Adaptive Subcarrier Assignment and Bit Allocation for Multiuser OFDM System Using Ordinal Optimization Approach

Shin-Yeu Lin and Jung-Shou Huang

***Abstract*—In this paper, we propose an ordinal optimization (OO) theory–based four-stage approach to deal with the subcarrier assignment and bit allocation problem of a multiuser orthogonal frequency-division multiplexing (OFDM) system. The four-stage OO approach ensures the quality of the obtained solution, however, at the cost of solving a continuous version of the considered problem in the first stage. To resolve this computational complexity problem, we propose a hardware-implementable dual projected gradient (DPG) method to exploit deep-submicrometer technology. Compared to some existing methods using numerous simulation cases with randomly generated parameters, our approach is excellent in the aspect of power consumption. In the meantime, the estimated computation time of our approach can meet the real-time application requirement. Furthermore, we use extensive simulations to compare our good-enough solutions with the true optimal solutions, and the results show that the objective value obtained by our approach deviates from the optimal objective value around 1% on average.**

***Index Terms*—Multiuser orthogonal frequency-division multiplexing (OFDM) system, ordinal optimization (OO), power control, resource allocation, wireless data transmission.**

I. Introduction

DUE TO THE increase of mobile users and devices, various resource management techniques, such as dynamic channel allocation [1] and dynamic fair resource allocation schemes [2], have been studied. One of the difficulties for such techniques is to keep track of the most updated state of the mobile users or devices caused by their mobility and portability and provide them with the appropriate resources. Therefore, the real-time requirement is the premise of the wireless network resource management solution methods to deal with the high mobility of the dynamic behaviors of mobile users and devices. Among the existing dynamic resource management problems in wireless networks, adaptive subcarrier assignment and bit allocation of an orthogonal frequency-division multiplexing (OFDM) system is a very fundamental issue in mobile communication. There are two types of formulations on this issue. One is the margin adaptive (MA) optimization, which minimizes the total consumed power under a data rate constraint [3], and the other is the rate adaptive (RA) optimization, which maximizes the data rate under a power constraint [4]. Kim *et al.* had shown in [5] that the RA optimization problem can be solved via recursive MA optimization. Therefore, in this paper, we will focus on the adaptive subcarrier assignment and bit allocation problem of MA optimization, with emphasis on the solution quality and the computational efficiency to meet the real-time application requirement.

The considered problem formulated by Wong *et al.* in their pioneer work [3] is a very hard nonlinear integer programming problem; thus, various heuristic solution methods are proposed in [3]–[9]. Wong *et al.* employed a Lagrangian relaxation method in [3] to solve the continuous version of the adaptive subcarrier assignment and bit allocation problem. They then rounded the optimal continuous subcarrier assignment solution off to the closest integer solution. Kim *et al.* [5] converted the adaptive subcarrier assignment and bit allocation problem formulated in [3] into a linear integer programming problem and employed a suboptimal approach to separately perform the subcarrier assignment and bit allocation. To claim for real-time application by not using a mathematical programming approach, Ergen *et al.* [6] proposed a heuristic two-module scheme, Kivanc *et al.* [7] and Zhang [8] proposed two-step subcarrier assignment approaches, and Han *et al.* [9] proposed an iterative grouping scheme to improve the performance by exchanging subcarrier assignment sets.

In general, to obtain a better solution of a *hard optimization problem*, such as the considered problem, usually requires a sophisticated but computationally intensive algorithm. This point had been commented on the method proposed by Wong *et al.* [3] in [5]–[9]. The *purpose* of this paper is to counter this seemingly correct argument by proposing a method that will not only obtain a *good-enough feasible solution* (better than the solution, if feasible, obtained by Wong *et al.*), but also meet the requirement of real-time application. Our approach consists of four *ordinal optimization* (OO) theory-based stages, and the most computationally intensive stage among the four lies in the first stage, where we need to solve a continuous version of the considered problem. To cope with the computational complexity caused by the nonlinear programming algorithm, we propose a *hardware-implementable* dual projected gradient (DPG) method to exploit the advantage of deep-submicrometer technology to obtain the optimal continuous solution extremely quickly.

Manuscript received October 16, 2006; revised May 27, 2007, September 3, 2007, and November 14, 2007. This work was supported in part by the National Science Council of Taiwan, R.O.C., under Grant NSC95-2221-E-009-099-MY2. The review of this paper was coordinated by Dr. S. Vishwanath.

S.-Y. Lin is with the Department of Electrical and Control Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: sylin@cc.nctu.edu.tw).

J.-S. Huang is with the Department of Electrical and Control Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C., and also with Elan Microelectronics Corporation, Hsinchu 300, Taiwan, R.O.C. (e-mail: rong@emc.com.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2007.914053

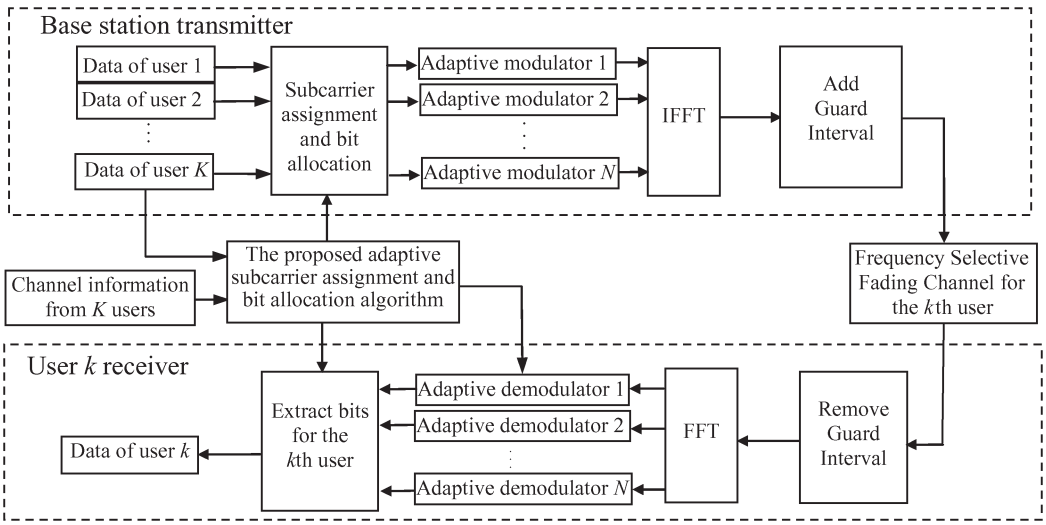

Fig. 1. Block diagram of a multiuser OFDM system with subcarrier assignment and bit location.

To demonstrate that the DPG method is really hardware implementable, we will present the corresponding hardware computing architecture, which will serve as the basis to estimate the computation time of the DPG method. However, to maintain the fluency of presentation, we put the hardware computing architecture in Appendix B, and readers who are not interested in the details may neglect this part. In addition to the Appendices, we organize our paper in the following manner. In Section II, we will state the considered problem. In Section III, we will present the DPG method to solve the continuous version of the considered problem. In Section IV, we will present the OO-theory-based four-stage approach for finding a good-enough feasible solution. In Section V, we will present some test results and compare our approach with some existing methods in the aspects of power consumption and computation time. In Section VI, we conclude the paper.

II. Problem Statement

As shown in Fig. 1, the OFDM system model employed here is derived from [3]. We assume that the system has K users to share N subcarriers. Each user's data rate request will be allocated to a nonoverlapping set of subcarriers and distributed among them. The allocating period in this model is a time interval that consists of several OFDM symbols and is assumed to be short enough so that users' channel gains will stay approximately constant. It is also assumed that a subcarrier cannot be shared by more than one user.

In the transmitter part of Fig. 1, the serial data from K users are fed into the block represented by the proposed adaptive subcarrier assignment and bit allocation algorithm. The algorithm will be executed in every allocation period to assign the set of subcarriers to each user and the number of bits to be transmitted on each assigned subcarrier based on the updated channel information of all users. For each subcarrier, the adaptive modulator will apply a proper modulation scheme to each symbol, depending on the number of bits assigned to the subcarrier, and the modulated symbols are transformed into time-domain samples by an inverse fast Fourier transform (IFFT), as indicated in Fig. 1. The guard interval is then added to ensure orthogonality between the subcarriers, provided that the maximum time dispersion is less than the guard interval. Finally, the transmitted signals pass through different frequency-selective fading channels to different users.

We assume that the subcarrier assignment and bit allocation information is sent to the receivers via a separate control channel. For the sake of simplicity, we only show the receiver part of one user in Fig. 1. At the kth user's receiver part, the guard interval is removed to eliminate the intersymbol interference, and the time sample of the kth user is transformed into modulated symbols using the fast Fourier transform (FFT). The modulation information is then used to configure the demodulators while the subcarrier assignment information is used to extract the demodulated bits from the subcarriers assigned to the kth user.

In this paper, we focus on proposing an efficient and effective algorithm to solve the following adaptive subcarrier assignment and bit allocation problem of the OFDM system for a good-enough feasible solution.[1] That is

$$\min_{c_{k,n},\rho_{k,n}} P_T\left(=\sum_{n=1}^{N}\sum_{k=1}^{K}\frac{\rho_{k,n}}{\alpha_{k,n}^2}f_k(c_{k,n})\right)$$

subject to

$$\begin{aligned}
R_k &= \sum_{n=1}^{N} c_{k,n}, && k=1,\ldots,K\\
\sum_{k=1}^{K}\rho_{k,n} &= 1, && n=1,\ldots,N\\
c_{k,n} &\in D, && \text{for all } k,n\\
\rho_{k,n} &= \begin{cases} 0, & \text{if } c_{k,n}=0\\ 1, & \text{otherwise,}\end{cases} && \text{for all } k,n
\end{aligned} \tag{1}$$

[1] Notation employed here is derived from [3].

where $f_k(c)$ denotes the required transmission power for c bits of user k *when the channel gain is equal to unity*, $\alpha_{k,n}$ is the channel gain for user k using subcarrier n, $f_k(c_{k,n})/\alpha_{k,n}^2$ denotes the transmission power for user k using subcarrier n, P_T denotes the total transmission power to be minimized, $\rho_{k,n}$ is an indicator variable (a subcarrier can be occupied by at most one user as described by the equality constraint on $\rho_{k,n}$), R_k (bits per OFDM symbol) denotes the requested data rate of the kth user, $c_{k,n}$ denotes the number of bits of the kth user assigned to the nth subcarrier, and $D = \{0, 1, 2, \ldots, M\}$ denotes the set of all possible values for $c_{k,n}$, and thus, the first equality constraint in the problem formulated in (1) implies that the subcarrier assignment and bit allocation should meet the user's data rate request.

Since problem (1) is a computationally intractable *combinatorial problem*, Wong *et al.* introduced the variable $r_{k,n} = c_{k,n}\rho_{k,n}$ to transform (1) into the following continuous-variable convex optimization problem over a convex set

$$\min_{r_{k,n}\in[0,M\rho_{k,n}]\rho_{k,n}\in[0,1]} \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\rho_{k,n}}{\alpha_{k,n}^2} f_k\left(\frac{r_{k,n}}{\rho_{k,n}}\right)$$

subject to

$$R_k = \sum_{n=1}^{N} r_{k,n}, \qquad k = 1, \ldots, K$$

$$1 = \sum_{k=1}^{K} \rho_{k,n}, \qquad n = 1, \ldots, N \tag{2}$$

where both $\rho_{k,n}$ and $r_{k,n}$ are continuous variables, satisfying $0 \le \rho_{k,n} \le 1$ and $0 \le r_{k,n} \le M\rho_{k,n}$, respectively. Note that when $\rho_{k,n} = 0$, $r_{k,n} = 0$, and $0/0$ becomes undefined; therefore, we define $f(0/0)$ in (2) as $f(0)$. The solution of (1) obtained by Wong *et al.* is to round the optimal solution of (2) off to the nearest discrete solution. Despite the fact that the consumed power could be significantly reduced while maintaining QoS, their approach faces the following two challenges: 1) the *computational complexity* of the Lagrangian relaxation method and 2) the possible *infeasibility* and no guarantee to be a good-enough solution by arbitrarily rounding off [7], [8]. To cope with these two challenges, we propose an OO-theory-based four-stage approach to solve (1) for a good-enough feasible solution. Stage 1, among the four, is the most computationally intensive stage, because we need to solve (2). Thus, for the sake of fluency in presentation, we will present our hardware-implementable numerical method for solving (2) in the following section first, then followed by the OO-theory-based four-stage approach.

III. Hardware Implementable DPG Method

If we apply a typical Lagrangian relaxation method to solve (2), there will be a *singularity* problem in the variable $\rho_{k,n}$, just like that shown in [3]. To develop a hardware-implementable dual-type method, we need to eliminate this singularity problem by adding extra terms in the objective function of (2) to strictly convexify $\rho_{k,n}$, for every k, n, as follows:

$$\min_{r_{k,n}\in[0,M\rho_{k,n}]\rho_{k,n}\in[0,1]} \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\rho_{k,n}}{\alpha_{k,n}^2} f_k\left(\frac{r_{k,n}}{\rho_{k,n}}\right) + \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\sigma}{2}\rho_{k,n}^2$$

subject to

$$R_k = \sum_{n=1}^{N} r_{k,n}, \qquad k = 1, \ldots, K$$

$$1 = \sum_{k=1}^{K} \rho_{k,n}, \qquad n = 1, \ldots, N. \tag{3}$$

If $\sigma > 0$, (3) is a convex programming problem with a strictly convex objective function.

Remark 1: 1) Adding the extra terms $\sum_{n=1}^{N}\sum_{k=1}^{K} (\sigma/2)\rho_{k,n}^2$ will help us build the surrogate model in stage 1 of our OO-theory-based four-stage approach, as will be shown later. 2) The optimal solution of (3) is a good approximate solution of (2), provided that σ is small enough.

The DPG method will solve the dual problem of (3), as shown in (4), instead of solving (3) directly. We have

$$\max \phi(\lambda) \tag{4}$$

where $\lambda = (\lambda_1^r, \ldots, \lambda_K^r, \lambda_1^\rho, \ldots, \lambda_N^\rho)^T$ is the Lagrange multiplier vector such that λ_k^r corresponds to the kth user's data rate request constraint, and λ_n^ρ corresponds to the nth subcarrier assignment constraint, and the dual function $\phi(\lambda)$ is defined by

$$\begin{aligned}\phi(\lambda) = &\min_{r_{k,n}\in[0,M\rho_{k,n}]\rho_{k,n}\in[0,1]} \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\rho_{k,n}}{\alpha_{k,n}^2} f_k\left(\frac{r_{k,n}}{\rho_{k,n}}\right) \\ &+ \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\sigma}{2}\rho_{k,n}^2 + \sum_{k=1}^{K} \lambda_k^r \left(R_k - \sum_{n=1}^{N} r_{k,n}\right) \\ &+ \sum_{n=1}^{N} \lambda_n^\rho \left(\sum_{k=1}^{K} \rho_{k,n} - 1\right).\end{aligned} \tag{5}$$

By suitably rearranging the terms, (5) can be rewritten in a more compact form as (6), shown at the bottom of the

$$\phi(\lambda) = \min_{r_{k,n}\in[0,M\rho_{k,n}]\rho_{k,n}\in[0,1]} \sum_{n=1}^{N}\sum_{k=1}^{K} \left\{\frac{\rho_{k,n}}{\alpha_{k,n}^2} f_k\left(\frac{r_{k,n}}{\rho_{k,n}}\right) + \frac{\sigma}{2}\rho_{k,n}^2 + \frac{1}{N}\lambda_k^r R_k - \lambda_k^r r_{k,n} + \lambda_n^\rho \rho_{k,n} - \frac{1}{K}\lambda_n^\rho\right\} \tag{6}$$

page. The DPG method employs the following iterations to solve (4):

$$\lambda(t+1) = \lambda(t) + \beta(t)\nabla\phi\left(\lambda(t)\right) \tag{7}$$

where t denotes the iteration index, $\beta(t)$ is a positive step size, and $\nabla\phi(\lambda(t)) = (\partial\phi(\lambda(t))/\partial\lambda_1^r, \ldots, \partial\phi(\lambda(t))/\partial\lambda_K^r, \partial\phi(\lambda(t))/\partial\lambda_1^\rho, \ldots, \partial\phi(\lambda(t))/\partial\lambda_N^\rho)^T$ is the gradient of $\phi(\lambda(t))$ evaluated at $\lambda = \lambda(t)$, which can be calculated by the following formula [10]:

$$\frac{\partial\phi\left(\lambda(t)\right)}{\partial\lambda_k^r} = R_k - \sum_{n=1}^{N}\hat{r}_{k,n}, \qquad k = 1, \ldots, K \tag{8}$$

$$\frac{\partial\phi\left(\lambda(t)\right)}{\partial\lambda_n^\rho} = \sum_{k=1}^{K}\hat{\rho}_{k,n} - 1, \qquad n = 1, \ldots, N. \tag{9}$$

The $(\hat{r}^T, \hat{\rho}^T) = (\hat{r}_{1,1}, \ldots, \hat{r}_{K,N}, \hat{\rho}_{1,1}, \ldots, \hat{\rho}_{K,N})$ in (8) and (9) is the solution of the minimization problem on the right-hand side (RHS) of (6) for a given $\lambda(t)$. Therefore, to obtain $\nabla\phi(\lambda(t))$, we need to solve for $\hat{r}$ and $\hat{\rho}$ first, and the key to making the DPG method *hardware-implementable* is that we use a two-phase strategy to fulfill this task.

The first phase is to solve the minimization problem on the RHS of (6) without the inequality constraints on $r_{k,n}$ and $\rho_{k,n}$, which is shown in (10) and will be called the *unconstrained minimization problem*. Thus, we have

$$\min \sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\frac{\rho_{k,n}}{\alpha_{k,n}^2}f_k\left(\frac{r_{k,n}}{\rho_{k,n}}\right) + \frac{\sigma}{2}\rho_{k,n}^2 + \frac{1}{N}\lambda_k^r R_k - \lambda_k^r r_{k,n} + \lambda_n^\rho\rho_{k,n} - \frac{1}{K}\lambda_n^\rho\right\}. \tag{10}$$

We denote the optimal solution of the unconstrained minimization problem (10) by $(\widetilde{r}^T, \widetilde{\rho}^T) = (\widetilde{r}_{1,1}, \ldots, \widetilde{r}_{K,N}, \widetilde{\rho}_{1,1}, \ldots, \widetilde{\rho}_{K,N})$. $(\widetilde{r}^T, \widetilde{\rho}^T)$ can be obtained by solving the first-order necessary conditions, which can be fully decomposed into $N \cdot K$ independent 2×2 equations, as shown in the following equation, for $n = 1, \ldots, N$ and $k = 1, \ldots, K$:

$$\frac{1}{\alpha_{k,n}^2}f_k'\left(\frac{\widetilde{r}_{k,n}}{\widetilde{\rho}_{k,n}}\right) - \lambda_k^r(t) = 0 \tag{11}$$

$$\frac{1}{\alpha_{k,n}^2}\left[f_k\left(\frac{\widetilde{r}_{k,n}}{\widetilde{\rho}_{k,n}}\right) - f_k'\left(\frac{\widetilde{r}_{k,n}}{\widetilde{\rho}_{k,n}}\right)\frac{\widetilde{r}_{k,n}}{\widetilde{\rho}_{k,n}}\right] + \sigma\widetilde{\rho}_{k,n} + \lambda_n^\rho(t) = 0. \tag{12}$$

For each n and k, the simple 2×2 equations shown in (11) and (12) can be solved analytically in (13) and (14), shown at the bottom of the page, where f'^{-1} can be derived once f is given, e.g., the f given in (16) in Section V.

The second phase is to handle the inequality constraints disappearing in (10) by projecting $(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ onto the range $[0, M\rho_{k,n}] \times [0, 1]$ for each k and n. The projection can be calculated based on simple geometry in (15), shown at the bottom of the page. The resulting projection will be the optimal solution $(\hat{r}^T, \hat{\rho}^T) = (\hat{r}_{1,1}, \ldots, \hat{r}_{K,N}, \hat{\rho}_{1,1}, \ldots, \hat{\rho}_{K,N})$ of the minimization problem on the RHS of (6), as had been proven in [11] and [12].

Convergence of the DPG method using a positive constant step size $\hat{\beta}$ (i.e., setting $\beta(t) = \hat{\beta}$ for every t) can be proved like that of the dual projected pseudo-quasi-Newton method in [11] and [12]. A typical value of $\hat{\beta}$ is 0.5.

As previously indicated, the optimal solution of (3) will be an approximate solution of (2) if σ is sufficiently small. However, a larger σ will speed up the convergence of the DPG method. Thus, we let $\sigma_0\ (= 1), \sigma_1, \ldots, \sigma_{j_{\max}}$ be a decreasing sequence of σ such that $\sigma_{j+1} = \eta\sigma_j$, where $\eta(< 1)$ is a reducing factor, and $j_{\max}$ is a positive integer that makes $\sigma_{j_{\max}}\ (= \eta^{j_{\max}})$ small enough. Then, we can initially set $\sigma = \sigma_0$ in (3) and use the obtained optimal solution as the initial guess to solve (3) again, with $\sigma = \sigma_1$. Repeating this process until $\sigma = \sigma_{j_{\max}}$, the corresponding optimal solution of (3) will be a very good approximate solution of (2).

Remark 2: Indeed, the DPG method is not a groundbreaking technique from the viewpoint of mathematical

$$\widetilde{r}_{k,n} = \widetilde{\rho}_{k,n}f_k'^{-1}\left(\lambda_k^r(t)\alpha_{k,n}^2\right) \tag{13}$$

$$\widetilde{\rho}_{k,n} = \frac{\lambda_n^\rho(t) - \frac{1}{\alpha_{k,n}^2}\left[f_k\left(f_k'^{-1}\left(\lambda_k^r(t)\alpha_{k,n}^2\right)\right) - \lambda_k^r(t)\alpha_{k,n}^2 f_k'^{-1}\left(\lambda_k^r(t)\alpha_{k,n}^2\right)\right]}{\sigma} \tag{14}$$

$$(\hat{r}_{k,n}, \hat{\rho}_{k,n}) = \begin{cases} (\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n}), & \text{if } 0 \le \widetilde{\rho}_{k,n} \le 1, \quad 0 \le \widetilde{r}_{k,n} \le M\widetilde{\rho}_{k,n} \\ (M, 1), & \text{if } \widetilde{\rho}_{k,n} \ge 1, \quad \widetilde{r}_{k,n} \ge M \\ (\widetilde{r}_{k,n}, 1), & \text{if } \widetilde{\rho}_{k,n} \ge 1, \quad \widetilde{r}_{k,n} < M \\ (M, 1), & \text{if } 0 \le \widetilde{\rho}_{k,n} \le 1, \quad \widetilde{r}_{k,n} \ge M + \frac{1}{M} - \frac{1}{M}\widetilde{\rho}_{k,n} \\ \left(\frac{M}{M^2+1}(M\widetilde{r}_{k,n} + \widetilde{\rho}_{k,n}), \frac{M}{M^2+1}\left(\widetilde{r}_{k,n} + \frac{1}{M}\widetilde{\rho}_{k,n}\right)\right), & \text{if } 0 \le \widetilde{\rho}_{k,n} \le 1, \quad \widetilde{r}_{k,n} < M + \frac{1}{M} - \frac{1}{M}\widetilde{\rho}_{k,n} \\ (0, 0), & \text{if } \widetilde{\rho}_{k,n} < 0 \end{cases} \tag{15}$$

programming. Its value is its simplicity and complete decomposition property that enables the method to be hardware implementable.

The description of the hardware architecture for the DPG method is very detailed but indispensable, as indicated in Section I; thus, we will present it in Appendix B.

IV. OO-Theory-Based Four-Stage Approach

The OO theory [13]–[15] is a recently developed optimization methodology designed to solve a hard optimization problem for a good-enough solution with *high probability*. The basic idea of the OO theory is using a surrogate model to quickly evaluate the estimated performance of a solution to select an *estimated good-enough subset* from the *candidate solution set* using limited computation time. When the size of the solution space is huge, the reduction of the search space can be done in several stages. The surrogate models in the stages can range from very rough to more refined ones, and the exact model will be employed in the last stage when there are only few solutions left in the candidate solution set.

Therefore, to obtain a good-enough solution of (1) while using limited computation time to meet the real-time application requirement in the OFDM system, we employ an OO-theory-based four-stage approach, as presented in the following.

A. Stage 1: Reduce the Search Space of Subcarrier Assignment Using a Continuous-Optimal-Solution-Based Model

Intuitively, a true solution that is neighboring to the optimal solution of the continuous version of the considered problem may be a good-enough solution. However, all the possible subcarrier assignments $\rho_{k,n}$, $k = 1, \ldots, K$, $n = 1, \ldots, N$ are neighboring to the optimal continuous solution, which is denoted by $\rho^*_{k,n}$. Wong *et al.* [3] chose the closest one, which, however, may cause *infeasibility*, and even if it is feasible, there is no guarantee that it is a good-enough subcarrier assignment, as indicated in Section II. Thus, in this stage, we will reduce the subcarrier assignment patterns by excluding all the ineffective subcarrier assignments based on our solution process for obtaining the approximate continuous optimal solution of (2). As previously indicated, $\sigma_0\ (=1), \sigma_1, \ldots, \sigma_{j_{\max}}$ is a decreasing sequence of σ such that $\sigma_{j+1} = \eta\sigma_j$. We let $\rho^*_{k,n}(\sigma_j)$ denote the optimal continuous $\rho_{k,n}$ of (3) when $\sigma = \sigma_j$. Then, we claim that subcarrier n is inappropriate to be assigned to user k if $\rho^*_{k,n}(\sigma_j) = 0$ for every $j = 0, 1, \ldots, j_{\max}$. The reason for this is simple, as stated in the following.

We let $\rho^*_{k',n}(\sigma)$ denote the largest $\rho^*_{k,n}(\sigma)$ among all $k = 1, \ldots, K$ for the given n. The term in the objective function of (3) regarding σ is $\sum_{n=1}^{N}\sum_{k=1}^{K}(\sigma/2)\rho^2_{k,n}$. The sensitivity of this term with respect to $\rho^*_{k,n}(\sigma)$ is $\sigma\rho^*_{k,n}(\sigma)$. Suppose we increase σ by $\Delta\sigma$. Then, the decrease of $\rho^*_{k,n}(\sigma)$ by the amount $-\Delta\rho_{k,n}$, where we assume $\Delta\rho_{k,n} > 0$, will cause an approximate *extra reduction* of the objective value due to the increase of σ by $-\rho^*_{k,n}(\sigma)\Delta\sigma\Delta\rho_{k,n}$. Thus, decreasing $\rho^*_{k',n}(\sigma)$ will be most beneficial if we increase σ. This implies that increasing σ in (3) will force $\rho^*_{k',n}(\sigma)$ to decrease. Then, by the constraint $\sum_{k=1}^{K}\rho_{k,n} = 1$ for a given n, if $\rho^*_{k',n}(\sigma)$ decreases, there must be at least one $k'' \in \{1, \ldots, K\}$, $k'' \neq k'$, such that $\rho^*_{k'',n}(\sigma)$ will increase. Among $\rho^*_{k,n}(\sigma)$, $k = 1, \ldots, K$, $n = 1, \ldots, N$, the ones with $\rho^*_{k,n}(\sigma) = 0$ are the most possible candidates to be $\rho^*_{k'',n}(\sigma)$, because the increase of such $\rho^*_{k,n}(\sigma)$ will cause an approximately zero increment in the objective value because $\rho^*_{k,n}(\sigma)\Delta\sigma\Delta\rho_{k,n} = 0$. Thus, if they do not, it simply implies that it is inappropriate to assign subcarrier n to those users with $\rho^*_{k,n}(\sigma) = 0$ and remaining 0 while σ increases. We have solved a sequence of (3) with σ_j, $j = 0, 1, \ldots, j_{\max}$ and obtained $\rho^*_{k,n}(\sigma_j)$, $j = 0, 1, \ldots, j_{\max}$ for all k and n. Then, based on the above argument, we have that $\rho^*_{k,n}(\sigma_j) = 0$, $j = 0, 1, \ldots, j_{\max}$ implies that subcarrier n is not suitable to be assigned to user k because when σ_{j+1} increases to σ_j $(\sigma_j > \sigma_{j+1})$, remains 0 as $\rho^*_{k,n}(\sigma_{j+1})$.

Therefore, our crude model for selecting roughly good subcarrier assignment patterns in this stage can be stated as follows. We first set $\rho_{k,n} = 0$ for the (k, n)'s whose corresponding optimal continuous $\rho^*_{k,n}(\sigma) = 0$ for all $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_{j_{\max}}$. We denote γ_n as the number of $\rho_{k,n}$'s that are not set to be 0 for a given n. Then, there will be γ_n possible subcarrier assignment patterns for a given n, which means these γ_n nonzero $\rho_{k,n}$'s take turns to be 1. Subsequently, we will choose *feasible* subcarrier assignment patterns from the $\prod_{n=1}^{N}\gamma_n$ possible patterns and form the set of roughly good subcarrier assignment patterns that resulted in this stage. It should be noted that checking the feasibility of a pattern, e.g., $(\rho_{1,1}, \ldots, \rho_{K,1}, \ldots, \rho_{1,N}, \ldots, \rho_{K,N})$, is simply checking whether $M\sum_{n=1}^{N}\rho_{k,n} \geq R_k$ hold for every k.

Remark 3: The reduction of the search space of subcarrier assignments had been reduced from K^N (which is considered to be the worst case and may include the infeasible patterns) to $\prod_{n=1}^{N}\gamma_n$. Based on our simulation experience, a lot of γ_n's are 1.

B. Stage 2: Choose s Estimated Good-Enough Subcarrier Assignment Patterns Using an Approximate Model

To evaluate the estimated performance of the feasible subcarrier assignment patterns obtained in stage 1, we will employ an approximate model to estimate the total power consumption of each pattern as follows. For a given feasible pattern $(\rho_{1,1}, \ldots, \rho_{K,1}, \ldots, \rho_{1,N}, \ldots, \rho_{K,N})$, we let $\hat{N}_k = \sum_{n=1}^{N}\rho_{k,n}$ and $\hat{\alpha}_k = \sum_{n=1}^{N}\rho_{k,n}\alpha_{k,n}/\hat{N}_k$ denote the total number of subcarriers assigned to and the average power consumption coefficient of user k, respectively. We assume that the requested data rate R_k of user k are equally distributed to the assigned subcarriers. Then, the approximate power consumed by user k, which is denoted by $\hat{P}_k$, can be calculated by $\hat{P}_k = (\hat{N}_k/\hat{\alpha}_k^2)f_k(R_k/\hat{N}_k)$. Consequently, the estimated total consumed power for the given pattern, which is denoted by $\hat{P}_T$, will be $\hat{P}_T = \sum_{k=1}^{K}\hat{P}_k$. We apply the above estimation process to each feasible pattern obtained in stage 1 and pick the $s\ (= 50)$ patterns with the smallest $\hat{P}_T$ to form the estimated good-enough subset, which is denoted by SS.

Remark 4: Based on the surrogate model with modeling noise w, the selected s patterns will contain at least y actual top $x\%$ patterns among the $\prod_{n=1}^{N}\gamma_n$ with a probability of 0.95.

There exists a quantitative relationship between the values of s, y, x, w, and $\prod_{n=1}^{N}\gamma_n$, as indicated in [15]. In general, a larger s can have more y and less x, but it will take more time for further evaluation. Therefore, for the real-time application consideration, we set $s = 50$.

C. Stage 3: Choose l Estimated Good-Enough Subcarrier Assignment Patterns From the SS

To evaluate the s $(= 50)$ subcarrier assignment patterns by using an exact model is still very time consuming when meeting the real-time application requirement. Therefore, we will employ a more refined model than the one used in stage 2 to select l $(= 3)$ estimated good-enough patterns from SS. The more refined model that we employ here is a supervised learning artificial neural network (ANN) used to evaluate the estimated consumed power of user k for a given subcarrier assignment pattern $(\rho_{1,1},\ldots,\rho_{K,1},\ldots,\rho_{1,N},\ldots,\rho_{K,N})$. This ANN model is constructed *off-line* using 5000 input/output pairs, and the details are described in the following.

The data associated with user k are the data rate request R_k, the power consumption coefficient $\alpha_{k,n}$, $n = 1,\ldots,N$, and the subcarrier assignment pattern regarding user k, i.e., $(\rho_{k,1},\ldots,\rho_{k,N})$. For such a given data vector $(R_k,\alpha_{k,1},\ldots,\alpha_{k,N},\rho_{k,1},\ldots,\rho_{k,N})^T$, an optimal bit allocation algorithm or the so-called greedy algorithm [16] can be used to optimally allocate the bits to the assigned subcarrier to meet the data rate request while minimizing the power consumption of user k.[2] However, the dimension of the vector $(R_k,\alpha_{k,1},\ldots,\alpha_{k,N},\rho_{k,1},\ldots,\rho_{k,N})^T$ is very large. Thus, we will employ $(R_k,\hat{N}_k,\hat{\alpha}_k,d\alpha_k)^T$ to characterize user k and serve as the input vector to the ANN, where $\hat{N}_k$ and $\hat{\alpha}_k$ had been defined in stage 2, and $d\alpha_k \equiv \max_{n,\rho_{k,n}=1}\ \alpha_{k,n} - \min_{n,\rho_{k,n}=1}\ \alpha_{k,n}$. Consequently, the 5000 input/output pairs used to train the ANN can be obtained as follows. We uniformly select 5000 pairs of $(R_k,\hat{N}_k)$ from the following ranges: $R_k \in [5,150]$ and $\hat{N}_k \in [R_k/M, 2R_k/M]$. For each $\hat{N}_k$, we randomly select distinct $n_1,\ldots,n_{\hat{N}_k}$ from $\{1,\ldots,N\}$, and randomly select α_{k,n_i} from the range [0,1.5] for each $i = 1,\ldots,\hat{N}_k$. The $\hat{\alpha}_k$ and $d\alpha_k$ can be computed accordingly. Thus, we have 5000 input vectors of the form $(R_k,\hat{N}_k,\hat{\alpha}_k,d\alpha_k)^T$. Now, for each $(R_k,\hat{N}_k,\alpha_{k,n_1},\ldots,\alpha_{k,n_{\hat{N}_k}})$, we can use the greedy algorithm to compute the corresponding minimum consumed power of user k denoted by $\widetilde{P}_k$. Thus, $(R_k,\hat{N}_k,\hat{\alpha}_k,d\alpha_k)$ and $\widetilde{P}_k$ form an input/output pair. We then use the obtained 5000 input/output pairs to train a three-layer ANN. The input layer consists of four neurons, which correspond to R_k, $\hat{N}_k$, $\hat{\alpha}_k$, and $d\alpha_k$. We use 15 neurons in the hidden layer, and each neuron uses *tansig* [17] as the transfer function to calculate the output from the summed value of its inputs. There is only one neuron in the output layer that corresponds to the consumed power $\widetilde{P}_k$ of user k, and we use *purelin* [17] as the transfer function. Using the above-mentioned 5000 input/output pairs, we train this ANN by the Levenberg–Marquardt method proposed in [18].

[2]This optimal bit allocation for each subcarrier assignment pattern is what we mean by the exact model, and it is used for the ANN's off-line training.

Remark 5: In general, the more neurons, if there are not too many, there are in the hidden layer, the more accurate the ANN will be. Although our ANN is trained off-line, more neurons in the hidden layer will increase the dimension of the arc weights, thus increasing the computation time to obtain the output of the ANN, which may hurt our purpose of real-time application. Since what we care about here is the performance orders rather than the performance values of the tested input vectors, perfect accuracy is not necessary. Therefore, to save on computation time, we select a moderate number of neurons, i.e., 15, in the hidden layer. In addition, there exist various activation functions such as *step* function, *hard limiter*, *ramp*, *tansig*, *purelin*, etc. [17]. Which function to be used really depends on the application. In our problem, we found that *tansig* and *purelin* are the most suitable activation functions for the hidden layer and output layer, respectively. Furthermore, we found that the Levenberg–Marquardt method converges quickly in training our ANN.

Once the ANN is trained, we can estimate $\widetilde{P}_k$ of each subcarrier assignment pattern in SS by setting up the corresponding input $(R_k,\hat{N}_k,\hat{\alpha}_k,d\alpha_k)$ and feeding it into the ANN, and the output will be the estimated $\widetilde{P}_k$. Thus, the estimated total consumed power of a subcarrier assignment pattern, which is denoted by $\widetilde{P}_T$, in SS will be $\widetilde{P}_T = \sum_{k=1}^{K}\widetilde{P}_k$, and the l $(= 3)$ patterns with the smallest estimated $\widetilde{P}_T$ among the s will be the estimated good-enough subcarrier assignment patterns determined in this stage.

Remark 6: In [19], we have found out through simulation that by using a more accurate surrogate model to evaluate s $(= 50)$ candidate solutions, the top l $(= 3)$ solutions will contain the actual best among the s with a probability of 0.99.

D. Stage 4: Determine the Good-Enough Subcarrier Assignment and Bit Allocation

Since there are only l $(= 3)$ subcarrier assignment patterns left, we can use the exact model, i.e., the greedy algorithm, to calculate the optimal consumed power P_T for each pattern with very limited computation time. The subcarrier assignment associated with the optimal bit allocation, corresponding to the smallest P_T among the three, will be the good-enough solution of (1) that we look for.

V. Test Results and Comparisons

In this section, we will test the performance of the proposed approach in the aspects of solution quality and computational efficiency. We will also compare our approach with the existing subcarrier assignment and bit allocation algorithms such as the algorithm proposed by Wong *et al.* [3], the linear programming approach proposed by Kim *et al.* [5], the iterative algorithm proposed by Ergen *et al.* [6], and Zhang's approach [8].

As depicted in Fig. 1, we assume that the OFDM system has 128 subcarriers (i.e., $N = 128$) over a 5-MHz band. The system uses M-ary quadrature amplitude modulation (MQAM) such that the square signal constellations 4-QAM, 16-QAM, and 64-QAM carry two, four, and six bits/symbol, respectively; therefore, in this system, $D = \{0,2,4,6\}$, and $M = 6$. We adopt the following formula for $f_k(c_{k,n})$ in $f_k(c_{k,n})/\alpha_{k,n}^2$,

appearing in (1), and this formula holds for all k in MQAM at a given BER denoted by P_e [3]:[3]

$$f(c) = \frac{N_0}{3}\left[Q^{-1}\left(\frac{P_e}{4}\right)\right]^2 (2^c - 1) \tag{16}$$

where $Q^{-1}(x)$ is the inverse function of

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{\frac{-t^2}{2}} dt \tag{17}$$

and N_0 denotes the noise power spectral density (PSD) level.[4]

Remark 7: The $f_k(c)$ in our hardware implementation is not limited to the formula given in (16), which is simply an example formula for the purpose of comparisons. However, we have to admit that once a function or a form of $f_k(c)$, which may correspond to certain *coding* and *modulation* schemes, is assumed, changing hardware is not as easy as the software.

In all simulations presented in this section, we set $P_e = 10^{-4}$ for each user, and the wireless channel is modeled as a frequency-selective channel consisting of six independent Rayleigh multipaths. Each multipath is modeled by Clark's flat-fading model [20]. We assumed that the delays and the corresponding gains of the six paths are $100 \cdot p$ ns and e^{-2p} (exponentially decay), respectively, where $p = 0$, 1, 2, 3, 4, and 5 denote the multipath index. Hence, the relative power of the six multipath components are 0, -8.69, -17.37, -26.06, -34.74, and -43.43 dB. We also assume that the average subcarrier channel gain $E|\alpha_{k,n}|^2$ is unity for all k and n. Based on the above assumptions, we generate power consumption coefficients $\alpha_{k,n}$, $k = 1, \ldots, K$, $n = 1, \ldots, N$ using MATLAB for our simulations.

We consider various numbers of users by setting $K = 2, 4, 8$, 16, and 32. For each K, we randomly generate 500 sets of $\alpha_{k,n}$, $k = 1, \ldots, K$, $n = 1, \ldots, N$, based on the aforementioned power consumption coefficient generation process and denote α^i as the ith set in the 500. We assume a fixed total requested data rate R_T (= 512 bits/symbol) and randomly generate R_k, $k = 1, \ldots, K$, based on the constraint $\sum_{k=1}^{K} R_k = R_T$. By the above test setup, we have run our approach for each K, each set of $\alpha_{k,n}$, $k = 1, \ldots, K$, $n = 1, \ldots, N$, and each set of R_k, $k = 1, \ldots, K$. We also apply the four methods mentioned at the beginning of this section to the same test.

For each K associated with the data rate request R_k, $k = 1, \ldots, K$, we denote $abSNR(\alpha^i)$ as the average bit SNR[5] when α^i is used and calculate $\sum_{i=1}^{500} abSNR(\alpha^i)/500$, i.e., the average $abSNR$, resulting from the 500 α^i's using our approach and the other four methods and report them in Fig. 2. We can see that the $\sum_{i=1}^{500} abSNR(\alpha^i)/500$ values obtained by our approach, which are marked by "△" in Fig. 2, are the smallest among all the methods. Moreover, the performance of our approach becomes even better as the number of users increases, as can be observed in Fig. 2.

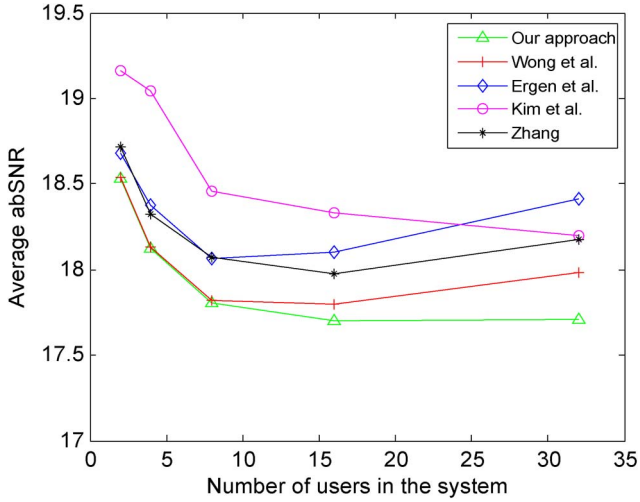

Fig. 2. Average $abSNR$ for $K = 2$, 4, 8, 16, and 32 obtained by the five methods.

To investigate the computational efficiency of our approach and the other four methods, we need to report the average computation time for obtaining $abSNR(\alpha^i)$. However, as we have indicated in Section I that the DPG method will be implemented by integrated circuits, the computation time of our approach is partly real and partly estimated, and its details are stated in the following.

All the computation times of our OO-theory-based four-stage approach, except for the DPG algorithm, are recorded in the employed Pentium 2.4-GHz-processor 512-MB-random-access-memory personal computer, and we denote it by T_R. For $K = 2$, 4, 8, 16, and 32 in the test results shown in Fig. 2, the corresponding average T_R's for obtaining $abSNR(\alpha^i)$ are 1.214, 1.686, 3.386, 5.366, and 11.136 ms, respectively. To estimate the computation time of the DPG method, we base on 90-nm CMOS integrated circuit technology and denote this estimated computation time as T_E. We let $\otimes$, $\oplus$, and read-only memory (ROM) denote the operations multiplication, addition, and accessing the data of the ROM, respectively, and we define $T_{(\cdot)}$ as the computation time for performing the operation $(\cdot)$. Referring to the work of Hsu *et al.* [21] and Kanan *et al.* [22], $T_\otimes = 1.0$ ns for a 16 × 16 bit multiplication,[6] and it takes 1.2 ns to access the data in the ROM. In practical designs, the circuit complexity of a 16 × 16 bit multiplication is five times greater than that of a 16 + 16 bit addition [23, Ch. 5, p. 113, and Ch. 13, p. 433]; thus, we can set $T_\oplus \approx 0.2$ ns. Then, based on

[3]Equation (16) is directly borrowed from [3, Sec. V, p. 1725], which is an approximation based on the bit-error probability $4Q((d^2/(2N_0))^{1/2})$ and the average energy $(M-1)d^2/6$ of an MQAM symbol, where d is the minimum distance between the points in the signal constellation. Furthermore, this formula is also used in [5], [6], and [8].

[4]Once N_0 and P_e are given, $N_0/3[Q^{-1}(P_e/4)]^2$ in (16) is a constant denoted by B. We can rewrite $f(c) = B(2^c - 1)$, and then, $f'^{-1}(c) = \ln(c/B \ln 2)/\ln 2$, which is the term needed in (13) and (14).

[5]It is noted that the average required transmit power (in energy per bit) is defined as the ratio of the overall transmit energy per OFDM symbol, i.e., P_T in (1), to the total number of bits transmitted per OFDM symbol, which consists of 512 bits in our test case. Moreover, we define the average bit SNR as the ratio of the average transmit power, i.e., $P_T/512$, to the noise PSD level N_0. As we have assumed that all the data rates per symbol are fixed at 512 and that N_0 is just a constant, P_T is thus proportional to the average bit SNR. Therefore, for the purpose of comparison, we can use the average bit SNR to replace P_T.

[6]The 16-bit data type has enough precision for the implementation of the DPG algorithm, because the resulting $\rho^*_{k,n}$ that we need from the hardware computation is only whether $\rho^*_{k,n}$ is zero or nonzero and not how accurate the nonzero value is.

TABLE I
AVERAGE COMPUTATION TIMES (IN MILLISECONDS) FOR OBTAINING $abSNR(\alpha^i)$ FOR VARIOUS NUMBERS OF USERS

Computation time (ms) / K / Method	2	4	8	16	32
Our approach	1.42	2.21	4.63	8.49	18.63
Wong *et al.*	103.32	185.3	371.3	701.2	1507.1
Ergen *et al.*	10.18	14.9	18.8	31.1	53.2
Kim *et al.*	24.95	30.5	40.6	96.6	225.9
Zhang	26.81	42.5	45.3	60.3	88.1

the last column of Table III (see Appendix B), we have $T_{\mathrm{PE}_n^1} = 6.6$ ns, $T_{\mathrm{PE}_n^2} = 2.2$ ns, $T_{\mathrm{PE}^4} = 1.6$ ns, $T_{\mathrm{PE}^5} = 1.2$ ns, and $T_{\mathrm{PE}^7} = 1.0$ ns. In our simulations, the values of $t_{\max} \times j_{\max}$ are set to be 8000, 10 000, 12 000, 15 000, and 18 000 for cases of $K = 2$, 4, 8, 16, and 32, respectively. Thus, based on (18) and (19) in Appendix C, the estimated computation times (T_E) of the DPG method are 0.208, 0.52, 1.248, 3.12, and 7.488 ms for the cases when $K = 2$, 4, 8, 16, and 32, respectively. Summing up T_R and T_E, the estimated average computation times of our approach to obtain $abSNR(\alpha^i)$ for various K's are reported in the second row of Table I. We also report the average computation times of the other four methods on the same test case in rows 3–6 of Table I. The method proposed by Wong *et al.* is the most computationally consuming, as indicated in [5]–[9]. Considering that the frame length of a wideband OFDM is 20 ms [24], the proposed approach can meet the real-time application requirement for high mobility circumstances.

As demonstrated above, our approach outperforms the other four methods in the aspect of power consumption, and we can obtain the results in real time.

Remark 8: It seems unfair that the computation times of our algorithm are partly estimated from hardware performance, while the computation times of the other algorithms are entirely from computer simulation. In fact, what we want to assert is that we can achieve the best performance among all the methods (the comparison results shown in Fig. 2) in real time (the data shown in the second row of Table I).

Remark 9: As we have indicated in Remark 2, the DPG method is simple, and hence, it takes more iterations to converge. However, the key to helping speed up our approach is its hardware implementability, which is the reason behind why we estimate its computation time based on the hardware architecture shown in Appendix B rather than the commonly adopted expression.

Remark 10: In the deep-submicrometer technology, the effect of the wire delay is prominent, particularly in the design of a large area or complicated routing. There are two types of wire delay in our hardware architecture: the intra and inter wire delay. The intra wire delay is the wire delay inside the hardware component of a processing element (PE), such as the multiplier. The inter wire delay is the wire delay between PEs and registers of the hardware architecture shown in Fig. 5 of Appendix B. The intra wire delay plays a dominant role in the overall wire delay; however, they had been taken into account in our estimation of computation time. Since our hardware architecture is very regular and modular, the inter wire delay can, at most, be a small fraction of T_E, i.e., the estimated computation time of the DPG algorithm, and will not affect the real-time application.

TABLE II
AVERAGE $(d - D/d) \times 100\%$ FOR EACH SET OF N, K AND ABPS

N	K	Average $\frac{d-D}{d}\times 100\%$		
		ABPS =3	ABPS =4	ABPS =5
32	4	0.246	0.300	0.283
32	6	0.646	1.260	0.874
32	8	1.606	1.634	1.115
32	10	1.837	1.778	1.568
64	4	0.198	0.114	0.131
64	8	0.557	0.422	0.241
64	12	1.872	1.663	1.295
64	16	2.227	2.328	2.090
128	4	0.056	0.081	0.080
128	8	0.131	0.110	0.144
128	16	0.501	0.863	0.852
128	32	2.572	2.842	2.793
Average		1.037	1.116	0.956

To evaluate the actual goodness of the obtained good-enough solutions, we should compare with the optimal solution of (1) using extensive simulations. To cover more system conditions in our simulations, we consider the cases when $N = 32$, 64, and 128 and take four various K's for each N. We define the average bit per subcarrier (ABPS) as $\sum_{k=1}^{K} R_k/N$ to denote the congestion condition of the system. We set $M = 6$ and consider three cases of ABPS, namely ABPS = 3, 4 and 5, for each N and each K. For each (N, K, ABPS), we randomly generate 250 sets of R_k, $k = 1, \ldots, K$, based on the constraint on ABPS, and randomly generate a set of $\alpha_{k,n}$, $k = 1, \ldots, K$, $n = 1, \ldots, N$ for each set of R_k, $k = 1, \ldots, K$. We employ (16) for the $f(c)$ but set $P_e = 10^{-4}$ and $N_0 = 1$. Table II shows the average of 250 $(d - D)/d \times 100\%$ for each (N, K, APBS), where D and d denote the actual optimal power consumption of (1) and the power consumption of good-enough solution obtained by our approach, respectively. For each ABPS, the average of the average $(d - D)/d \times 100\%$ of various N's and K's is shown in the last row of Table II, which indicates that the average deviation of d from D is around 1.0% in various congestion conditions of the system. This shows that the good-enough solutions that we obtained are really good enough.

VI. CONCLUSION

In this paper, we have proposed an OO-theory-based four-stage approach to solve the adaptive subcarrier assignment and bit allocation problem of a multiuser OFDM system for a good-enough feasible solution. To resolve the computational complexity problem caused by the DPG method in our approach, we propose a hardware architecture to implement the DPG method to exploit deep-submicrometer technology. Compared with some existing methods, the quality of the good-enough feasible solution obtained by our approach is excellent, and the estimated computation time meets the real-time application requirement.

APPENDIX A
HARDWARE-IMPLEMENTABLE ALGORITHM OF THE DPG METHOD

To further enhance the computation speed, we will propose a hardware architecture to implement the DPG method. To do so, we need to put the DPG method in algorithmic steps that can be mapped into the operations of the arrays of PEs, which are defined as the hardware for carrying out the arithmetic operations in the DPG method. First, we should modify the convergence criteria of the DPG method by setting a large-enough number of iterations, e.g., $t_{\max}$, such that if $t \geq t_{\max}$, we assume that the DPG method converges. Furthermore, we should predetermine the value of $j_{\max}$, which is the number of times that σ will be reduced.

It can be observed that all the computation formulas of the DPG method, namely, (8), (9), and (13)–(15), achieve a complete decomposition property, i.e., the computations for each k and each n can be carried out independently. Furthermore, all these computations consist of simple arithmetic operations only. These facts imply that the DPG method is very suitable for hardware implementation. However, it is not wise to assign a PE to calculate each individual component, e.g., calculating $\widetilde{\rho}_{k,n}$ for every k and every n in (14), because this will make the size of the integrated circuit chip too big to be implemented. Therefore, to render the difficulty of chip size, we can use N arrays of PEs such that the nth PE array will take care of all the K computations corresponding to one n in (8), (9), and (13)–(15). Although such an arrangement seems to degrade the computational speed, in fact, it will not affect the purpose of real-time application, as shown in Section V and Appendix C. On the basis of using N PE arrays, we can put the DPG method in the following parallel-computation algorithmic steps.

Step 0) Set the values of $\lambda(0)$, $\sigma(0)$, $\eta(<1)$, $t_{\max}$, and $j_{\max}$; set $j=0$ and $t=0$.

Step 1) Set $k=1$ and $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho) = -1$ for each n. (Note that $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ represents a temporary memory for the term $\partial\phi(\lambda(t))/\partial\lambda_n^\rho$.)

Step 2) Compute in parallel $(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ by calculating (13) and (14) for each n.

Step 3) Project in parallel $(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ onto the range $([0, M\rho_{k,n}], [0,1])$ for each n using (15) to obtain $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ for each n.

Step 4) Compute $(\partial\phi(\lambda(t))/\partial\lambda_k^r) = R_k - \sum_{n=1}^{N} \hat{r}_{k,n}$.

Step 5) Compute in parallel $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho) := R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho) + \hat{\rho}_{k,n}$ for each n.

Step 6) Update $\lambda_k^r(t+1) = \lambda_k^r(t) + \hat{\beta}(\partial\phi(\lambda(t))/\partial\lambda_k^r)$.

Step 7) If $k=K$, go to Step 8); otherwise, set $k=k+1$, and return to Step 2).

Step 8) Update in parallel $\lambda_n^\rho(t+1) = \lambda_n^\rho(t) + \hat{\beta}(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ for each n. [Note that the value of $(\partial\phi(\lambda(t))/\partial\lambda_n^\rho) = \sum_{k=1}^{K} \hat{\rho}_{k,n} - 1$ is stored in $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$.]

Step 9) If $t \geq t_{\max}$, go to Step 10); otherwise, set $t = t+1$, and return to Step 1).

Step 10) Set $\sigma(j+1) = \eta\sigma(j)$.

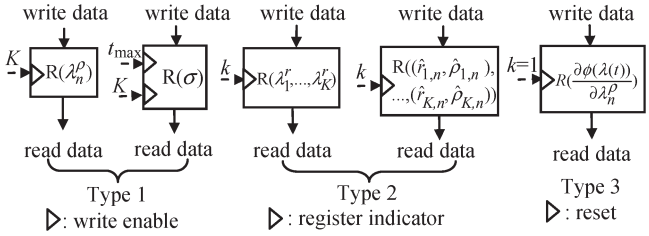

Fig. 3. Three types of registers for storing the temporary computed values of the DPG algorithm.

Step 11) If $j \geq j_{\max}$, go to Step 12); otherwise, set $j = j+1$, $\lambda_k^r(0) = \lambda_k^r(t)$ for each k, $\lambda_n^\rho(0) = \lambda_n^\rho(t)$ for each n, $k=1$, and $t=1$, and return to Step 1).

Step 12) Stop.

Remark 11: The reason behind why we execute Step 5) for K iterations to calculate $\partial\phi(\lambda(t))/\partial\lambda_n^\rho$ for each n is because we use N instead of $N \cdot K$ PE arrays.

APPENDIX B
HARDWARE COMPUTING ARCHITECTURE OF THE DPG ALGORITHM

Mapping the DPG algorithm into a hardware architecture needs to consider the following four parts: 1) the data storage; 2) the computations; 3) the iteration count and the convergence detection for branching the data flow; and 4) the interconnections between PEs and data storage elements. In the following, we will present the details of each part.

1) Regarding the data storage, we employ registers to store the constants η, $t_{\max}$, $j_{\max}$, $\hat{\beta}$, R_k, $\alpha_{k,n}^2$, and $1/\alpha_{k,n}^2$ of the algorithm and the temporary values of the variables $\lambda_n^\rho(t)$, $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$, $\lambda_k^r(t)$, $k=1,\ldots,K$, $\sigma(j)$, $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$, $k=1,\ldots,K$ generated in the algorithm. For the sake of simplicity in interconnection, the registers for storing the constants are embedded in the PE responsible for the computations that need the constants. However, there are three types of registers, denoted by $\mathrm{R}(\Delta)$, for storing the temporary computed values of the variables Δ, as shown in Fig. 3. For example, $\mathrm{R}(\lambda_n^\rho)$ denotes the register for storing the computed value of λ_n^ρ. The type 1 registers are for storing the computed values of λ_n^ρ and σ; λ_n^ρ is updated when $k=K$, and σ is updated when both $k=K$ and $t=t_{\max}$, as indicated in Steps 7) and 8) and Steps 9) and 10), respectively. Therefore, a write enable controlled by the value of k and t is needed, as shown in Fig. 3. The type 2 registers are K-bank registers for storing the computed values of λ_k^r, $k=1,\ldots,K$, or $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$, $k=1,\ldots,K$, such that the kth bank will store the value of λ_k^r or $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$. Since the iteration index of the innermost loop of the DPG algorithm is k, we need a register indicator k to point to the corresponding register bank, as shown in Fig. 3. Furthermore, λ_k^r and $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ are computed for every k, and thus, type 2 registers are always write enabled. The type 3 register is for storing the value of $\sum_{l=1}^{k} \hat{\rho}_{l,n} - 1$; therefore, a type 3 register is always write enabled. However, its value has to be reset to -1 when

TABLE III
CHARACTERISTICS OF PEs

PE	Algorithmic Step	Embedded Constant	Input Data [from]	Output Data [to]	Computation Complexity
PE_n^1	Step 2	$\alpha_{k,n}^2, 1/\alpha_{k,n}^2$	$\lambda_n^\rho(t)$ [R(λ_n^ρ)], $\sigma(j)$ [R(σ)], $\lambda_k^r(t)$ [R($\lambda_1^r,...,\lambda_K^r$)]	$(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ [PE_n^2]	$5\otimes$ & $2\oplus$ & 1 ROM[§]
PE_n^2	Step 3	-	$(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ [PE_n^1]	$\hat{r}_{k,n}$ [PE^4], $\hat{\rho}_{k,n}$ [PE_n^6], $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ [R($(\hat{r}_{k,n}, \hat{\rho}_{k,n}), k = 1,..., K$)]	$2\otimes$ &1 $\oplus$
PE_n^3	Step 8	$\hat{\beta}$	$\sum_{l=1}^{K} \hat{\rho}_{l,n} - 1$ [PE_n^6], $\lambda_n^\rho(t)$ [R(λ_n^ρ)]	$\lambda_n^\rho(t+1)$ [R(λ_n^ρ)]	$1\otimes$ & $1\oplus$
PE^4	Step 4	R_k	$\hat{r}_{k,1}$ [PE_1^2],...,$\hat{r}_{k,N}$ [PE_N^2]	$\frac{\partial\phi(\lambda(t))}{\partial\lambda_k^r}$ [PE^5]	$\log_2(N+1)\oplus$
PE^5	Step 6	$\hat{\beta}$	$\frac{\partial\phi(\lambda(t))}{\partial\lambda_k^r}$ [PE^4], $\lambda_k^r(t)$ [R($\lambda_1^r,...,\lambda_K^r$)]	$\lambda_k^r(t+1)$ [R($\lambda_1^r,...,\lambda_K^r$)]	$1\otimes$ & $1\oplus$
PE_n^6	Step 5	-	$\sum_{l=1}^{k-1} \hat{\rho}_{l,n} - 1$ [$R(\frac{\partial\phi(\lambda(t))}{\partial\lambda_n^\rho})$], $\hat{\rho}_{k,n}$ [PE_n^2]	$\sum_{l=1}^{k} \hat{\rho}_{l,n} - 1$ [$R(\frac{\partial\phi(\lambda(t))}{\partial\lambda_n^\rho})$, PE_n^3]	$1\oplus$
PE^7	Step 10	η	$\sigma(j)$ [R(σ)]	$\sigma(j+1)$ [R(σ)]	$1\otimes$

[§]$\otimes$: operation of a multiplication; $\oplus$: operation of an addition; ROM: operation of accessing data of a ROM.

$k = 1$, as shown in Fig. 3. It should be noted that the action of writing data into the registers occurs at the end of the clock pulse (i.e., at the positive edge of the next clock pulse) when write enable is active. For example, σ_j in R(σ) is updated to σ_{j+1} at the end of the clock pulse corresponding to both $k = K$ and $t = t_{\max}$.

2) Regarding the computations, we employ seven types of PE to carry out all the arithmetic operations required in the DPG algorithm. These PEs are named as PE_n^1, PE_n^2, PE_n^3, PE^4, PE^5, PE_n^6, and PE^7, where the superscript denotes the type of PE, the subscript denotes the index of the array, and the PE without any subscript means that it is single in the N PE arrays. Each type of PE consists of different hardware components that are needed for calculating a specific step in the DPG algorithm and yields the results that are needed in other step or steps. We will state the hardware components in a PE and its corresponding algorithmic step in the following. In the nth PE array, PE_n^1 performs Step 2) and outputs the computed $(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ to PE_n^2; its hardware components depend on the function $f_k(c)$. In addition, PE_n^1 consists of registers for storing $\alpha_{k,n}^2$ and $1/\alpha_{k,n}^2$ for all k and a register indicator k for choosing the corresponding register. PE_n^2 consists of six multipliers, four adders, and several comparators to perform Step 3) and output the computed $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ to PE_n^6, PE^4, and R$((\hat{r}_{k,n}, \hat{\rho}_{k,n}), k = 1, \ldots, K)$; PE_n^3 consisting of a register (for storing the constant $\hat{\beta}$), one adder, and one multiplier is designed to perform Step 8) and output the computed $\lambda_n^\rho(t+1)$ to R(λ_n^ρ); PE_n^6 consists of one adder to perform Step 5) and output the computed $\sum_{k=1}^{K} \hat{\rho}_{k,n} - 1$ to $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ and PE_n^3. The single PE^4 consists of $\log_2(N+1)$ adders to perform Step 4) and output the computed $\partial\phi(\lambda(t))/\partial\lambda_k^r$ to the single PE^5. In addition, PE^4 consists of registers for storing R_k for all k and a register indicator k to choose the corresponding register. The single PE^5 consisting of a register (for storing the constant $\hat{\beta}$), one adder, and one multiplier is designed to perform Step 6); PE^5 outputs the computed $\lambda_k^r(t+1)$ to R$(\lambda_1^r, \ldots, \lambda_K^r)$. The single PE^7 consists of a register for storing the constant η and a multiplier to perform Step 10); PE^7 outputs the computed $\sigma(j+1)$ to R(σ). We summarize the characteristics of these PEs in Table III to indicate the corresponding algorithmic step, embedded constant, input data [from], output data [to], and the computation complexity of a PE, which is shown in the last column and will be analyzed later.

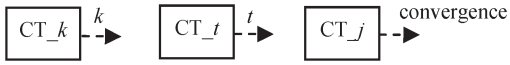

Fig. 4. Three counters.

3) There are three loops in the DPG algorithm, and the number of iterations in each loop has been set fixed. A branching will occur when completing the iterations of a loop, as described in Steps 7), 9), and 11). Therefore, we need three counters to count the number of iterations consumed in each loop to control the branching of the DPG algorithm.

The three counters are the k-counter, t-counter, and j-counter, which are denoted by CT_k, CT_t, and CT_j, respectively, and represented by the square blocks shown in Fig. 4. The values of CT_k and CT_t will be fed into the corresponding registers for proper operation. For example, the value k of CT_k will be used to indicate the iteration index of the innermost loop, i.e., Steps 1)–7), to point to the corresponding register bank in the type 2 registers. When $k = K$, there will be a branching at Step 7) such that the output data of PE_n^3, which performs Step 8),

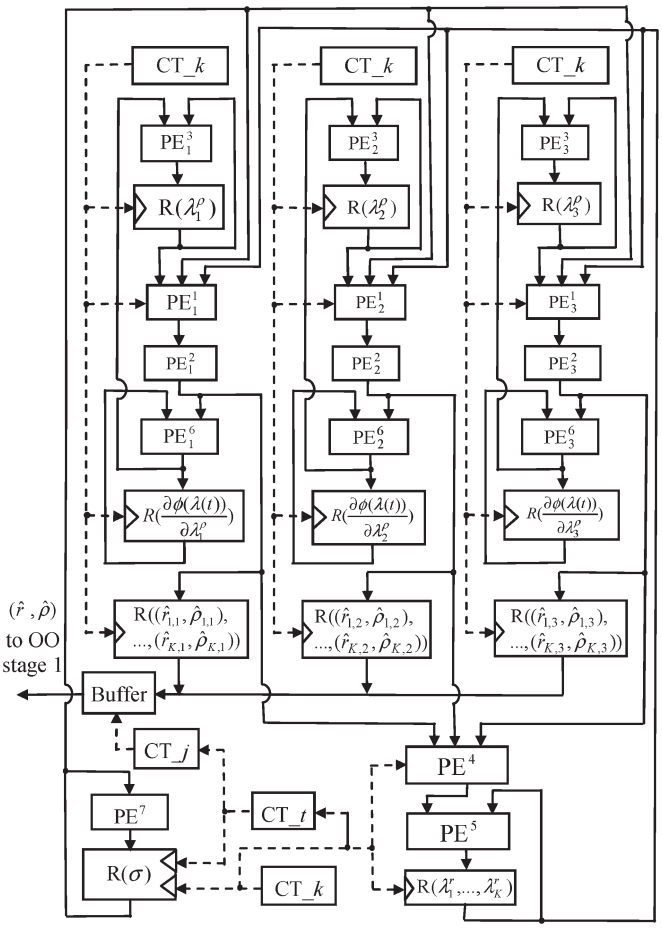

Fig. 5. Hardware architecture of the DPG algorithm.

will be written into the register R(λ_n^ρ). A similar reason applies to CT_t. When $t = t_{\max}$, there will be a branching at Step 9) such that the output data of PE^7, which performs Step 10), will be written into the register R(σ). Furthermore, when the value of CT_k changes from K to 1, the value of the register $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ will be reset to -1 to perform Step 1). The value of CT_j will be used to detect the convergence of the DPG algorithm. Thus, when $j = j_{\max}$, the DPG algorithm will be stopped and will output the approximate solution of (2), i.e., $(\hat{r}_n, \hat{\rho}_n)$, $k = 1, \ldots, K$, $n = 1, \ldots, N$.

The counters are designed such that CT_k will circulate from 1 to K and increase by 1 for every clock pulse, CT_t will circulate from 1 to $t_{\max}$ and increase by 1 for every K clock pulses, and CT_j will increase by 1 for every $K \cdot t_{\max}$ clock pulses.

4) Now, we are ready to interconnect the array PEs, registers, and counters to execute the DPG algorithm. From the columns of the input data [from] and the output data [to] in Table III, we can use solid lines with arrow heads to indicate the direction of the data flow to interconnect the PEs and registers, as shown in Fig. 5, in which we assume $N = 3$ and do not show the system clock for the sake of simplicity.

The three counters, namely CT_k, CT_t, and CT_j, as previously described, are used to count the number of iterations and control the branching of data flow. Therefore, to distinguish from the regular data flow, we use the dashed lines with arrow heads to indicate the flow of counter values to the corresponding registers, as shown in Fig. 5.

For the sake of simplicity, we will illustrate how the hardware architecture executes the DPG algorithm for one array as follows.

We initialize the values of registers $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ (Step 1: $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho) = -1$), R($\lambda_n^\rho$) (Step 0: $\lambda_n^\rho = 0$), R($\lambda_1^r, \ldots, \lambda_K^r$) (Step 0: $\lambda_k^r = 0,\ k = 1, \ldots, K$), R($\sigma$) (Step 0: $\sigma_0 = 1$), the counter values of CT_k (Step 1: $k = 1$), CT_t (Step 0: $t = 1$), CT_j (Step 0: $j = 1$), and command PE$_n^1$ to start execution. Then, PE$_n^1$ will perform Step 2) and output the resulting $(\widetilde{r}_{k,n}, \widetilde{\rho}_{k,n})$ to PE$_n^2$, as shown in Fig. 5. Then, PE$_n^2$ will perform Step 3) and output the resulting $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ to PE$_n^6$, PE^4, and R($(\hat{r}_{k,n}, \hat{\rho}_{k,n})$, $k = 1, \ldots, K$). It should be noted that the data $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$ will be written into the kth register bank of R($(\hat{r}_{k,n}, \hat{\rho}_{k,n})$, $k = 1, \ldots, K$), as selected by the value k of the counter CT_k, as shown in Fig. 5. Once PE^4 receives the computed $\hat{r}_{k,n}$, $n = 1, \ldots, N$ from the N PE$_n^2$s, it will perform Step 4) and output the value of $\partial\phi(\lambda(t))/\partial\lambda_k^r$ to the single PE^5. Then, PE^5 will perform Step 6) to update $\lambda_k^r(t)$, and the updated value will be sent to register banks R($\lambda_1^r, \ldots, \lambda_K^r$). In the meantime, PE$_n^6$ will perform Step 5) using the computed data $\hat{\rho}_{k,n}$ from PE$_n^2$ and the data $\sum_{l=1}^{k-1} \hat{\rho}_{l,n} - 1$ in $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ and will then output the resulting data $\sum_{l=1}^{k} \hat{\rho}_{l,n}$ to $R(\partial\phi(\lambda(t))/\partial\lambda_n^\rho)$ and PE$_n^3$, as shown in Fig. 5. The above calculations and data flow complete one iteration of the innermost loop, i.e., Steps 1)–7), and should be done in *one clock pulse*, whose period needs to be long enough such that the output data of all PEs can reach steady states. The counter CT_k will increase by 1 for each activation of a clock pulse (Step 7). As k increases by 1, the output of the register R($\lambda_1^r, \ldots, \lambda_K^r$) will be λ_{k+1}^r instead of λ_k^r, and hence, the next iteration of the innermost loop starts.

The above process will repeat, and a branching will occur when the value of CT_k reaches K. When $k = K$, the write enable of the register R(λ_n^ρ) will be active, and the output data of PE$_n^3$, which performs Step 8) with input data from PE$_n^6$ and R(λ_n^ρ), as shown in Fig. 5, will be written into R(λ_n^ρ). After this clock pulse, the value of CT_k will start from 1 again, and the value of CT_t will be increased by 1 (Step 9). As t increases by 1, the value of λ_n^ρ, $n = 1, \ldots, N$, as well as λ_k^r, $k = 1, \ldots, K$, has been updated, and hence, the next iteration of the middle loop starts.

The above process will repeat, and a branching will occur when the value of CT_t reaches $t_{\max}$ (Step 9), which will activate the write enable of R(σ), and the output data of PE^7, which performs Step 10 with input data from R(σ), as shown in Fig. 5, will be written into R(σ). In the meantime, the value of CT_k will start from 1 again, and CT_t also starts from 1, while the value of CT_j will be increased by 1 (Step 11). As j increases by 1, the above process starts all over with a new $\sigma(= \sigma_{j+1})$. This process will proceed until CT_j reaches $j_{\max}$, which implies that the DPG algorithm converges (Step 12), and then, as shown in Fig. 5, the buffer will be activated to output the data $(\hat{r}_{k,n}, \hat{\rho}_{k,n})$, $k = 1, \ldots, K$, $n = 1, \ldots, N$, the solution of (3) when $\sigma = \sigma_{j_{\max}}$, or the

approximate solution of (2). This completes the description of the execution of the DPG algorithm for one PE array.

In Fig. 5, we can see that the structure is very regular, modular, and locally interconnected; hence, it is hardware implementable.

Remark 12: By exploiting the merits of hardware computation and parallelism of the DPG algorithm, the computation time estimated based on the hardware architecture is almost independent of the N, but it is at the cost of large area when N is large. By taking $N = 128$ for example, we need 1666 multipliers in the hardware architecture, which is huge indeed. Manufacturing an integrated circuit with large gate counts is a challenging issue; however, it can be resolved due to the advancement of the semiconductor manufacturing technology.

APPENDIX C
COMPUTATION COMPLEXITY OF THE DPG ALGORITHM

As previously indicated, the clock period should be long enough such that the outputs of all PEs can reach steady states during an iteration. In other words, the clock period should be longer than the computation time of the *critical path*, i.e., the most time-consuming path of the DPG hardware architecture. To identify the critical path, we need to analyze the computation complexity of each PE first. The computation complexity of a PE can be directly derived from its corresponding arithmetic operations. For example, PE^4 performing Step 4) of the DPG algorithm requires $\log_2(N+1)$ stages of adders, and therefore, it takes $\log_2(N+1)\oplus$, where $\oplus$ denotes an arithmetic operation of addition. A similar reasoning applies to PE_n^2, PE_n^3, PE^5, PE_n^6, and PE^7. However, the hardware component of PE_n^1 depends on $f_k(c)$. A typical $f_k(c)$ can be $B \cdot (2^c - 1)$, where B is a constant, and we may then use six multipliers, three adders, and one ROM to perform Step 2), though the details of which are omitted here. We have reported the computation complexity of all PEs in the last column of Table III. The data propagation time between PEs and registers are negligible compared with addition or multiplication, and the action of writing data into or reading data from a register consumes time of no more than that of one addition. We let T_{clock} denote the to-be-designed clock period. Identifying the critical path in Fig. 5, we have

$$T_{\mathrm{clock}} = T_{\mathrm{PE}_n^1} + T_{\mathrm{PE}_n^2} + T_{\mathrm{PE}^4} + T_{\mathrm{PE}^5} + T_{\mathrm{PE}^7} + 2T_{\oplus} \quad (18)$$

where $T_{\mathrm{PE}_n^i}$ and T_{PE^j} denote the time complexity of executing PE_n^i and PE^j, respectively, and $2T_{\oplus}$ represents the time that is needed for writing data into and reading data from registers. Note that the action of writing data into the three registers in the critical path occurs simultaneously. Therefore, writing data into three registers only counts for one $T_{\oplus}$. A similar reasoning applies to reading data from these registers. Hence, the computation time of the proposed DPG algorithm is

$$K \cdot t_{\max} \cdot j_{\max} \cdot T_{\mathrm{clock}}. \quad (19)$$

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers who gave them constructive comments and helpful suggestions to improve the readability of this paper.

REFERENCES

[1] J. Blogh, P. Cherriman, and L. Hanzo, "Comparative study of adaptive beam-steering and adaptive modulation-assisted dynamic channel allocation algorithms," *IEEE Trans. Veh. Technol.*, vol. 50, no. 2, pp. 398–415, Mar. 2001.

[2] L. Xu, X. Shen, and J. W. Mark, "Fair resource allocation with guaranteed statistical QoS for multimedia traffic in wideband CDMA cellular network," *IEEE Trans. Mobile Comput.*, vol. 4, no. 2, pp. 166–177, Mar./Apr. 2005.

[3] C. Y. Wong, R. S. Cheng, K. B. Letaief, and R. D. Murch, "Multiuser OFDM with adaptive subcarrier, bit, and power allocation," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 10, pp. 1747–1758, Oct. 1999.

[4] W. Rhee and J. M. Cioffi, "Increase in capacity of multiuser OFDM system using dynamic subchannel allocation," in *Proc. IEEE VTC—Spring*, Tokyo, Japan, May 2000, vol. 2, pp. 1085–1089.

[5] I. Kim, H. L. Lee, B. Kim, and Y. H. Lee, "Use of linear programming for dynamic subcarrier and bit allocation in multiuser OFDM," *IEEE Trans. Veh. Technol.*, vol. 55, no. 4, pp. 1195–1207, Mar. 2006.

[6] M. Ergen, S. Coleri, and P. Varaiya, "QoS aware adaptive resource allocation techniques for fair scheduling in OFDMA based broadband wireless access systems," *IEEE Trans. Broadcast.*, vol. 49, no. 4, pp. 362–370, Dec. 2003.

[7] D. Kivanc, G. Li, and H. Liu, "Computationally efficient bandwidth allocation and power control for OFDMA," *IEEE Trans. Wireless Commun.*, vol. 2, no. 6, pp. 1150–1158, Oct. 2003.

[8] G. Zhang, "Subcarrier and bit allocation for real-time services in multiuser OFDM systems," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2004, vol. 5, pp. 2985–2989.

[9] Z. Han, Z. Ji, and K. J. R. Liu, "Low-complexity OFDMA channel allocation with Nash bargaining solution fairness," in *Proc. IEEE Global Telecommun. Conf.*, Dec. 2004, vol. 6, pp. 3726–3731.

[10] D. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. Reading, MA: Addison-Wesley, 1984.

[11] C. Lin and S. Lin, "A new dual-type method used in solving optimal power flow problems," *IEEE Trans. Power Syst.*, vol. 12, no. 4, pp. 1667–1675, Nov. 1997.

[12] S.-Y. Lin and C.-H. Lin, "A computationally efficient method for nonlinear multicommodity network flow problems," *Networks*, vol. 29, no. 4, pp. 225–244, Jul. 1997.

[13] Y. C. Ho, *Soft Optimization for Hard Problem*. Cambridge, MA: Harvard Univ. Press, 1996.

[14] Y. C. Ho, C. G. Cassandras, C. H. Chen, and L. Dai, "Ordinal optimization and simulation," *J. Oper. Res. Soc.*, vol. 51, no. 4, pp. 490–500, Apr. 2000.

[15] T. W. E. Lau and Y. C. Ho, "Universal alignment probabilities and subset selection for ordinal optimization," *J. Optim. Theory Appl.*, vol. 39, no. 3, pp. 455–489, Jun. 1997.

[16] S. K. Lai, R. S. Cheng, K. B. Letaief, and R. D. Murch, "Adaptive trellis coded MQAM and power optimization for OFDM transmission," in *Proc. IEEE VTC*, Houston, TX, May 1999, pp. 290–294.

[17] C. T. Lin and C. S. George Lee, *Neural Fuzzy System: A Neuro-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[18] M. T. Hagan and M. Menhaj, "Training feedforward networks with Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[19] S. Lin, Y. C. Ho, and C. Lin, "An ordinal optimization theory based algorithm for solving the optimal power flow problem with discrete control variables," *IEEE Trans. Power Syst.*, vol. 19, no. 1, pp. 276–286, Feb. 2004.

[20] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2002.

[21] S. K. Hsu, S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R. K. Krishnamurthy, and S. Y. Borkar, "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 256–264, Jan. 2006.

[22] R. Kanan, B. Hochet, M. Declercq, and A. Guyot, "A low-power high storage capacity structure for GaAs MESFET ROM," in *Proc. Int. Workshop Memory Technol. Des. Test.*, San Jose, CA, Aug. 1997, pp. 58–63.

[23] K. C. Chang, *Digital Systems Design With VHDL and Synthesis: An Integrated Approach*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1999.

[24] J. Chuang and N. Sollenberger, "Beyond 3G: Wideband wireless data access based on OFDM and dynamic packet assignment," *IEEE Commun. Mag.*, vol. 38, no. 7, pp. 78–87, Jul. 2000.

Shin-Yeu Lin was born in Taiwan, R.O.C. He received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, the M.S. degree in electrical engineering from the University of Texas, El Paso, and the D.Sc. degree in systems science and mathematics from Washington University, St. Louis, MO, in 1975, 1979, and 1983, respectively.

From 1984 to 1985, he was with Washington University, working first as a Research Associate and then as a Visiting Assistant Professor. From 1985 to 1986, he was with GTE Laboratories, working as a Senior Member of Technical Staff. In 1987, he joined the Department of Electrical and Control Engineering, National Chiao Tung University, where he has been a Professor since 1992. His major research interests include wireless communication and networks, data mining, ordinal optimization theory and applications, and distributed computations.

Jung-Shou Huang was born in Taiwan, R.O.C. He received the B.S. degree in electrical engineering in 1996 from Tamkang University, Taipei, Taiwan, and the M.S. degree in electrical and control engineering in 1998 from National Chiao Tung University, Hsinchu, Taiwan, where he is currently working toward the Ph.D. degree in electrical and control engineering.

He is also a Digital IC Designer with Elan Microelectronics Corporation, Hsinchu, Taiwan, R.O.C. His major research interests include optimization theory with applications, image processing, and digital IC design.