, stochastic control, and machine learning. He is a Series Editor of IEEE Communications Magazine on the “Artificial Intelligence and Data Science for Communications” series.