IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 18, 2020, accepted July 25, 2020, date of publication August 5, 2020, date of current version August 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014507

Parallel Two-Way Relay Channel Estimation in Cloud-Based 5G Radio Access Networks

ALI A. EL-MOURSY[1], (Senior Member, IEEE), SAEED ABDALLAH[2], (Member, IEEE), MOHAMED SAAD[1], (Senior Member, IEEE), AND KHAWLA ALNAJJAR[2], (Member, IEEE)

[1]Department of Computer Engineering, University of Sharjah, Sharjah, United Arab Emirates
[2]Department of Electrical Engineering, University of Sharjah, Sharjah, United Arab Emirates

Corresponding author: Ali A. El-Moursy (aelmoursy@sharjah.ac.ae)

This research was supported by the University of Sharjah, in part by the Distributed and Networked Systems Research Group Operating Grant 150410, in part by the Targeted Project Grant 1602040336-P, and in part by the Competitive Project Grant 18020403109.

ABSTRACT The cloud radio access network (C-RAN) aims at migrating the traditional base station functionality to a cloud-based centralized base band unit (BBU) pool, thereby providing a promising paradigm for fifth-generation (5G) wireless systems. This results in a novel wireless architecture in which mobile users communicate with the cloud via distributed remote radio heads (RRHs) as relays, through two successive wireless links. The availability of accurate channel state information at the BBU pool is a critical requirement in such systems. This paper addresses the channel estimation problem at the terminals of a C-RAN using a two-way relay network (TWRN) model. To the best of our knowledge, for the first time we introduce a cloud-based channel estimation algorithm implementation leveraging cloud computing capabilities of virtualization and parallelization. By bridging the gap between cloud computing and wireless communication, this work achieves a step towards the open problem of network function virtualization (NFV) in C-RANs. Through a deep serial algorithm analysis, we are able to utilize data decomposition as well as exploratory decomposition in order to achieve significant speedup, which scales well with problem size. We assess the performance gains of our cloud-based algorithm via extensive simulation experiments, and report almost 5× reduction in computation time as compared to the state-of-the-art.

INDEX TERMS Distributed processing, parallelization, performance analysis, cloud radio access network, two-way relay network, semi-blind channel estimation, multithreading.

I. INTRODUCTION

The demand for high-speed data applications, e.g., high-quality wireless video streaming, social networking, machine-to-machine communication, Internet-of-Things and unforeseen applications that can reasonably be expected to materialize in the near future, necessitates a paradigm shift from the current (4G) system to the fifth-generation (5G) system [1]. One of the enabling technologies towards 5G is known as the cloud radio access network (C-RAN), which was first introduced by China Mobile in 2011 [2].

In C-RANs, the traditional base station functions are decoupled into two parts: the remote radio heads (RRHs) and the base band unit (BBU) pool implemented in a centralized cloud server [3]. RRHs perform radio functions, including frequency conversion, amplification, and analog/digital and digital/analog conversion, while the BBU pool acts as a digital unit implementing the base station functionality from baseband signal processing to packet processing [4]. The BBU pool is also known as Node C, a new communication entity defined to converge the existing ancestral base stations and manage all accessed RRHs [5]. The BBU pool is connected to the RRH via fronthaul links, where wireless fronthaul links represent a cost-effective and flexible alternative to optical fibers. The RRHs are serving the mobile users. Moving baseband processing and resource allocation from the base stations to the cloud reduces the network capital and operational capital expenditure. Also, a cloud-based BBU pool enables implementing network functionalities in software on general purpose computing platforms, a concept known as network function virtualization (NFV) [1]. Clearly, NFV adds flexibility and ease in deployment, maintenance and upgrade of network functions. Other advantages of C-RANs include lower energy consumption and efficient

The associate editor coordinating the review of this manuscript and approving it for publication was Cunhua Pan.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

large-scale cooperative processing [3]. A system model for a C-RAN is shown in Fig. 1. For further details on C-RANs, refer to [6] for a survey.

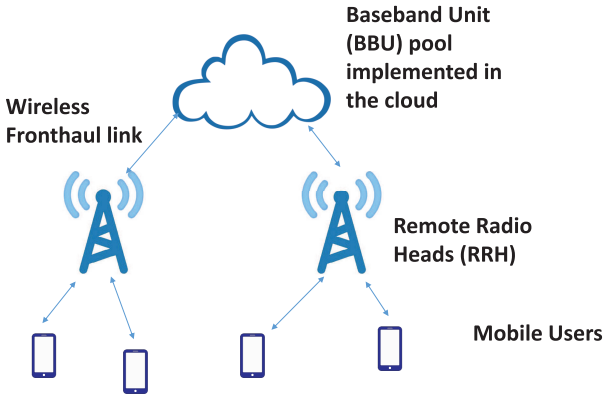

FIGURE 1. System Model of Cloud radio Access Network.

One of the most useful features of the cloud computing is the direct access to massive computing resources [7]. Accordingly, parallel implementations transparently may utilize the high-performance resources of the cloud [8]–[11]. This work seeks to address the implementation of network functionalities as cloud-based services while leveraging the following capabilities of cloud computing:

- The ability to establish, reconfigure and tear down virtual machines.
- The ability to configure virtual machines with multiple cores, which gives rise to the implementation of network functionalities with a high degree of parallelization.

As a step towards this goal, this paper addresses the problem of channel estimation in *both* cloud-to-RRH and RRH-to-user wireless channels. Since the cloud (BBU pool) and the mobile user can be regarded as terminals communicating in both (downlink and uplink) directions using the RRH as a relay, we use a two-way relay network [12] (TWRN) model. This provides a spectrally efficient solution for bidirectional communication between two terminals at twice the communication rate of the conventional one-way relay networks. Both the amplify-and-forward (AF) and the decode-and-forward (DF) relaying protocols have been considered in TWRNs [12]. AF TWRNs are particularly appealing because they require minimal processing at the relay (i.e., the RRH). However, they also require highly accurate channel state information at the terminals, both for coherent decoding, and for mitigating the effects of the inherent self-interference. The problem of channel estimation for AF TWRNs continues to attract considerable attention from researchers [13]–[16]. A semi-blind solution to this problem has been proposed in [17] that exploits both the transmitted pilots as well as the transmitted data to improve the estimation performance. To further enhance performance, as superimposed training at the relay is also utilized. Extensive simulations showed that the algorithm in [17] provides very high estimation accuracy, approaching the theoretical limit, though at a nontrivial computational cost.

In this paper, we propose a parallel implementation of the semi-blind two-way relay channel estimation algorithm proposed in [17]. Our goal is to leverage the increased computational resources available to the cloud, in order to provide a practical real-time implementation. Up to date, automated tools fail to exploit efficient parallelization opportunities or identify the true dependency among different tasks of a serial algorithm. Automated tools such as OpenMP [18] can only parallelize naive parallel loops. Despite the iterative nature of the semi-blind two-way relay channel estimation algorithm, a considerable level of loop carried dependency does exist. This makes the parallelization process challenging and non-trivial. Through a deep serial algorithm analysis, we are able to utilize data decomposition as well as exploratory decomposition in order to achieve significant speedup, which scales well with problem size. Our parallel algorithm allows an efficient utilization of the cloud computing resources. This moves us one step towards the cloud-based implementation of the communication algorithms, facilitating more centralized and efficient decision making and resource allocation [19]. In light of the above, the contribution of this work can be summarized as follows.

- To the best of our knowledge, for the first time we address the implementation of TWRN channel estimation as a cloud-based service. To this end, we leverage the cloud computing capabilities of virtual machines and multiple computing cores to devise an efficient parallel implementation of the algorithm.
- We investigate via extensive simulations the performance gains of the proposed parallel implementation of our algorithm. In fact, we are able to achieves up to $5\times$ reduction in computation time as compared to the state-of-the-art.

The remainder of this paper is organized as follows. Related work is discussed in Section II. Section III provides an overview of the channel estimation algorithm under consideration. Section IV describes the proposed parallel two-way relay channel estimation. The details of the implementation setup, as well as quantitative evaluation and numerical results are presented in Section V. Finally, Section VI concludes the paper.

II. RELATED WORK

In this section, we review the literature related to C-RAN research in general, as well as the literature on channel estimation efforts for C-RANs in particular.

C-RAN research work has been subject to considerable interest in the recent wireless communication literature. For example, the study in [4] reported on C-RAN fronthaul compression, medium access control, resource allocation and standardization efforts. The study [3] reported on signal compression and radio resource allocation. The study in [20] addressed RRH selection, beamforming and power control.

The study in [21] looked into resource scheduling and power control. The studies in [22] and [23] addressed routing problems in the fronthaul network, jointly with compression in [22], and jointly with precoder design in [23]. The study in [24] addressed the problem of interference mitigation in C-RANs by applying rate-splitting and common message decoding. The study in [25] addressed the problem of source coding/compression at the transmitter and channel decoding at the receiver of a C-RAN. The study in [26] addressed the problem of energy-efficient user association in C-RANs. Furthermore, the study in [27] considered the problem of downlink secure beamforming in C-RANs. None of the above studies, however, considered the details of implementing network functionalities in the cloud. The existence of a BBU pool implemented in the cloud was only a general assumption. In fact, it has been reported in [3] that NFV is still an open problem in C-RANs, and that the state-of-the-art is to implement network functions on dedicated and application-specific hardware.

The vital problem of channel estimation in C-RANs has been explored in a number of works [28]–[33]. The authors of [28] proposed a training-based sequential minimum-mean-squared error scheme with segmented training for C-RANs employing AF relaying. Training-based channel estimation for C-RANs employing full-duplex AF relaying was explored in [29], where a combination of least-squares and maximum-likelihood techniques were employed for joint channel estimation and self-interference suppression. In [30], channel estimation was considered in conjunction with cluster formation, where individual C-RAN clusters were formed by the RRHs and joint channel estimation and cluster formation was performed using semi-blind estimation with a likelihood function that combines both pilots and data symbols. The likelihood function was maximized using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm as in [17]. Semi-blind channel estimation for Uplink C-RANs was also studied in [31]. Focusing on estimation at the RRHs, maximum-likelihood estimation was adopted and the semi-blind likelihood function was again maximized iteratively using the BFGS algorithm. The authors of [32] considered C-RANs operating over millimeter-wave frequencies, and equipped with lens antenna arrays. A channel estimation algorithm was proposed that exploits the energy focusing property of lens antenna arrays. The authors of [33], considered joint channel estimation and user activity detection in C-RANs, proposing an algorithm based on variational Bayesian inference that exploits user activity sparsity and signal spatial sparsity.

To the best of our knowledge, however, none of the existing works on C-RAN channel estimation have developed parallel implementations that seek to exploit the computational capabilities of the cloud. Parallel processing can play an important role in providing real-time implementations of wireless communication algorithms by dividing a problem into multiple sub-problems that can be handled in different cores/processing units. Although several studies have exploited the parallelization capabilities of the graphics processing units (GPUs) to speed up signal detection/decoding algorithms for wireless receivers [34]–[36], only a small number of works have so far explored parallel implementations of channel estimation. In [37], a GPU implementation was used to accelerate least squares channel estimation and demodulation for large scale antenna systems. We note, however, that channel estimation algorithms considered in [37] do not apply to C-RANs and are not compatible with the semi-blind estimation strategy that we adopt in work. Semi-blind estimation is both more accurate and more computationally demanding as compared to [37], and is often implemented using iterative algorithms as done in [30] and [31].

III. TWO-WAY RELAY CHANNEL ESTIMATION

A. BACKGROUND

Our proposed parallelization targets the semi-blind channel estimation algorithm developed in [17]. This algorithm yields highly accurate channel estimates for TWRNs operating under frequency-selective fading conditions, where orthogonal frequency division multiplexing (OFDM) was adopted to combat the multipath phenomenon. The algorithm is based on the semi-blind estimation strategy, which utilizes both pilot and data samples to obtain the most accurate estimation. The transmitted data is incorporated into the log-likelihood function, and the resulting maximum likelihood estimator reduces to a nonlinear minimization problem, which was solved numerically using an iterative quasi-Newton method.

In this section, we provide a summary of the communication system model and the channel estimation algorithm in its conventional serial implementation. This is necessary in order to understand the computational requirements of the algorithm, the inherent data dependency, and the significance of the proposed parallelization techniques.

The communication system under consideration is an AF TWRN with two source nodes, $\mathcal{T}_1$ and $\mathcal{T}_2$, and a single relaying node $\mathcal{R}$, operating in frequency-selective channel conditions, shown in Fig. 2. In the context of cloud radio access networks, the terminal $\mathcal{T}_1$ represents the BBU pool, while the relay represents the RRH and the terminal $\mathcal{T}_2$ represents the user end (UE). To counter the effects of the

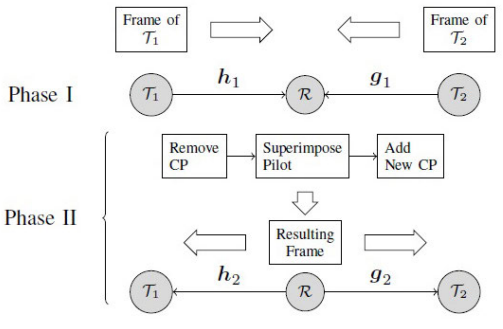

FIGURE 2. OFDM-based two-way relay network with superimposed training at the relay [17].

frequency selective fading, OFDM transmission with N subcarriers is used. Each round of data exchange between $\mathcal{T}_1$ and $\mathcal{T}_2$ consists of two phases. In the first phase, both terminals simultaneously transmit an OFDM frame to $\mathcal{R}$. In the second phase, an amplified version of the received frame is broadcasted by $\mathcal{R}$ to both terminals.

The more practical scenario of non-reciprocal channels is considered. The $L_1 \times 1$ vector $\boldsymbol{h}_1$ and the $L_3 \times 1$ vector $\boldsymbol{g}_1$ represent the baseband channel from terminals $\mathcal{T}_1$ and $\mathcal{T}_2$ to $\mathcal{R}$, respectively, while the $L_2 \times 1$ vector $\boldsymbol{h}_2$ represents the channel from $\mathcal{R}$ to $\mathcal{T}_1$. The purpose of the channel estimation algorithm is to acquire estimates of these three vectors. This task is critical from the communication perspective for the sake of successful recovery of transmitted data.

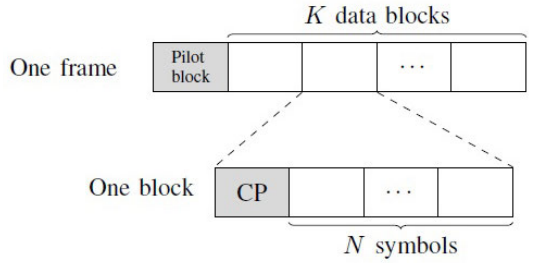

FIGURE 3. Structure of the OFDM frame transmitted by the terminals [17].

As shown in Fig. 3, each of the OFDM frames transmitted by the terminals consists of one pilot block and K data blocks. Moreover, a single OFDM block (whether pilot or data) consists of N time-domain symbols and a cyclic prefix (CP) of appropriate length, which is inserted at the beginning of the block to avoid inter-block interference (see Fig. 3).

B. THE ESTIMATION ALGORITHM

The unknown channel parameters to be estimated are collected into the vector $\boldsymbol{\theta} \triangleq [\boldsymbol{h}_1^T, \boldsymbol{h}_2^T, \boldsymbol{g}_1^T]^T$. The inputs to the algorithm consist of the vector $\boldsymbol{y}$, which represents the received pilot-carrying signal at $\mathcal{T}_1$ and the vectors $z_1, \ldots, z_K$, which represent the received data carrying signal. The exact form of these vectors can be found in [17].

The channel estimation algorithm was developed using the Gaussian maximum likelihood estimation strategy. Under this assumption, estimates of $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ and $\boldsymbol{g}_1$ are obtained through the following minimization

$$\{\hat{\boldsymbol{h}}_1^{(s)}, \hat{\boldsymbol{h}}_2^{(s)}, \hat{\boldsymbol{g}}_1^{(s)}\} = \underset{\boldsymbol{h}_1,\, \boldsymbol{h}_2,\, \boldsymbol{g}_1}{\arg\min}\ \mathcal{F}(\boldsymbol{\theta}), \tag{1}$$

where

$$\mathcal{F}(\boldsymbol{\theta}) \triangleq \log|\boldsymbol{C}| + (\boldsymbol{y}-\boldsymbol{\mu})^H \boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{\mu}) + K\log|\boldsymbol{Q}| + \sum_{k=1}^{K}(z_k-\boldsymbol{\mu}_k)^H \boldsymbol{Q}^{-1}(z_k-\boldsymbol{\mu}_k). \tag{2}$$

In the above objective function, $\boldsymbol{\mu}$ and $\boldsymbol{C}$ represent the mean vector and covariance matrix of $\boldsymbol{y}$, respectively, while $\boldsymbol{\mu}_k$ and $\boldsymbol{Q}$ represent the mean vectors and covariance matrix of the vectors $z_1, \ldots, z_K$.

The above objective function is nonconvex, and the solution was obtained iteratively using the BFGS method [38]. This is one of the most popular quasi-Newton methods and is known for its robustness and efficiency. Backtracking linesearch was employed to find the step size at each iteration [39]. BFGS with backtracking linesearch coupled with proper initialization results in performance that is very close to the theoretical bound, which indicates that convergence to the global minimum occurs most of the time. The algorithm is summarized in Algorithm 1 below.

Algorithm 1 Semi-Blind Estimation

Inputs: $\boldsymbol{y},\ z_1, \ldots, z_K$.
Initialize:
$\hat{\boldsymbol{\theta}}^{(0)} \triangleq [\Re\{\hat{\boldsymbol{h}}_1\}, \Re\{\hat{\boldsymbol{h}}_2\}, \Re\{\hat{\boldsymbol{g}}_1\}, \Im\{\hat{\boldsymbol{h}}_1\}, \Im\{\hat{\boldsymbol{h}}_2\}, \Im\{\hat{\boldsymbol{g}}_1\}]^T$.
$\boldsymbol{R}_0^{-1} = \beta \boldsymbol{I}_{2N}$.
Repeat until convergence:
1. Obtain the search direction:
$\boldsymbol{\Delta}_k = -\boldsymbol{R}_k^{-1} \nabla \mathcal{F}(\boldsymbol{\theta}^{(k)})$.
2. Find step size t using backtracking linesearch.
3. Update the estimate:
$\hat{\boldsymbol{\theta}}^{(k+1)} = \hat{\boldsymbol{\theta}}^{(k)} + t\boldsymbol{\Delta}_k$.
4. Set $\boldsymbol{u}_k = t\boldsymbol{\Delta}_k,\ \boldsymbol{v}_k = \nabla \mathcal{F}(\hat{\boldsymbol{\theta}}^{(k+1)}) - \nabla \mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)})$.
5. Update the inverse Hessian approximation:
$\boldsymbol{R}_{k+1}^{-1} = \boldsymbol{R}_k^{-1} + \frac{\left(\boldsymbol{u}_k^T \boldsymbol{v}_k + \boldsymbol{v}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{v}_k\right)\boldsymbol{u}_k \boldsymbol{u}_k^T}{(\boldsymbol{u}_k^T \boldsymbol{v}_k)^2} - \frac{\boldsymbol{R}_k^{-1}\boldsymbol{v}_k \boldsymbol{u}_k^T + \boldsymbol{u}_k \boldsymbol{v}_k^T \boldsymbol{R}_k^{-1}}{\boldsymbol{u}_k^T \boldsymbol{v}_k}$.

The downside, however, is that there is nontrivial computational burden until convergence is achieved. This is our motivation for proposing a parallel implementation that will significantly reduce of the algorithm convergence time.

Before delving into the proposed parallel implementation, it is important to identify the computational bottlenecks of the algorithm. The main tasks that affect the computational complexity of the BFGS procedure are: 1) the computation of the gradient, 2) the computation of the step size through linesearch and 3) the evaluation of the approximate inverse-Hessian update. The computation of the gradient and the step size are the computationally dominant task, both of which are performed once in each iteration of the algorithm. A substantial number of mathematical operations is involved in computing the gradient, while the linesearch is itself an iterative procedure that requires repeated evaluation of the objective function. It is worthwhile to explore these two steps in more detail, as they will be the focus of our parallelization in the next section.

C. COMPUTATION OF THE GRADIENT

The computation of the gradient involves evaluating the derivatives of the objective function in Eq. (2) with respect to all the channel parameters. Let θ_i be the ith element of the

vector $\boldsymbol{\theta}$ of channel parameters, where $i = 1, \ldots, L_1 + L_2 + L_3$. The gradient $\nabla\mathcal{F}(\boldsymbol{\theta})$ is an $(L_1+L_2+L_3)\times 1$ vector[1] whose ith element is the partial derivative $\frac{\partial\mathcal{F}(\boldsymbol{\theta})}{\partial\theta_i^*}$ which is given by

$$\begin{aligned}\frac{\partial\mathcal{F}(\boldsymbol{\theta})}{\partial\theta_i^*} &= \mathrm{tr}\left[\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i^*}\right] + K\,\mathrm{tr}\left[\boldsymbol{Q}^{-1}\frac{\partial\boldsymbol{Q}}{\partial\theta_i^*}\right] \\ &\quad - \frac{\partial\boldsymbol{\mu}}{\partial\theta_i^*}\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{\mu}) + (\boldsymbol{y}-\boldsymbol{\mu})^H\frac{\partial\boldsymbol{C}^{-1}}{\partial\theta_i^*}(\boldsymbol{y}-\boldsymbol{\mu}) \\ &\quad - \sum_{k=1}^{K}\frac{\partial\boldsymbol{\mu}_k^H}{\partial\theta_i^*}\boldsymbol{Q}^{-1}(\boldsymbol{z}_k-\boldsymbol{\mu}_k) \\ &\quad + \sum_{k=1}^{K}(\boldsymbol{z}_k-\boldsymbol{\mu}_k)^H\frac{\partial\boldsymbol{Q}^{-1}}{\partial\theta_i^*}(\boldsymbol{z}_k-\boldsymbol{\mu}_k) \qquad (3)\end{aligned}$$

where

$$\frac{\partial\boldsymbol{C}^{-1}}{\partial\theta_i^*} = -\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i^*}\boldsymbol{C}^{-1} \qquad (4)$$

and

$$\frac{\partial\boldsymbol{Q}^{-1}}{\partial\theta_i^*} = -\boldsymbol{Q}^{-1}\frac{\partial\boldsymbol{Q}}{\partial\theta_i^*}\boldsymbol{Q}^{-1}. \qquad (5)$$

Hence, to compute the gradient we need first to compute the terms $\boldsymbol{\mu}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, $\boldsymbol{C}^{-1}$, $\boldsymbol{Q}^{-1}$ as well as the differential terms $\frac{\partial\boldsymbol{C}}{\partial\theta_i^*}$, $\frac{\partial\boldsymbol{Q}}{\partial\theta_i^*}$, $\frac{\partial\boldsymbol{\mu}}{\partial\theta_i^*}$ and $\frac{\partial\boldsymbol{\mu}_k}{\partial\theta_i^*}$ for $i = 1, \ldots, L_1 + L_2 + L_3$ and $k = 1, \ldots, K$.

The computation of the terms $\boldsymbol{C}^{-1}$ and $\boldsymbol{Q}^{-1}$ can be done more efficiently by noticing that both $\boldsymbol{C}^{-1}$ and $\boldsymbol{Q}^{-1}$ are circulant matrices. An $N \times N$ circulant matrix $\boldsymbol{W}$ whose first column is $\boldsymbol{w}$ can be expressed as $\boldsymbol{W} = \boldsymbol{F}^H\mathrm{diag}(\tilde{\boldsymbol{w}})\boldsymbol{F}$, where $\tilde{\boldsymbol{w}}$ is the N-point Fast Fourier Transform (FFT) of $\boldsymbol{w}$ and $\boldsymbol{F}$ is the $N \times N$ normalized discrete Fourier transform (DFT) matrix whose (p, q)th entry is $1/\sqrt{N}e^{-\jmath 2\pi(p-1)(q-1)/N}$. Hence, the matrix inversion operation to obtain $\boldsymbol{C}^{-1}$ and $\boldsymbol{Q}^{-1}$ can be replaced by matrix multiplication and the FFT operation. In particular, it can be shown that

$$\boldsymbol{C}^{-1} = \boldsymbol{F}^H\mathrm{diag}(\tilde{\boldsymbol{v}_1})^{-1}\boldsymbol{F}, \qquad (6)$$

where $\boldsymbol{v}_1$ is the $N \times 1$ vector whose ith element is

$$v_{1i} = \sigma^2(A^2|\tilde{h}_{2i}|^2 + 1), \qquad (7)$$

and $\tilde{h}_{2i}$ is ith element of the vector $\tilde{\boldsymbol{h}}_2$, the N-point FFT of $\boldsymbol{h}_2$. Similarly,

$$\boldsymbol{Q}^{-1} = \boldsymbol{F}^H\mathrm{diag}(\tilde{\boldsymbol{v}_2})^{-1}\boldsymbol{F}. \qquad (8)$$

The detailed expressions for the differential terms $\frac{\partial\boldsymbol{C}}{\partial\theta_i^*}$, $\frac{\partial\boldsymbol{Q}}{\partial\theta_i^*}$, $\frac{\partial\boldsymbol{\mu}}{\partial\theta_i^*}$, $\frac{\partial\boldsymbol{\mu}_k}{\partial\theta_i^*}$ are not included due to the space limitations, as there are three different expressions for each term depending on the range of i. In particular, there is a set of differential expressions that work for $i = 1, \ldots, L_1$, another set of expressions for $i = L_1 + 1, \ldots, L_1 + L_2$ and finally another for $i = L_1 + L_2 + 1, \ldots, L_1 + L_2 + L_3$. The reason is that in the first case θ_i is an element of the channel vector $\boldsymbol{h}_1$, while in the second it is an element of $\boldsymbol{h}_2$ and finally in the third it is an element of $\boldsymbol{g}_1$. We note also that the terms $\boldsymbol{\mu}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, $\boldsymbol{C}^{-1}$, $\boldsymbol{Q}^{-1}$, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are needed for the computation of the differential terms, and thus have to be computed first.

For better understanding of the computational requirements of the gradient computation, we show in Table 1 the number of mathematical operations (complex multiplications and additions) involved in obtaining the terms $\boldsymbol{\mu}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, $\boldsymbol{C}^{-1}$, $\boldsymbol{Q}^{-1}$, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Table 2 shows the number of computations required to compute the ith element of the gradient (depending on value of i), assuming the availability of the terms listed in Table 1.

TABLE 1. Number of mathematical operations for obtaining the common terms of the gradient evaluation.

Term	Number of Operations
$\boldsymbol{v}_1$	$7N$
$\boldsymbol{v}_2$	$12N + 2$
$\boldsymbol{C}^{-1}$	$4N^3 - 2N^2$
$\boldsymbol{Q}^{-1}$	$4N^3 - 2N^2$
$\boldsymbol{\mu}$	$8N^2 - N$
$\tilde{\boldsymbol{y}}$	$2N^2$
$\boldsymbol{\mu}_k, k = 1, \ldots, K$	$4N^2K - NK$
$\tilde{\boldsymbol{z}}_k, k = 1, \ldots, K$	NK

D. COMPUTATION OF THE STEP SIZE

The step size t is calculated using backtracking linesearch. Let $\hat{\boldsymbol{\theta}}^{(k)}$ be the current estimate during the k-th iteration of the BFGS algorithm. This estimate is updated by taking a step of size t in the designated search direction, given by $\boldsymbol{\Delta}_k = -\boldsymbol{R}_k^{-1}\nabla\mathcal{L}(\hat{\boldsymbol{\theta}}^{(k)})$, where $\boldsymbol{R}_k^{-1}$ is the current approximation of the inverse Hessian matrix. Backtracking linesearch works as outlined in Algorithm 2 below.

Algorithm 2 Backtracking Linesearch

Inputs: $\boldsymbol{y}$, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K, \hat{\boldsymbol{\theta}}^{(k)}, \nabla\mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)})$.
Parameters: $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$.
Search Direction: $\boldsymbol{\Delta}_k$
Initial Step Size: $t := 1$
while $\mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)} + t\boldsymbol{\Delta}_k) > \mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)}) + \alpha t\boldsymbol{\Delta}_k^T\nabla\mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)})$
$t := \beta t$

It can be shown that the number of operations required for a single iteration of the While Loop in the linesearch is $N^2(6K + 14) + (4K + 25)N + 1$.

In the following section, we introduce an efficient parallel implementation of the compute-intensive semi-blind estimation algorithm, including the calculation of the gradient and step-size. Again, this parallelization is motivated by the 5G paradigm, where communication functionalities will be implemented in software in a centralized, cloud-based BBU pool.

[1]The gradient can equivalently be calculated in the real domain, in which case it will contain the same information but the dimension will be $2(L_1 + L_2 + L_3) \times 1$.

TABLE 2. Number of operations for obtaining the *i*th element of the gradient.

Gradient Element Range	Number of Operations
$i \in \{1, \ldots, L_1\}$	$N^2(4+2K) + N(2K^2 - K + 1) - K^2 + K - 1$
$i \in \{L_1 + 1, \ldots, L_1 + L_2\}$	$4N^3 + (6K+4)N^2 + (4K^2 - 2K + 30)N - K^2 + 2K + 7$
$i \in \{L_1 + L_2 + 1, \ldots, L_1 + L_2 + L_3\}$	$4N^3 + (2K+2)N^2 + (2K^2 - K + 14)N - K^2 + 3$

IV. PARALLEL TWO-WAY RELAY CHANNEL ESTIMATION

The development of an efficient parallel processing algorithm for a complex compute-intensive module involves many steps, and is affected by many factors. The dependency analysis of the major tasks of the serial algorithm, the exploration of the possible maximum level of concurrency, the decomposition methods (data, tasks, recursion etc.), the granularity of decomposition, the parallel system architecture (shared vs. distributed memory system), and the parallel programming paradigm (multithreading vs. message passing) are the key factors affecting the net speedup to be achieved. Deep data/task dependency analysis is the starting point to identify the concurrency level along with the relative computation load per computation task/sub-task. The analysis would start with the abstract coarse-grain level major tasks and would proceed deeper and deeper in identifying sub-tasks up till the per-instruction level of execution. The finer the decomposition would go, the smaller the computation tasks be and the higher the achieved concurrency level is. With higher level of concurrency and finer grain level of task decomposition, we would achieve more opportunities for parallel processing. However, the implementation overhead would be the cost paid. The larger the number of Processing Elements (PEs) used cooperatively to complete some computation task, the higher the overhead added. This can be explicit communication overhead in the distributed memory system due to the message passing communication among the PEs, or it can be implicit synchronization of multiple threads sharing the same memory subsystem. Hence, a top-down approach is the most efficient one to parallelize complex computation tasks. In subsection IV-A, we utilize this approach for the data/task dependency analysis. Then, we propose a parallel processing algorithm for each of the major subtasks of the two-way relay channel estimation.

A. DEPENDENCY ANALYSIS

The semi-blind estimation algorithm from a top computation level is demonstrated through the flow chart in Fig. 4. We only highlight the major components that are critical to the calculations. The flowchart shows the flow of data dependency. First, a group of arrays are initialized: (1) the received signal array denoted as *rx_sig*, (2) the self interference array denoted as *self_intrf*, and (3) the initial channel estimates denoted as *init_est*. It is worth noting that array *rx_sig* represents the received signal vectors $\boldsymbol{y}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_K$ (each of size $N \times 1$), array *self_intrf* represents the self-interference vectors $\boldsymbol{s}_{1k}, k = 1, \ldots, K$ (each of size $N \times 1$), and array *init_est* represents the initial estimates $\hat{\boldsymbol{h}}_1$, $\hat{\boldsymbol{h}}_2$ and $\hat{\boldsymbol{g}}_1$ (of sizes $L_1 \times 1$, $L_2 \times 1$ and $L_3 \times 1$, respectively). These initialized arrays are the inputs for the function *obtain_semi_blind_estimates* (*oSB*), which returns the semi-blind estimates after convergence. Then, the current gradient (*curr_grad*) of the objective function $\mathcal{F}(\hat{\boldsymbol{\theta}})$ at the location of *init_est* is calculated through the function *obtain_gradient* (*ogr*), which has as inputs the current estimates (*curr_est*, initially set as *init_est*), *rx_sig*, *self_intrf*. The search direction Δ_k is obtained based on the gradient and the approximate Hessian (*approx_Hess*). The purpose of the iterations is to search for the best semi-blind channel estimates. The threshold *grad_thresh* is used to identify convergence.

A loop-carried dependency in each iteration of recalculation of the asymptotically converging semi-blind estimate prevents parallelization across the loop iterations. In each iteration, two dependent steps are performed. In the

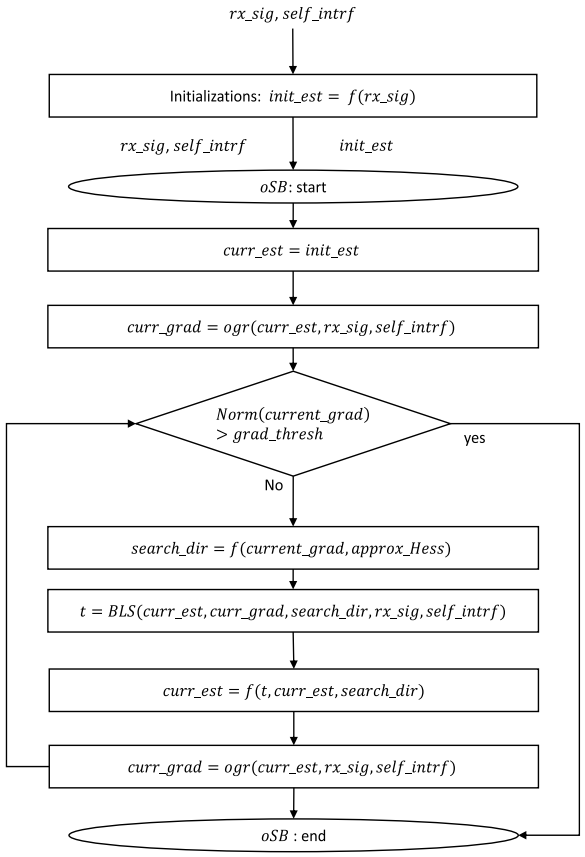

FIGURE 4. Serial semi-blind estimation algorithm.

first step, the appropriate *step_size t* is obtained using the function *backtracking_linesearch* (*BLS*), and used along with the search direction (*search_dir*) Δ_k to update *curr_est*. Then the updated *curr_est* is used to calculate a new *curr_grad* via the *ogr* function, and the search direction is updated. The true dependency shown indicates lack of parallelization across those two tasks.

B. PARALLEL GRADIENT

To further elaborate on the breakdown of the tasks of *ogr* we show its flowchart in Fig. 5. We can discriminate between two main computation tasks in that function. In Task A: $\hat{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{y}} \triangleq \boldsymbol{F}^H(\boldsymbol{y} - \boldsymbol{\mu})$, $\tilde{\boldsymbol{z}}_k \triangleq \boldsymbol{F}^H(\boldsymbol{z}_k - \boldsymbol{\mu}_k)$, $\boldsymbol{v}_1$, $\boldsymbol{C}^{-1}$, $\boldsymbol{v}_2$, $\boldsymbol{Q}^{-1}$ are calculated given specific arrays for *rx_sig* and *self _intrf* and the estimate *input_est* provided by the caller function as described in the previous flowchart in Fig. 4. Hence, Task A can be even decomposed to two subtasks, Task A(p1) and Task A(p2), as shown in Fig. 6. Each of these calculations can be performed independently, however, we should check their relative time significance to gain any benefit from that parallelism. Finer grain parallelization is possible for Task A with its two subtasks. However, the computational complexity analysis performed in Section III does not indicate high computation cost (7% relative to the total execution time) to motivate further decomposition. We will analyze the relative weights of the computations in the time analysis subsection.

The second part of *ogr* (Task B) is truly dependent on Task A. However, it consists of an intensive *for* loop to calculate the gradient of the objective function in Eq. (2) as expressed in Section III-C. Each element of the arrays $\boldsymbol{h}_1, \boldsymbol{h}_2$, and $\boldsymbol{g}_1$ is calculated independently, i.e., $grad(i)$ for $i = 1, \ldots, L_1 + L_2 + L_3$. Hence, this is a candidate of data parallelization where each iteration could be executed independently on a Processing Element (PE). The maximum concurrency level for that loop is $L_{tot} = L_1 + L_2 + L_3$ (e.g., $L_{tot} = 30$ if each vector is of size 10). Fig. 6 demonstrates how the tasks are decomposed among different PEs based on the dependency analysis performed.

The dependency analysis performed for the serial and the parallel tasks *ogr* can be evaluated theoretically through a simple timing analysis. However, the actual parallelization speedup can only be assessed quantitatively through parallel implementation. Actual execution on parallel system accounts for the parallelization overhead (idle and communication/synchronization time) as well as the relative execution time for each concurrent subtask. To simplify the analysis, we can define in the abstract level the serial time of *ogr* as t_{s_ogr} reflecting the flow chart in Fig. 5.

Eq. (9), captures most timing components of the serial time of *ogr* where t_{ogr_init} is the initialization timing for

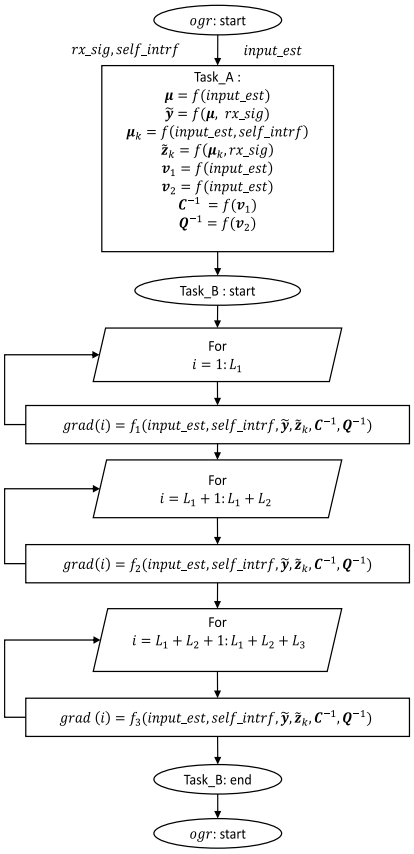

FIGURE 5. Serial evaluation of the gradient.

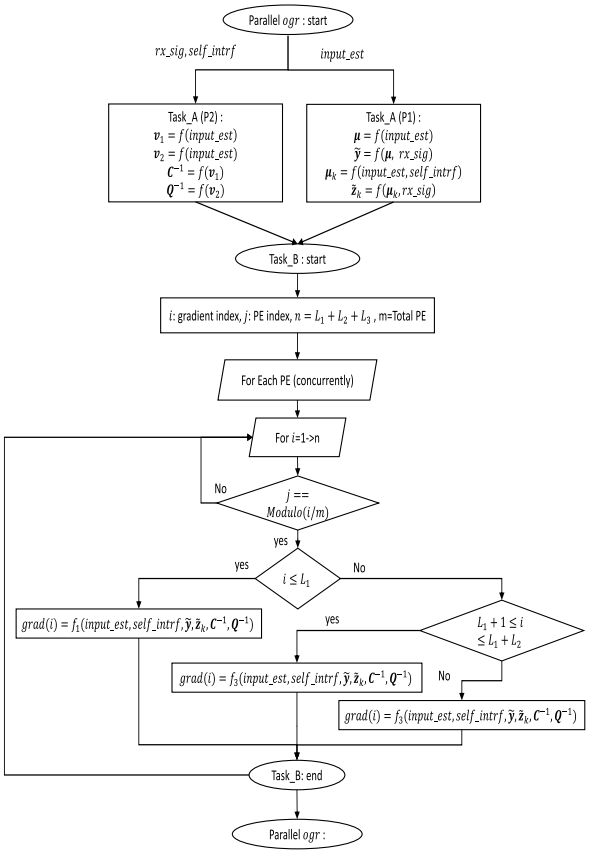

FIGURE 6. Parallel evaluation of the gradient.

any computation task outside the major tasks of *ogr*, and $t_{s\_ogr\_task\_A}$ and $t_{s\_ogr\_task\_B}$ are the time of *TaskA* and *TaskB* respectively. Eq. (10) and (11) show the further decomposition of the serial *TaskA* and *TaskB* respectively. Assuming that the parallel system has number of PEs $N_{PE} \geq L_{tot}$, the time for the parallel execution of the proposed parallel *ogr* in Fig. 6 can be expressed by Eq. (12) and (13) for the parallel *TaskA* and *TaskB* respectively. Hence, the total execution time for the parallel *ogr* is given by Eq. (14) where t_{ogr_oh} is the overhead time for parallelization due to communication/synchronization. Accordingly, the speed up of *ogr* parallelization is represented in Eq. (15).

$$t_{s\_ogr} = t_{ogr\_init} + t_{s\_ogr\_task\_A} + t_{s\_ogr\_task\_B} \quad (9)$$

$$t_{s\_ogr\_task\_A} = t_{ogr\_p1} + t_{ogr\_p2} \quad (10)$$

$$t_{s\_ogr\_task\_B} = (L_1 * t_{ogr\_grad\_h1}) + (L_2 * t_{ogr\_grad\_h2}) + (L_3 * t_{ogr\_grad\_g1}) \quad (11)$$

$$t_{p\_ogr\_task\_A} = \max(t_{ogr\_p1}, t_{ogr\_p2}) \quad (12)$$

$$t_{p\_ogr\_task\_B} = \max(t_{ogr\_grad\_h1}, t_{ogr\_grad\_h2}, t_{ogr\_grad\_g1}) \quad (13)$$

$$t_{p\_ogr} = t_{ogr\_init} + t_{ogr\_oh} + \max(t_{ogr\_p1}, t_{ogr\_p2}) + \max(t_{ogr\_grad\_h1}, t_{ogr\_grad\_h2}, t_{ogr\_grad\_g1}) \quad (14)$$

$$Speedup_{ogr} = \frac{t_{ogr\_init} + t_{s\_ogr\_task\_A} + t_{s\_ogr\_task\_B}}{t_{p\_ogr}} \quad (15)$$

C. PARALLEL BACKTRACKING LINE SEARCH

The next task to be studied in detail is the backtracking linesearch (*BLS*) function to calculate the appropriate step size in order to update *curr_est*. The *BLS* flowchart is shown in Fig. 7. Again this process contains an intensive iterative routine searching for a convergence for the step size. First, the initial value of the objective function $\mathcal{F}(\hat{\boldsymbol{\theta}}^{(k)})$ (denoted as *init_val*) is calculated based on the provided *rx_sig*, *self_intrf* and *curr_est* by calling the function *eval_obj*. Then the step size t is initialized to 1 and another value of the objective function, denoted *curr_val* is evaluated at the location $curr\_est + t * search\_dir$. The value *curr_val* is compared to a threshold value that depends on t, the search direction *search_dir* and the current gradient *curr_grad*, and this procedure is repeated until the threshold is satisfied, each time updating the step size through the update $t := \beta t$. Calculations in *eval_obj* could be even broken-down to a set of equations. However we want to see the performance significance of this process before we go into that fine-grain decomposition of tasks. While the iterations of the *eval_obj* function have the same carried loop dependency feature of the higher level iterative process of gradient calculation, the search process for the best step size can be performed independently for each scaled updated estimate $curr\_est + \beta^n * search\_dir$, where n is the iteration index. However, if PE is available to generate a separate *eval_obj* value from each scaled estimate, a comparison is needed to pick the lowest n that satisfies the threshold condition. Then, we have to ignore the rest of the calculated estimates by the other processing elements. Still each iteration can be independently performed on a separate processing element. However, the net speedup depends on recursive dependency and is not a direct scaling based on the number of used processing elements [40], [41]. Computations are repeated on each processing element with different value for the size. Then a comparison is performed for all the log-likelihood values from different processing elements. All are flushed except the best which will be the starting point for the next parallel iteration as shown in Fig. 8. Net speedup is highly dependent on the efficiency of the synchronization among the PEs. Since, PEs have to communicate after every parallel iteration to realize whether the target has been reached or not. Suppose that for the serial execution, we have to perform number of iterations $N_{iter\_eval\_obj}$ till the convergence is achieved. Hence, the serial time for *Backtrack LinearSearch* (BLS) t_{s_BLS} can be formulated as in Eq. (16) where t_{eval_obj} is the time for one backtracking linersearch iteration. The number of iterations if N_{PE} cooperatively participate in the search is $\lceil \frac{N_{iter\_eval\_obj}}{N_{PE}} \rceil$. Hence, *BLS* parallel time would be

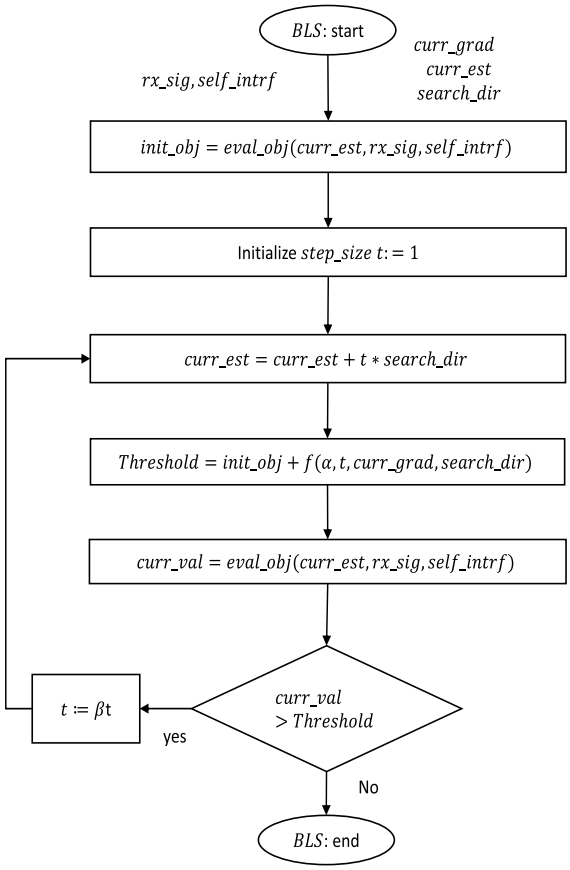

FIGURE 7. Serial backtracking linesearch.

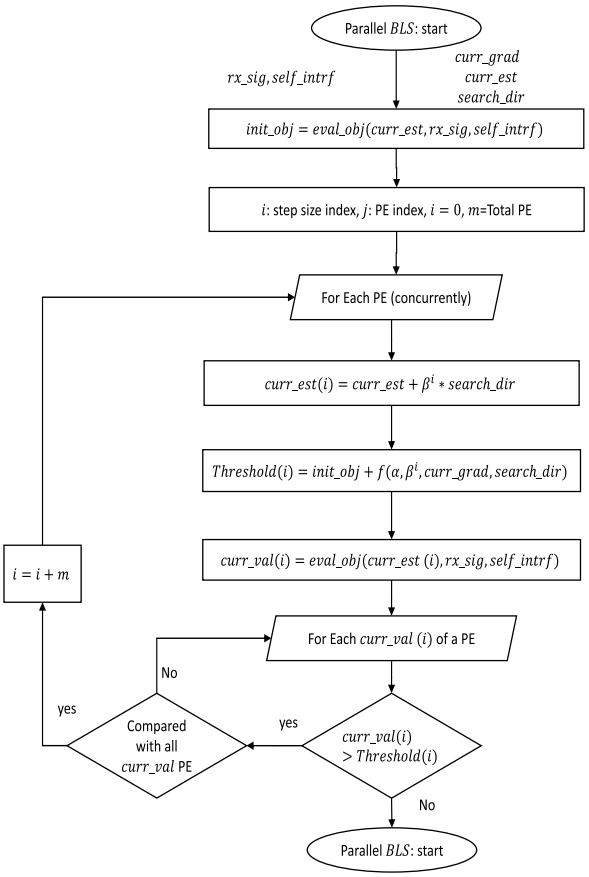

FIGURE 8. Parallel backtracking linesearch.

given by Eq. (17) where $t_{eval\_obj\_oh}$ is the overhead time to compare the values across the PEs and decide whether to go for one more iteration or not. Accordingly, the Speedup can be formulated by Eq. (18).

$$t_{s\_BLS} = N_{iter\_eval\_obj} * t_{eval\_obj} \tag{16}$$

$$t_{p\_BLS} = \lceil \frac{N_{iter\_eval\_obj}}{N_{PE}} \rceil * (t_{eval\_obj\_oh} + t_{eval\_obj}) \tag{17}$$

$$Speedup_{BLS} = \frac{N_{iter\_eval\_obj} * t_{eval\_obj}}{\lceil \frac{N_{iter\_eval\_obj}}{N_{PE}} \rceil * (t_{eval\_obj\_oh} + t_{eval\_obj})} \tag{18}$$

V. QUANTITATIVE EVALUATION

The objective of this section is to provide a numerical assessment of the performance of the parallel implementation of the Semi-blind Estimation Algorithm, as compared to its regular serial implementation. The main focus is to assess the execution speedup achieved by the parallel implementation.

Since the parallel algorithm is neither data intensive nor relies on a massive parallelization, nowadays mid-range servers with high a core count can be utilized for the parallel execution. Hence, we use a multithreaded implementation (shared memory and address space parallel paradigm) [40], [41], and a server with up to 32 cores is employed. Our server is Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, L1 cache:32K, L1i cache:32K, L2 cache: 256K, L3 cache:32MB with 64GB of RAM. First we converted the semi-blind estimation algorithm originally implemented on MATLAB to C++ using Armadillo version 7.8 linear algebra library (matrix maths) for the C++ language [42]. Then we added the multithreaded implementation. The C++ (g++ 5.4) for Ubuntu is used to compile the code with -04 compiler optimization level.

A. IDEAL SPEEDUP ANALYSIS

The table in Fig. 9 shows both the frequency of execution as well as the relative execution time to the total execution time for each of the major tasks/functions identified in the previous section. We used gprof performance profiling tool [43] and the dot-graph generation to assist in our analysis. The analysis performed for the semi-blind estimate algorithm in Fig. 9 is for the setup $N = 64$ and $L_1 = L_2 = L_3 = 10$. The function *oSB* is an iterative process searching for the best semi-blind estimate. It consists of the two major tasks (*ogr* and *BLS*) executed serially and dependently. The number of iterations performed is not fixed and is dependent on the signal-to-noise ratio (SNR). We ran experiments for seven different SNR values and averaged our performance results. For the experiments performed we experience around 44 to 88 iterations for SNR ranges from 0 dB to 30 dB. Although our parallel system has only 32 cores, peak performance is reached way below the maximum utilization of cores as will be shown in the results. On average, 65 iterations are needed for *oSB* to converge to the final estimate. Across those iterations, no parallelization can be performed due to the loop-carried-dependency discussed in the previous section. However, the relative time

Task flow	Average frequency	time (%)	Theoretical Speedup
for each SNR	1	100	6.6
Initializations	1	7	1.0
oSB	1	93	11.4
ogr	1	1	9.5
while convergence not achieved	65 (43->87)	92	11.5
BLS	65 (43->87)	33	18.0
ogr	65 (43->87)	59	9.5
end while			
end *oSB*			
end for each SNR			
ogr	66 (44->88)	60	9.5
Task_A	66 (44->88)	7	2.0
Task_B	66 (44->88)	53	18.9
for $i = 1{:}L_1$ calculate *grad(i)* End	660	9	10.0
for $i = L_1{+}1{:}\ L_1{+}L_2$ calculate *grad(i)* End	660	28	10.0
for $i = L_1{+}L_2{+}1{:}\ L_1{+}L_2{+}L_3$ calculate *grad(i)* End	660	16	10.0
end Task_B			
end *ogr*			
BLS	65	33	18.0
eval_obj	65	3	
while convergence not achieved *eval_obj* end while	1190 (600->1900)	30	
end *BLS*			

FIGURE 9. Relative execution time analysis for the algorithm.

is divided to about 33% for *BLS* and about 60% for *ogr*. Hence, each of those functions should further be studied to explore parallelization opportunities. For *ogr*, Task A takes 7% while Task B takes 53%. This directly suggests focusing on Task B which is a good candidate for parallelization from the dependency analysis side as discussed in the previous subsection. Task B is an iterative process that can be disjointedly performed on each element of gradient array ($\boldsymbol{h}_1, \boldsymbol{h}_2$, and $\boldsymbol{g}_1$) separately in a data decomposition fashion [40], [41].

We further profiled the three components of the *ogr* and we can see a significant variation in the weight of calculations related to each of them ($\boldsymbol{h}_1$ takes 9% and $\boldsymbol{h}_2$ takes 28%). In the experiments performed each of those vectors is of size 10×1. This means we can go as small as 2.8% per compute iteration ($\boldsymbol{h}_2$ is the most time critical with ten iterations that take for total 28%). Further decomposition is possible for the gradient iteration, however, it is directly dependent on the capabilities of the underlying hardware and its concurrency efficiency.

Moving to *BLS*, which searches for the appropriate step size along the designated search direction, the relative execution time is almost half of *ogr*. However, it is a more iterative function. With each *ogr* iteration, *BLS* iterates for 18 iterations on average. However, *BLS* iterations are harder to parallelize since with 33% weight of the *BLS* function, each iteration computation weight is around 1.5% (32/18).

The last column of the Table in Fig. 9 predicts the theoretical speedup if our parallel proposal is implemented on an ideal system with no overheads (see Eqs. 15 and 18). *ogr* and *BLS* can achieve around 9.5× and 18× speedup respectively. However, due to the relative weights of the different tasks and the remaining serial (non-parallelized) tasks, the ideal parallel semi-blind estimation algorithm may achieve at maximum 6.6× speedup for $N = 64$, $L_1 = L_2 = L_3 = 10$. This gives us a good estimate of the upper limit of what we can achieve and how close we can get to the best speedup limit.

B. IMPLEMENTATION SETUP

We have used a master-slave model for the parallel implementation in which the master process (main thread) forks into a number of threads that matches the maximum level of concurrency identified to be $\max(N_{iter\_eval\_obj}, L_{tot})$. These threads will be waiting on some conditional variables for triggering to perform some computation tasks assigned by the master. The master will perform all serial parts of the estimation process. If a function is to be parallelized the master will signal the slave threads to perform the parallel tasks. Each thread is signaled based on the thread index. Then the threads will signal the master back when they finish processing. Barriers are used to trigger threads to start processing as well as the master to regain the control of the flow of code execution. Since a shared memory paradigm is utilized, no explicit data transfer is needed among the processing threads or processing elements (PEs). However, a synchronization is needed to move from one processing task to the next.

As for the algorithm variables, unless mentioned otherwise, the number of subcarriers (FFT size) is set to $N = 64$ and the channel length (size of vectors $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ and $\boldsymbol{g}_1$) is $L_1 = L_2 = L_3 = 10$, hence $L_{tot} = 30$. However, we will also consider in some experiments $N = 128$ and also channel lengths of 5 and 15 to explore the scalability of the parallelization gains with the size of the problem.

1) PERFORMANCE ANALYSIS FOR PARALLELIZATION OF THE GRADIENT COMPUTATION

We begin by discussing the performance of the *ogr* function. Fig. 10 shows different components of that function for both serial and parallel implementations. Obviously no time variation is noticed for the serial runs of *ogr*. As our performance analysis shows, the major computation time of the *ogr* function is spent in Task B. Task A is not much scalable since the level of parallelism is at most 2 as discussed in the previous section. Hence, we merge Task A to the *ogr* serial Tasks. Before we discuss the parallelization of the major task (Task B), we note that the major overhead of parallelism is the synchronization needed among the running Processing Elements (threads in a shared address space parallel paradigm). Hence, we run a hypothetical experiment (serial + synch) *ogr* shown in Fig.10. In which we keep the master node doing the computations while signaling the threads to start computations without real calculations performed per thread. This experiment will allow us to see the amount of overhead since the computation time is still carried out by the master node alone. However, the synchronization time is still measured across the threads. Without a real work done by the threads, the synchronization time increases gradually especially when the number of threads goes beyond ten threads. This indicates a corresponding slowdown to the *ogr* time with more threads running the code.

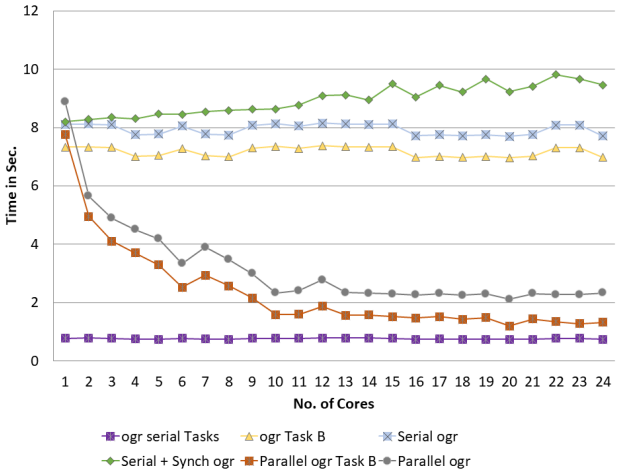

FIGURE 10. Performance of serial and parallel gradient computation.

Accordingly, the time curve of parallelizing the *ogr* Task B scales down significantly with more and more cores (threads) cooperatively performing the *ogr* Task B computation until it reaches the point of ten cores. Time for *ogr* Task B still scales down with more cores, however, with a much slower

slope. A tradeoff between the benefit of time scaling down for the cooperative computation among the cores and the thread synchronization is the factor that determines whether we get a speedup or slowdown through parallelization. However, the case is different for the overall *ogr* time considering the *ogr* serial Tasks. Any benefit of parallelization vanishes beyond the ten cores and steady timing is produced.

The speedup of the *ogr* function is shown in the Fig. 11. Despite the synchronization overhead, parallel *ogr* Task B can keep achieving speedup to about 6× of serial *ogr* Task B, however relative to parallel *ogr* only around 4× is achieved for ten cores. For the total time of the whole *oBS*, only 2.5× speedup is achieved for the same number of cores.

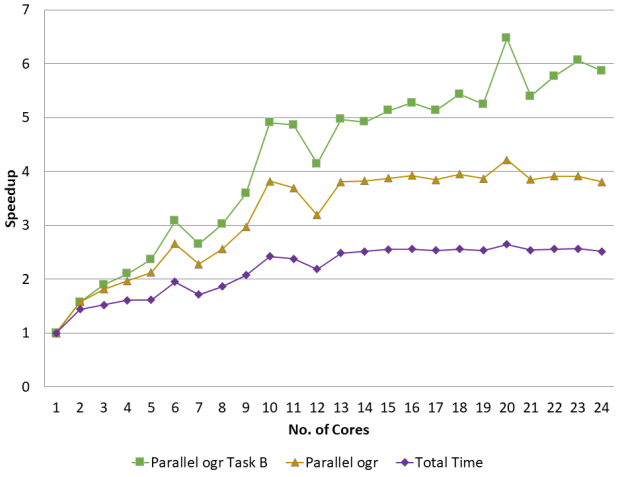

FIGURE 11. Overall speedup in *ogr* versus the number of cores.

2) PERFORMANCE ANALYSIS FOR THE PARALLELIZATION OF BACKTRACKING LINESEARCH

BLS function is also interesting to study. For this function, no data decomposition is possible due to the loop-carried dependency and the high frequency of iterations performed. Hence, there is no room to breakdown the execution time to independent tasks. Fig, 12 shows two lines for the serial execution. One for the *BLS* time and the other considering a similar hypothetical case to *ogr* in which threads are signaled to wake up just to account for the synchronization time. Although no data decomposition could be performed, *exploratory decomposition* is utilized for this function, in which each thread is doing the search for a different step in each iteration. The master waits all cores to finish then compares their results to select the best before going to the next iteration. If the target is reached, no new iteration is started. If not, all threads will go to the next iteration with each of them performing its search with a different step.

It is worth noting that the number of iterations for the convergence is not fixed. Hence, some of the slaves' work is wasted, which means needless overhead to compare their computations. The synchronization time for this function does not increase linearly with the number of cores (threads)

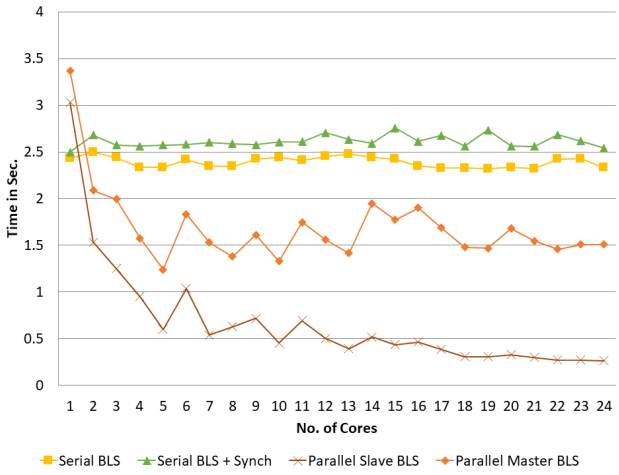

FIGURE 12. Performance of serial and parallel backtracking linesearch.

participating in the search process. This is due to two contradicting factors. Increasing the number of synchronized cores increases the synchronization time, however, with more cores used in the search process fewer iterations are needed to converge. The net synchronization time needed is almost constant with a slight increase over large number of cores. The computation time for the slaves decays asymptotically with increasing the number of cores used. However, this does not necessarily induce a continuous reduction of the master node time since the master should compare the results from a larger number of slaves. We can observe that again for ten cores (slaves) the best time reduction is achieved. Beyond this point, the overhead of synchronization and comparing the results will waste any benefit from reducing the number of iterations.

The results in Fig. 13 show another dimension for the *BLS* parallelization. The first line chart shows how the number of iterations needed for the linesearch to converge decays as

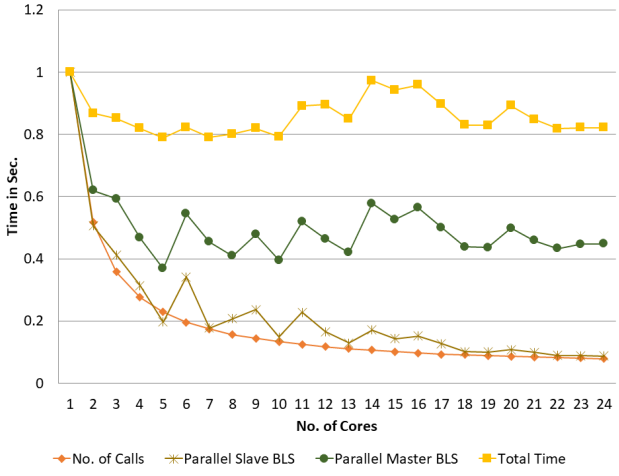

FIGURE 13. Relative scaling to one core of backtracking linesearch.

more cores cooperate in the search process. A steep reduction in the number of iterations is noticed with few slaves (up to ten). Beyond that a shallow reduction in iterations is achieved. This directly reflects on the relative reduction in time for the slaves in the parallel slave *BLS* curve. However, the master node time is not decaying as good as the slave time. Only around five slaves point we can get the best time reduction for the master. The same is noticed for the total time. However, the total time gets much less reduction due to the relative weight of *BLS* which is around the 32% as explained in the previous section. In the best result for the total time for five cores reduces the time by 20% which indicates a speedup of 1.25×.

3) PERFORMANCE ANALYSIS FOR OVERALL CHANNEL ESTIMATION PARALLELIZATION

Fig. 14 shows the time for serial, serial + Synch (for both *ogr* and *BLS*), Parallel *ogr* alone, Parallel *BLS* alone and Parallel *ogr_BLS* (for both of the functions together). Fig. 15 shows the speedup achieved for parallelizing *ogr* alone (Parallel *ogr*), *BLS* alone (Parallel *BLS*) and both *ogr* and *BLS* (Parallel *ogr_BLS*). The timing chart shows higher rate of increase in the synchronization time due to the more frequent barriers needed among the threads after each iteration of *ogr* as well as *BLS*. The total time reduction for parallelizing each of *ogr* and *BLS* alone is discussed in the previous charts, however, this chart adds a very interesting result of parallelizing both functions together and comparing to the parallelized for each individual function. While *BLS* function scales the total time down till five cores, *ogr* function scales the total time down till ten cores. Applying both parallelization techniques in Parallel *ogr_BLS* can achieve a time reduction up to eight cores. Keeping the master and slaves busy with calculations makes the scaling more smooth and gradual which avoids long idle times for either the master or the slaves. The maximum speedup of 3.6× is achieved with eight cores as shown

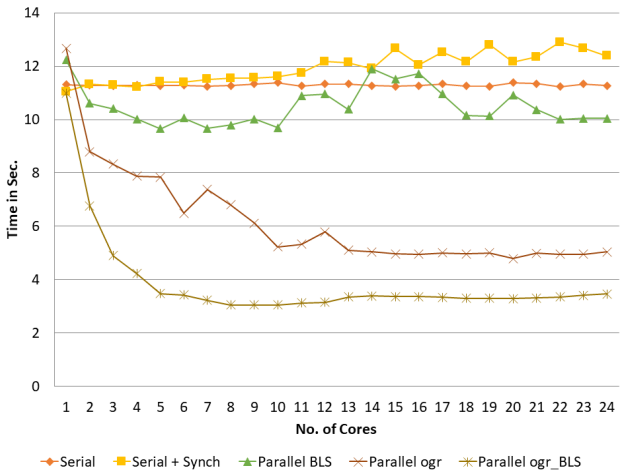

FIGURE 14. Time for serial, serial + Synch, Parallel *ogr*, Parallel *BLS* and Parallel *ogr_BLS*.

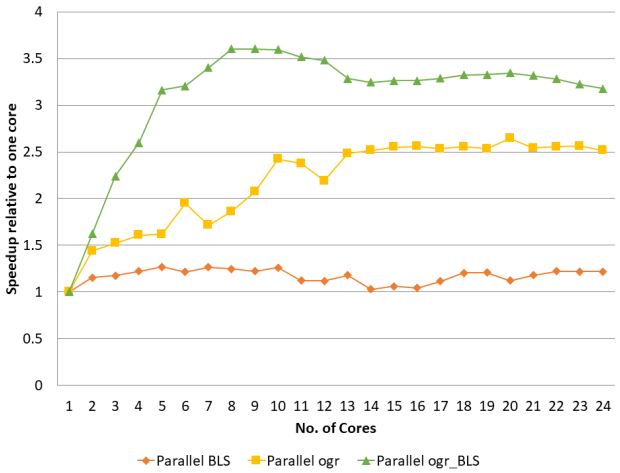

FIGURE 15. Time for serial, serial + Synch, Parallel *ogr*, Parallel *BLS* and Parallel *ogr_BLS*.

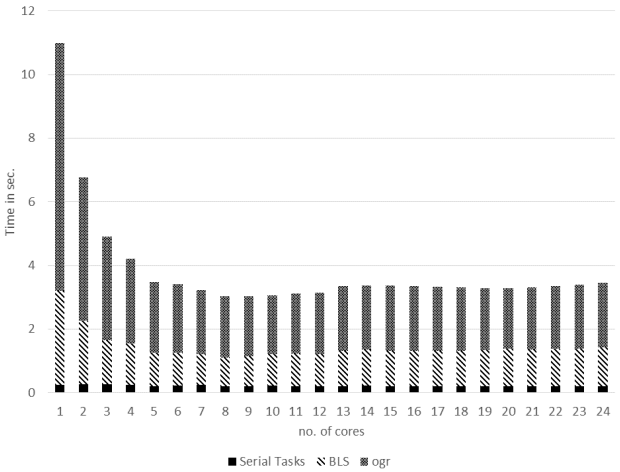

FIGURE 16. Breakdown for the absolute time and speedup relative to one core.

in Fig. 15. It is worth noting that our theoretical speed up is estimated to be 6.6× as shown in section. Hence, almost 60% of the theoretical/ideal speed up is waisted due to the overhead.

Fig. 16 and Fig. 17 show the breakdown for both the absolute time and speedup relative to one core respectively for *ogr* function, *BLS* function and overall channel estimation function. The parallelization of the two functions together cooperatively achieves speedup of 4.25× for ten cores, 3.25× for eight cores and 3.6× for eight cores for the functions *ogr*, *BLS*, and overall channel estimation respectively.

4) SCALABILITY AND CLOUD VIRTUALIZATION ANALYSIS

The real proof of developing an efficient parallel algorithm is the ability of that algorithm to scale particularly with respect to the problem size. In order to evaluate the scalability of our proposed parallelization in the dataset dimension, we run experiments for three more dataset for $\{N = 64, L_1 = L_2 = L_3 = 5\}$, $\{N = 128, L_1 = L_2 = L_3 = 10\}$, and $\{N = 128, L_1 = L_2 = L_3 = 15\}$ besides

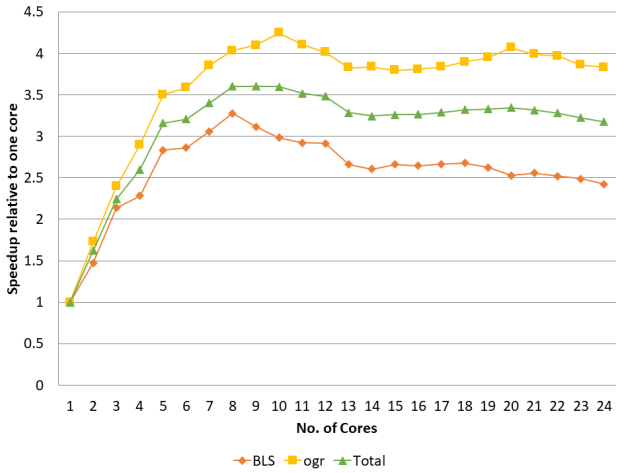

FIGURE 17. Breakdown for the speedup relative to one core.

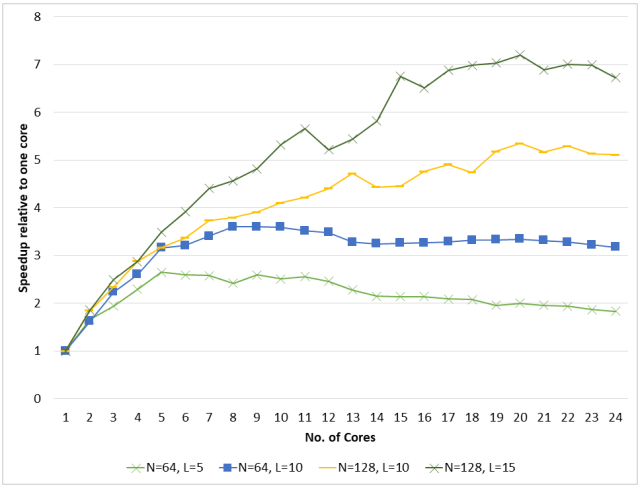

FIGURE 18. Scalability of Parallel semi-blind estimation with respect to dataset size.

the dataset for $\{N = 64, L_1 = L_3 = L_3 = 10\}$ that is used in the full performance analysis. The channel length parameters (L_1, L_2, L_3) directly affect the parallelization of *ogr* function. On the other hand, the FFT size parameter N affects the iterations for the *BLS* function to converge. Fig. 18 shows that $\{N = 64, L = 5\}$, $\{N = 64, L = 10\}$, $\{N = 128, L = 10\}$, and $\{N = 128, L = 15\}$ achieve the maximum speedup with five, eight, twenty and twenty cores respectively. This significant increase in the number of cores achieving speedup shows the robustness of the approach taken to parallelize the semi-blind estimation algorithm. The maximum speedup goes from 2.5× and 3.6× for $\{N = 64, L = 5\}$ and $\{N = 64, L = 10\}$ respectively up to 5.4× and 7.2× for $\{N = 128, L = 10\}$, and $\{N = 128, L = 15\}$ respectively. This projects to a significantly improved algorithm performance for even higher problem sizes.

All discussed results was for the relative performance (execution time) to the serial implementation not accounting for the overhead caused by the Cloud implementation. Although the extra flexible computing resources the Cloud offers for C-RAN, nothing comes for free. The layer of virtualization that allow the flexible recourse provisioning, however adds overhead on the execution time which is a direct price paid for Cloud utilization. To account for that overhead, Fig. 19 shows how much slowdown is caused comparing bare-metal (Physical Machine (PM)) implementation to the Virtual Machine (VM). Virtualization causes a slowdown between 2× to 1.5× for the different datasets. It is worth noting that some studies, e.g., [44] and [45], reported a slowdown due to virtualization overhead of up to 5×. These studies, however, focus on pure network transactions applications (e.g., iPerf sending and receiving for TCP/IP, and Apache2 many-client HTTP download). In contrast, our work focuses on a pure computer-intensive application (i.e., channel estimation). This explains the difference in slowdown due to virtualization overhead. We can observe less overhead for the larger dataset due to the higher computation time for the large dataset compared to the small one. Finally,

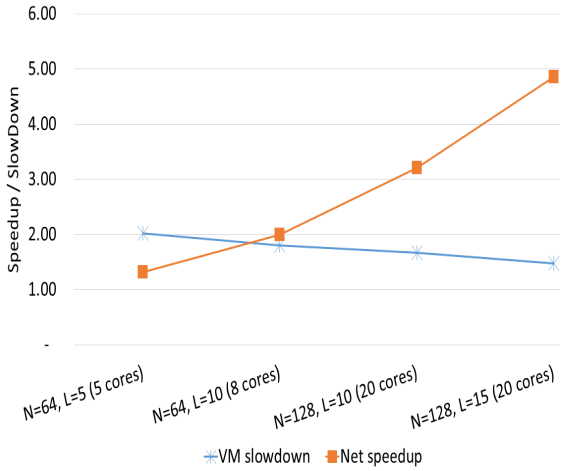

FIGURE 19. Overhead of Virtualization and the Net Speedup for the best speedup for each dataset.

the figure shows the net speedup considering the best speedup core number for each dataset from the previous figure of the scalability analysis. The net speedup ranges from 1.3× for the smallest dataset $\{N = 64, L = 5\}$ to almost 5× for the largest dataset $\{N = 128, L = 15\}$.

VI. CONCLUSION

In this paper, we propose a parallel semi-blind channel estimation algorithm for two-way relay networks. Our goal is to leverage the network virtualization and parallel/cloud computing capabilities of C-RANs, in order to provide a real-time implementation of this highly accurate estimation algorithm. A course look at the algorithm and its loop-carried dependency may suggest that parallelization is not be possible. However, a deep analysis utilizing data decomposition as well as exploratory decomposition reveals amenability to parallelization leading to significant execution-time speedup. The obtained speedup scales well with problem size, with more than 7× and 5× speedup obtained for large datasets with and without virtualization overhead respectively. In conclusion,

the utilization of parallel processing power of nowadays' cloud computing systems to efficiently implement even more complex, multidimensional and sophisticated communication algorithms will be witnessed in the near future. Our future research will investigate the energy efficiency of the proposed parallel Two-way relay semi-blind channel estimation utilizing C-RAN Architecture. We are also interested to parallelize other communication modules of the C-RAN.

ACKNOWLEDGMENT

This research was supported by the University of Sharjah, in part by the Distributed and Networked Systems Research Group Operating Grant 150410, in part by Targeted Project Grant 1602040336-P, and in part by Competitive Project Grant 18020403109.

REFERENCES

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5G be?" *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.
[2] C.-L. I, C. Rowell, S. Han, Z. Xu, G. Li, and Z. Pan, "Toward green and soft: A 5G perspective," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 66–73, Feb. 2014.
[3] M. Peng, C. Wang, V. Lau, and H. V. Poor, "Fronthaul-constrained cloud radio access networks: Insights and challenges," *IEEE Wireless Commun.*, vol. 22, no. 2, pp. 152–160, Apr. 2015.
[4] J. Wu, Z. Zhang, Y. Hong, and Y. Wen, "Cloud radio access network (C-RAN): A primer," *IEEE Netw.*, vol. 29, no. 1, pp. 35–41, Jan. 2015.
[5] M. Peng, Y. Li, Z. Zhao, and C. Wang, "System architecture and key technologies for 5G heterogeneous cloud radio access networks," *IEEE Netw.*, vol. 29, no. 2, pp. 6–14, Mar. 2015.
[6] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for mobile networks—A technology overview," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 405–426, 1st Quart., 2015.
[7] J. Wu, S. Rangan, and H. Zhang, *Green Communications: Theoretical Fundamentals, Algorithms, and Applications*. Boca Raton, FL, USA: CRC Press, 2016.
[8] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *Proc. Australas. Telecommun. Netw. Appl. Conf. (ATNAC)*, Melbourne, VIC, Australia, Nov. 2014, pp. 117–122.
[9] K. Gai, M. Qiu, L. Tao, and Y. Zhu, "Intrusion detection techniques for mobile cloud computing in heterogeneous 5G," *Secur. Commun. Netw.*, vol. 9, no. 16, pp. 3049–3058, Feb. 2015.
[10] L. T. Yang and M. Guo, *High-Performance Computing: Paradigm Infrastructure*, vol. 44. Hoboken, NJ, USA: Wiley, 2005.
[11] D. A. Bader and G. Cong, "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Santa Fe, NM, USA, Apr. 2004.
[12] B. Rankov and A. Wittneben, "Spectral efficient protocols for half-duplex fading relay channels," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 2, pp. 379–389, Feb. 2007.
[13] S. Abdallah, "Spectrally efficient channel estimation for asynchronous amplify-and-forward two-way relay networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 11, pp. 7333–7347, Nov. 2017.
[14] S. Chakraborty and D. Sen, "Joint estimation of MCFOs and channel gains for two-way multi-relay systems with high mobility," *IEEE Wireless Commun. Lett.*, vol. 6, no. 5, pp. 610–613, Oct. 2017.
[15] X. Li, C. Tepedelenlioglu, and H. Senol, "Channel estimation for residual self-interference in full-duplex Amplify-and-Forward two-way relays," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4970–4983, Aug. 2017.
[16] M.-L. Wang, C.-P. Li, and W.-J. Huang, "Semiblind channel estimation and precoding scheme in two-way multirelay networks," *IEEE Trans. Signal Process.*, vol. 65, no. 10, pp. 2576–2587, May 2017.
[17] S. Abdallah and I. N. Psaromiligkos, "Semi-blind channel estimation with superimposed training for OFDM-based AF two-way relaying," *IEEE Trans. Wireless Commun.*, vol. 13, no. 5, pp. 2467–2468, May 2014.
[18] A. Cilardo, L. Gallo, A. Mazzeo, and N. Mazzocca, "Efficient and scalable openmp-based system-level design," in *Proc. Conf. Design, Autom. Test Eur.* San Jose, CA, USA: EDA Consortium, 2013, pp. 988–991.
[19] G. Taentzer, "Parallel and distributed graph transformation: Formal description and application to communication-based systems," Ph.D. dissertation, Technische Univ. Berlin, Berlin, Germany, 1996.
[20] J. Tang, W. P. Tay, and T. Q. S. Quek, "Cross-layer resource allocation with elastic service scaling in cloud radio access network," *IEEE Trans. Wireless Commun.*, vol. 14, no. 9, pp. 5068–5081, Sep. 2015.
[21] A. Douik, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini, "Coordinated scheduling and power control in cloud-radio access networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2523–2536, Apr. 2016.
[22] L. Liu and W. Yu, "Cross-layer design for downlink multihop cloud radio access networks with network coding," *IEEE Trans. Signal Process.*, vol. 65, no. 7, pp. 1728–1740, Apr. 2017.
[23] W.-C. Liao, M. Hong, H. Farmanbar, X. Li, Z.-Q. Luo, and H. Zhang, "Min flow rate maximization for software defined radio access networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1282–1294, Jun. 2014.
[24] A. Alameer Ahmad, H. Dahrouj, A. Chaaban, A. Sezgin, and M.-S. Alouini, "Interference mitigation via rate-splitting and common message decoding in cloud radio access networks," *IEEE Access*, vol. 7, pp. 80350–80365, 2019.
[25] I. El Bakoury and B. Nazer, "Integer-forcing architectures for uplink cloud radio access networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2336–2351, Apr. 2020.
[26] M. Saimler and S. C. Ergen, "Uplink/downlink decoupled energy efficient user association in heterogeneous cloud radio access networks," *Ad Hoc Netw.*, vol. 97, Feb. 2020, Art. no. 102016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870519304524
[27] W. Hao, G. Sun, J. Zhang, P. Xiao, and L. Hanzo, "Secure millimeter wave cloud radio access networks relying on microwave multicast fronthaul," *IEEE Trans. Commun.*, vol. 68, no. 5, pp. 3079–3095, May 2020.
[28] Q. Hu, M. Peng, Z. Mao, X. Xie, and H. V. Poor, "Training design for channel estimation in uplink cloud radio access networks," *IEEE Trans. Signal Process.*, vol. 64, no. 13, pp. 3324–3337, Jul. 2016.
[29] H. Chunjing, L. Jian, M. Zhendong, and B. Yourong, "Channel estimation for full-duplex relay transmission in cloud radio access networks," *China Commun.*, vol. 12, no. 11, pp. 1–8, Nov. 2015.
[30] Z. Zhao, Y. Ban, D. Chen, Z. Mao, and Y. Li, "Joint design of iterative training-based channel estimation and cluster formation in cloud-radio access networks," *IEEE Access*, vol. 4, pp. 9643–9658, Oct. 2016.
[31] Y. Ban, Q. Hu, Z. Mao, and Z. Zhao, "Semi-blind pilot-aided channel estimation in uplink cloud radio access networks," *China Commun.*, vol. 13, no. 9, pp. 72–79, Sep. 2016.
[32] R. G. Stephen and R. Zhang, "Uplink channel estimation and data transmission in millimeter-wave CRAN with lens antenna arrays," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6542–6555, Dec. 2018.
[33] J. Wang, J. Yi, R. Han, L. Bai, and J. Choi, "Variational Bayesian inference for channel estimation and user activity detection in C-RAN," *IEEE Wireless Commun. Lett.*, vol. 9, no. 7, pp. 953–956, Jul. 2020.
[34] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A. M. Vidal, "Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector," *IEEE Trans. Veh. Technol.*, vol. 61, no. 8, pp. 3796–3800, Oct. 2012.
[35] A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Implementation of a fully-parallel turbo decoder on a general-purpose graphics processing unit," *IEEE Access*, vol. 4, pp. 5624–5639, Jun. 2016.
[36] T. Chen and H. Leib, "GPU acceleration for fixed complexity sphere decoder in large MIMO uplink systems," in *Proc. IEEE 28th Can. Conf. Electr. Comput. Eng. (CCECE)*, Halifax, NS, Canada, May 2015, pp. 771–777.
[37] B. Gokalgandhi, C. Segerholm, N. Paul, and I. Seskar, "Accelerating channel estimation and demodulation of uplink OFDM symbols for large scale antenna systems using GPU," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Honolulu, HI, USA, Feb. 2019, pp. 955–959.
[38] J. Nocedal and S. Wright, *Numerical Optimization*. Berlin, Germany: Springer-Verlag, 1999.
[39] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K: Cambridge Univ. Press, 2004.
[40] P. Pacheco, *An Introduction to Parallel Programming*. Amsterdam, The Netherlands: Elsevier, 2011.
[41] S. Brawer, *Introduction to Parallel Programming*. New York, NY, USA: Academic, 2014.
[42] C. Sanderson, "Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments," Tech. Rep., 2010, pp. 1–15.
[43] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *Proc. 1982 SIGPLAN Symp. Compiler Construct. (SIGPLAN)*. Boston, MA, USA: ACM, Jun. 1982, vol. 17, no. 6, pp. 120–126.

[44] R. Shea, H. Wang, and J. Liu, "Power consumption of virtual machines with network transactions: Measurement and improvements," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 1051–1059.
[45] R. S. Alhumaima, R. K. Ahmed, and H. S. Al-Raweshidy, "Maximizing the energy efficiency of virtualized C-RAN via optimizing the number of virtual machines," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 4, pp. 992–1001, Dec. 2018.

ALI A. EL-MOURSY (Senior Member, IEEE) received the Ph.D. degree in high-performance computer architecture from the University of Rochester, Rochester, NY, USA, in 2005. He was with the Software Solution Group, Intel Corporation, CA, USA, until early 2007. In 2007, he joined the Electronics Research Institute, Giza, Egypt. He was also a Visitor Research Scientist with the IBM Cairo Technology Development Center, Egypt, from February 2007 to January 2010. In September 2010, he joined the ECE Department, University of Sharjah, Sharjah, United Arab Emirates, as an Assistant Professor, where he has been promoted to the Associate Professor, in January 2017. His research interests include high-performance computer architecture, multi-core multi-threaded mirco-architecture, power-aware micro-architecture, simulation and modeling of architecture performance and power, workload profiling and characterization, cell programming, high performance computing, parallel computing, and cloud computing.

SAEED ABDALLAH (Member, IEEE) received the B.Eng. degree in computer and communications engineering from the American University of Beirut, Beirut, Lebanon, in 2005, and the M.Sc. and Ph.D. degrees in electrical engineering from McGill University, Montreal, QC, Canada, in 2008 and 2013, respectively. He held a postdoctoral fellow position at the Department of Electrical and Computer Engineering, Queen's University, from 2013 to 2014. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Sharjah, Sharjah, United Arab Emirates. His research interests include signal processing for wireless communications, with a special emphasis on relay networks, multicarrier systems, MIMO and massive MIMO systems, Wi-Fi systems, channel estimation/prediction, and adaptive modulation.

MOHAMED SAAD (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from McMaster University, Hamilton, Canada, in 2004. He has held research positions with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, and the Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Canada. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, University of Sharjah, United Arab Emirates. His research interests include networking, communications, and optimization, with current activity focused on the optimal design of wireless and wired communication networks, and optimal network resource management. He was a recipient of the Best Paper Award at the IEEE Symposium on Computers and Communications, Riccione, Italy, in June 2010. He was a recipient of the University of Sharjah Annual Incentive Award for Distinguished Faculty Members, for excellence in research, in April 2010 (university-wide). He also received the two Best Teaching Awards from the IEEE Women in Engineering Society, University of Sharjah, in 2007 and 2009, respectively. He was also a recipient of the 2005-2006 Natural Sciences and Engineering Research Council of Canada (NSERC) Post-Doctoral Fellowship. He is an Editor of the *International Journal of Distributed Sensor Networks*.

KHAWLA ALNAJJAR (Member, IEEE) received the B.S. degree in electrical engineering, communication track, from United Arab Emirates University (UAEU), Al-Ain, in 2008, the M.S. and P.E.E. degrees in electrical engineering from Columbia University, New York, in 2010 and 2012, respectively, and the Ph.D. degree in electrical and electronics engineering from the University of Canterbury, Christchurch, New Zealand, in 2015. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Sharjah, United Arab Emirates. Her research interests include wireless communication systems, mathematical statistics, and network information theory and power grids. She has received more than 30 competitive awards for her successful studies and research during these ten years.