Fast Approximation For Peak Power Driven Voltage Partitioning in Almost Linear Time

Jia Wang, Xiaodao Chen, Lin Liu, Shiyan Hu
Department of Electrical and Computer Engineering
Michigan Technological University
Houghton, Michigan 49931
Email: {jiaw, cxiaodao, lliu7, shiyan}@mtu.edu.

***Abstract*—Voltage partitioning on functional units/blocks targeting peak power minimization has been demonstrated to be effective for energy reduction considering voltage island shutdown impact. However, the existing technique can only solve this NP-hard problem efficiently on small designs. In this paper, a much faster linear time approximation scheme is proposed, which can approximate the optimal voltage partitioning solution within a factor $1+\epsilon$, for any $0 < \epsilon < 1$, and runs in $O(n + 1/\epsilon^{O(1)})$ time, where n is the number of functional units. There are multiple ingredients in such a surprisingly low time complexity algorithm. It first categorizes all the functional units into big functional units and small functional units using an ϵ related threshold. Subsequently, a rounding based dynamic programming procedure is performed to handle big functional units and a linear programming based algorithm is performed to handle small functional units, which is followed by the discretization of the continuous linear programming solution. Moreover, through the exploration of the unique nature of our problem, a greedy algorithm is proposed to optimally solve the linear programming formulation in a combinatorial fashion. Further, since patching a salient partitioning solution of big functional units with that of small functional units could lead to a much worse combined solution, a highly efficient enumeration process running in time independent of n is proposed. The experimental results demonstrate that our algorithm runs very fast. It needs only 0.3 second to partition a testcase with 5000 functional units which is more than $10000\times$ faster than the existing algorithm while still reducing the peak power by 7.4%.**

Keywords: Linear Time Approximation Scheme, Voltage Partitioning, Peak Power, Energy Efficiency.

I. Introduction

Power has been a critical factor in limiting the VLSI circuit scaling. Dynamic power consumption, which is typically estimated as the multiplication of the capacitance and the squared voltage, still contributes significantly to the total power consumption. For example, the work [1] shows that 60% of total power is due to dynamic power in a recent chip designed by Intel. Various optimization techniques such as multiple supply voltage and voltage island have been successfully deployed in practical chip design for effective dynamic power reduction.

In the floorplanning problem, some functional units (also known as blocks or modules) are to be placed. Quite often, their voltage levels are determined before floorplanning using a procedure called voltage partitioning. After voltage partitioning, functional units with the same voltage levels can be grouped to form some voltage islands during floorplanning [2], [3], [4]. An important usage of formed voltage islands is that during the runtime, some of the voltage islands can be shut down if they are idle so as to reduce the dynamic power consumption [3]. Clearly, voltage partitioning enables the voltage island generation and impacts the runtime voltage island shutdown for energy reduction [8].

There are a few works focused on voltage partitioning such as [3], [4], [5], [7], [8]. Most of them consider the total power minimization over all partitions. However, as demonstrated in [8], these solutions are not effective in reducing the energy consumption considering the voltage island shutdown effect, which motivates it [8] to design a peak power driven voltage partitioning technique. Although [8] designs a provable good polynomial time approximation scheme to effectively solve the peak power driven voltage partitioning problem (which is shown to be NP-complete in [8]), the main issue is that it runs slow in practice. Given n functional units and k partitions, their technique runs in $O(k^2(\frac{n^2}{\epsilon^4})^k)$ time, where ϵ is the target approximation ratio and $0 < \epsilon < 1$. This means that it runs in at least $\Omega(n^{2k})$ time. This is why in their experiments, [8] has to use the solution restriction technique, which aggressively prunes the solutions even if they are not redundant, for handling even moderate n (e.g., $n = 500$ and $k = 2$). As a result of the solution restriction, the solution quality of their technique degrades significantly and the target approximation ratio cannot be achieved. This is why our algorithm, which runs fast and does not need solution restriction, can generate the solution with on average 7.4% better peak power with the same ϵ.

This paper designs a new peak power driven voltage partitioning technique, which runs much faster than [8] while still with the theoretical guarantee in approximation ratio. Recall that a voltage partitioning algorithm is a fully polynomial time approximation scheme (FPTAS) to the NP-hard voltage partitioning problem if it can compute a voltage partitioning solution with peak power always within a $1+\epsilon$ factor of the optimal voltage partitioning solution and it runs in time polynomial in n and $1/\epsilon$ for any $\epsilon > 0$. In particular, it is a linear time approximation scheme if it runs in time linear in n (while usually nonlinear in $1/\epsilon$). Linear time approximation scheme is in fact the fastest possible $1+\epsilon$ approximation algorithm for any NP-hard problem in theory. Motivated from [9] which solves a knapsack problem, this paper proposes a linear time approximation scheme for solving our peak power driven voltage partitioning problem. The main contribution of this paper is summarized as follows.

- A linear time approximation scheme is proposed to approximate the optimal voltage partitioning solution within a factor of $1+\epsilon$ running in $O(n+1/\epsilon^{O(1)})$ time for any $0 < \epsilon < 1$, where n is the number of functional units.
- There are various advanced optimization techniques developed in our algorithm. It first categorizes all the functional units into big functional units and small functional units, followed by a rounding based dynamic programming procedure to handle big functional units and a linear programming based algorithm to handle small functional units. After that, a discretization process of the best continuous linear programming solution is performed. All of the above components have ϵ approximation error guarantee. Moreover, due to the unique nature of our problem, a greedy algorithm can be designed to optimally solve the linear programming formulation in a combinatorial fashion without calling the generic linear programming solver. Further, a highly efficient enumeration process running in time independent of n is proposed to integrate with the above optimization for handling the interaction between big functional units and small functional units.
- The new algorithm is very efficient in practice. Our experimental results show that our algorithm needs only 0.3 second to partition a testcase with 5000 functional units into 5 partitions which is more than $10000\times$ faster than the previous work [8]. This huge speedup is due to the pure linear time dependance on n of our algorithm in contrast to the $\Omega(n^{10})$ time complexity of the previous work.
- On average, the new algorithm reduces the peak power by 7.4% compared to the previous work [8] due to the fact that the previous work needs the solution restriction technique for acceleration which loses the theoretical guarantee on approximation ratio.
- To demonstrate the usage of our algorithm in voltage island shutdown for energy reduction, compared to a natural greedy

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

TABLE I
AN EXAMPLE FOR PEAK POWER DRIVEN VOLTAGE PARTITIONING.

Functional units	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
Capacitance	7	3	5	2	4	2	7	10	6
Voltage	0.8	1.0	1.0	1.2	1.0	0.8	1.5	1.0	1.0

frequency based voltage partitioning algorithm our peak power driven voltage partitioning technique can lead to about 25% more energy reduction.

II. PROBLEM FORMULATION

Given a set F of n functional units, $F = \{u_1, ..., u_n\}$, each of which has a voltage $v(u_i)$ level from a given technology library and a capacitance $c(u_i)$. Let V denote the technology library which consists of m ordered voltage levels, namely, $V = \{v_1, v_2, \ldots, v_m\}$ and $v_1 < v_2 \ldots < v_m$. It is well known that in a practical technology library, there can be very few (e.g., 5) voltage levels [6], [2]. Let $v_{min} = v_1$ denote the lowest voltage level and $v_{max} = v_m$ denote the highest voltage level. Note that due to the timing constraint on a functional unit u_i, it can only be assigned with a voltage no lower than some available voltage level (i.e., $v(u_i)$) in the given technology library. Such a minimum allowed voltage $v(u_i)$, which is part of the input to our problem, can be computed using the timing-power tradeoff curve of the functional unit u_i as indicated in [3], [4].

The power of a set of functional units is computed as the multiplication of the sum of capacitances and the square of the maximum voltage over these functional units. F is to be partitioned into k pair-wise disjoint partitions, denoted by $F_1, F_2, \ldots, F_k$, such that the maximum power over all partitions (or *peak power* in short) is minimized. In practice, the values of m, k, v_{max}, v_{min} and v_{max}/v_{min} are all quite small. They can be treated as constant numbers.

In this paper, for the convenience of illustration we generalize the operators $p(\cdot)$, $v(\cdot)$ and $c(\cdot)$. Precisely, $p(\gamma)$ computes the power when γ is a partition, and computes peak power when γ is a set of partitions (i.e., a partitioning solution). $v(\gamma)$ computes the voltage when γ is a functional unit, and computes maximum voltage when γ is a partition (i.e., a set of functional units). $c(\gamma)$ computes the capacitance when γ is a functional unit, and computes the sum of capacitances when γ is a partition. Let s denote a voltage partitioning solution. Thus, $F(s)$ consists of k subsets, namely, $F_1(s), F_2(s), \ldots, F_k(s)$. The power of $F_i(s)$ is given by $p(F_i(s))$ and is computed as $p(F_i(s)) = \sum_{u \in F_i(s)} c(u) \cdot v^2(F_i(s))$, where $v(F_i(s)) = \max_{u \in F_i(s)} v(u)$. Subsequently, the peak power (the maximum power over all partitions) is given by $p(F(s))$ and is computed as $p(F(s)) = \max_i p(F_i(s))$. Our peak power driven voltage partitioning problem is the same as [8] which is shown to be NP-complete.

Peak Power Driven Voltage Partitioning Problem: Given a set F of n functional units, each of which has a capacitance and a voltage level from a given voltage library consisting of m voltages, and an integer k, to compute a voltage partitioning solution s with partitions $\{F_1(s), F_2(s), ..., F_k(s)\}$ such that $\bigcup_{i=1}^{k} F_i(s) = F, F_i(s) \bigcap_{\forall i \neq j} F_j(s) = \emptyset, \forall 1 \leq i, j \leq k$, and the maximum power of all the $F_i(s)$ is minimized.

III. THE ALGORITHM

A. Overall flow

Our algorithm begins with grouping all the functional units into two categories. The first category contains the big functional units, denoted by F_{big}, where each functional unit has capacitance greater than or equal to a threshold. The second category contains the small functional units, denoted by F_{small}, where each functional unit has capacitance smaller than the threshold.

The two categories of functional units are handled differently in our FPTAS algorithm. The big functional units will significantly impact the peak power of the voltage partitioning solution due to their large capacitances. Thus, a relatively time consuming dynamic programming technique will be applied to compute the voltage partitioning solution on them. Fortunately, we will show that there can be only few large functional units if one defines the threshold nicely. In contrast, small functional units can only slightly impact the quality of the voltage partitioning solution from the perspective of their small capacitances. Thus, a relatively computationally efficient algorithm will be applied to handle them. Such an efficient algorithm is based on forming the voltage partitioning problem as the relaxed linear programming problem followed by the solution discretization. Discretizing a linear programming solution (a continuous solution) is a common idea in VLSI physical design, but it is usually without theoretical guarantee. In contrast, through exploring the unique nature of our problem, our discretization process always has error limited by ϵ. Moreover, since the formulated linear program is a quite specific one, we propose a greedy algorithm to optimally solve it in a combinatorial fashion, which is more computationally efficient than the generic linear programming solver. Refer to Figure 1 for the flow chart of our algorithm.

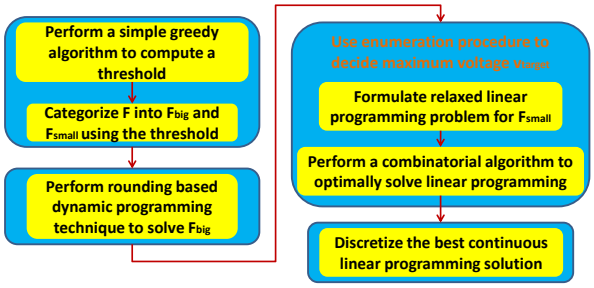

Fig. 1. Flow chart for the algorithm.

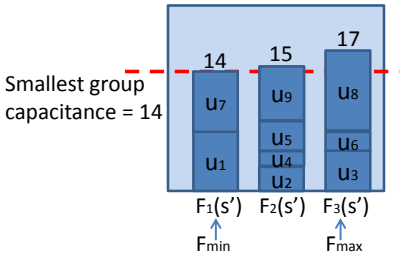

Fig. 2. An example of greedy algorithm to determine the threshold.

B. Determining the threshold

Before describing the algorithm, assume that the target approximation ratio ϵ satisfies $0 < \epsilon < 1$. This is a valid assumption since one is always interested in computing a solution close to the optimal solution.

The first step is to determine the threshold. For this, we design a simple greedy algorithm to partition all the functional units in F, whose solution will be used in setting the threshold. Let s' denote the voltage partitioning solution from the greedy algorithm. Our greedy algorithm proceeds as follows. Initially, all of $F_1(s'), F_2(s'), \ldots, F_k(s')$ are empty. Recall that $c(F_i(s'))$ computes the sum of capacitances of all the functional units in $F_i(s')$ for any i. We call it *group capacitance* for group/partition i. The functional units are ordered in arbitrary order. Functional units will be processed one by one. Each time, a functional unit is put into the partition with the smallest group capacitance. When there is a tie, it can be put into any group among the tie.

For the first functional unit u_1, it will be put into the partition with smallest group capacitance $c(F_i)$. At this moment, all of the group capacitances are zero. Thus, u_1 can be put into any group. Suppose that it is assigned to $F_1(s')$ and we will update $c(F_1(s'))$ to be $c(u_1)$. One similarly handles all the other functional units. This greedy algorithm runs in linear time in terms of n.

In the resulting voltage partitioning solution, denote by F_{min} the group with the smallest group capacitance and denote by F_{max} the group with the largest group capacitance. When the above greedy algorithm is applied to the example in Table I, we can obtain the solution as shown in Figure 2 where F_{min} is $F_1(s')$ and F_{max} is $F_3(s')$. Recall that v_{min} is the lowest voltage level over all functional units and v_{max} is the highest voltage level over all functional units. Let c_{max} denote the largest capacitance over all functional units. Let OPT denote the optimal voltage partitioning solution over all the functional units. One easily sees that $p(OPT) \leq c(F_{max}) \cdot v_{max}^2$. This is because that we already compute a voltage partitioning solution using our greedy algorithm and this solution has the largest group capacitance as $c(F_{max})$ and in the worst case this group has the voltage v_{max}. Similarly, $c(F_{min}) \cdot v_{min}^2 \leq p(OPT)$. The reason is as follows. We claim that $c(F_{min})$ gives the lowest possible peak

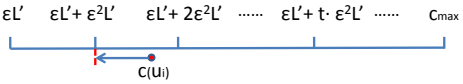

Fig. 3. Round the capacitances of big functional units, assuming that $0 < \epsilon < 1$.

capacitance (i.e., the lowest possible maximum group capacitance over all partitions) for any voltage partitioning solution. Note that the total capacitance (i.e., the sum of capacitances of all functional units) is always the same for any voltage partitioning solution. If a voltage partitioning solution can have largest group capacitance lower than $c(F_{min})$, it means that the total capacitance of that solution is lower than the total capacitance of our greedy solution which is not possible. Thus, any voltage partitioning solution (including the optimal solution) has peak group capacitance at least $c(F_{min})$. Together with the fact that the lowest voltage can be v_{min}, the peak power for any voltage partitioning solution is lower bounded by $c(F_{min}) \cdot v_{min}^2$.

In our greedy solution, we claim that $c(F_{min}) \geq c(F_{max}) - c_{max}$. Suppose to the contrary that $c(F_{min}) < c(F_{max}) - c_{max}$. Among the functional units added into F_{max} during our greedy algorithm, let u' denote the last such functional unit. For example, u' is u_8 Figure 2. According to our greedy algorithm, u' is added into F_{max} since at the moment of handling u', F_{max} has the smallest group capacitance. However, $c(F_{min}) < c(F_{max}) - c_{max}$ means that F_{min} has smaller group capacitance than F_{max} at that moment since $c(u') \leq c_{max}$. Thus, u' would be added to F_{min} but not F_{max}. This contradicts the assumption that u' is in F_{max}. Therefore, $c(F_{min}) \geq c(F_{max}) - c_{max}$.

Denote by U (resp. L) to be the upper (resp. lower) bound on $p(OPT)$ which is the peak power of the optimal voltage partitioning solution. Note that $c_{max} \cdot v_{min}^2 \leq p(OPT)$ since the functional unit with largest capacitance has to be put in a partition and this partition has the voltage at least v_{min}. Therefore, one can set the upper bound as

$$U = c(F_{max}) \cdot v_{max}^2 \tag{1}$$

and lower bound as

$$L = \max\{c(F_{min}) \cdot v_{min}^2, c_{max} \cdot v_{min}^2\}. \tag{2}$$

The ratio between U and L can be bounded as follows. First,

$$c(F_{min}) \cdot v_{min}^2 \leq p(OPT) \leq c(F_{max}) \cdot v_{max}^2. \tag{3}$$

Since the above shows that $c(F_{max}) \leq c(F_{min}) + c_{max}$, one has

$$c(F_{max}) \cdot v_{max}^2 \leq (c(F_{min}) + c_{max}) \cdot v_{max}^2. \tag{4}$$

Since $c(F_{min}) \cdot v_{min}^2 \leq L$ and $c_{max} \cdot v_{min}^2 \leq L$, one has

$$c(F_{max}) \cdot v_{max}^2 \leq \frac{v_{max}^2}{v_{min}^2}[c(F_{min}) \cdot v_{min}^2 + c_{max} \cdot v_{min}^2] \leq \frac{v_{max}^2}{v_{min}^2} \cdot 2L. \tag{5}$$

This means that

$$\frac{U}{L} \leq \frac{2v_{max}^2}{v_{min}^2}. \tag{6}$$

We reach the following lemma.

Lemma 1: There exist a lower bound L and an upper bound U on the peak power of the optimal voltage partitioning solution such that $\frac{U}{L} \leq \frac{2v_{max}^2}{v_{min}^2}$.

Set $L' = \frac{L}{v_{max}^2}$. Our threshold is $\epsilon L'$ where ϵ is the target approximation ratio given by the user. We will group all the functional units with capacitance greater than or equal to $\epsilon L'$ to F_{big} and the remaining functional units to F_{small}.

C. Dynamic programming for big functional units

We first show how many functional units can be in F_{big}. Since each functional unit there has the capacitance at least $\frac{\epsilon L}{v_{max}^2}$, each functional unit has power at least $\frac{\epsilon L v_{min}^2}{v_{max}^2}$. There can be at most $\frac{2kv_{max}^4}{\epsilon v_{min}^4}$ of them. That is,

$$|F_{big}| \leq \frac{2kv_{max}^4}{\epsilon v_{min}^4} \tag{7}$$

Otherwise, the total power (the sum of power over all partitions) is greater than

$$\frac{\epsilon L v_{min}^2}{v_{max}^2} \cdot \frac{2kv_{max}^4}{\epsilon v_{min}^4} = \frac{2kLv_{max}^2}{v_{min}^2}, \tag{8}$$

which is obtained assuming that all the functional units have the voltage level v_{min}. In any partitioning solution on those functional units, there exists at least one partition with power greater than

$$\frac{1}{k} \cdot \frac{2kLv_{max}^2}{v_{min}^2} = \frac{2Lv_{max}^2}{v_{min}^2}. \tag{9}$$

However, $\frac{2Lv_{max}^2}{v_{min}^2}$ is the upper bound U on the optimal solution as shown in Equation (6). This means that if one has more than $\frac{2kv_{max}^4}{\epsilon v_{min}^4}$ functional units, there cannot be any partitioning solution with peak power smaller than or equal to U. Otherwise, it contradicts the fact that U is a valid upper bound. Since $\frac{v_{max}}{v_{min}}$ is assumed to be a constant in Equation (7), one has $O(\frac{k}{\epsilon})$ number of functional units in F_{big}. This value seems large when ϵ is small. However, from our experiments we see that even when ϵ is set to 5%, our experiments on the testcases from [8] show that there are quite few functional units in F_{big}. For example, when $n = 100, k = 2$, there are only 4 functional units in F_{big}.

Before introducing the dynamic programming algorithm on performing the voltage partitioning for F_{big}, we first describe a rounding process. It can significantly reduce the number of distinct capacitances over all functional units in F_{big} with the theoretical guarantee on approximation error. Precisely, we round down the capacitance of each functional unit in F_{big} to the nearest value in the form of $\epsilon L' + t \cdot \epsilon^2 L'$ where t is any nonnegative integer. For example, if a capacitance c is in the range of $[\epsilon L' + \epsilon^2 L', \epsilon L' + 2\epsilon^2 L']$, it will be rounded down to $\epsilon L' + \epsilon^2 L'$. Refer to Figure 3 for an illustration and recall that we assume $0 < \epsilon < 1$.

After this rounding procedure, each capacitance is in the form of $\epsilon L' + t \cdot \epsilon^2 L'$ for some t. According to Equation (2), $c_{max} \leq \frac{L}{v_{min}^2} = \frac{L' v_{max}^2}{v_{min}^2}$. We claim that there can be at most $\frac{L' v_{max}^2}{v_{min}^2 \cdot \epsilon^2 L'} = \frac{v_{max}^2}{\epsilon^2 \cdot v_{min}^2}$ distinct capacitances after rounding. To see this, one can view the rounding procedure as that after rounding every capacitance is always a multiple of $\epsilon^2 L'$ while the capacitances below $\epsilon L'$ are removed. The above bound computes the maximum number of distinct capacitances even if one does not remove those below $\epsilon L'$. It is also clear that since $c_{max} \leq \frac{L' v_{max}^2}{v_{min}^2}$, the integer t is upper bounded by $\frac{v_{max}^2}{\epsilon^2 \cdot v_{min}^2} = O(\frac{1}{\epsilon^2})$.

We are to bound the number of distinct sums of capacitances of F_{big}. After rounding, the sum of capacitances of any number of functional units from F_{big} will be in the form of $t_1 \cdot \epsilon L' + t_2 \cdot \epsilon^2 L'$ for some t_1, t_2. Since there are only $O(\frac{k}{\epsilon})$ number of functional units in F_{big}, $t_1 \leq O(\frac{k}{\epsilon})$. Since the maximum t is bounded by $O(\frac{1}{\epsilon^2})$, the maximum t_2 is bounded by $O(\frac{k}{\epsilon}) \cdot O(\frac{1}{\epsilon^2}) = O(\frac{k}{\epsilon^3})$. Since t_1 and t_2 are nonnegative integers within the above bounds, the number of distinct sums of capacitances is bounded by

$$O(\frac{k}{\epsilon}) \cdot O(\frac{k}{\epsilon^3}) = O(\frac{k^2}{\epsilon^4}). \tag{10}$$

The analysis again presents the worst case bound. From our experiments, since the actual number of functional units in F_{big} is much smaller than $O(\frac{k}{\epsilon})$, the number of distinct sums of capacitances is much smaller than the above bound as well. We reach the following lemma.

Lemma 2: There are $O(\frac{k^2}{\epsilon^4})$ number of distinct sums of capacitances in F_{big} after rounding.

Note that the previous work [8] also rounds the capacitance but their rounding is much more straightforward and less efficient. After rounding, we will use the dynamic programming algorithm originally proposed in [8] to partition the functional units in F_{big}. For completeness, we briefly overview the algorithm as follows. In the dynamic programming algorithm, the functional units will be processed one by one. When processing one functional unit, it is assigned to each of k possible partitions. This means that if there are w number of solutions

before processing this functional unit, after processing it there will be kw solutions since k solutions will be generated from each of the original solution. If one performs this process to all the functional units in a straightforward fashion, there will be exponential number of solutions. For acceleration, the work [8] characterizes each partial voltage partitioning solution, denoted by s, as follows. s consists of a set of k partitions $\{F_1(s), F_2(s), \ldots, F_k(s)\}$, and it is characterized by a $2k$-tuple $(v(F_1(s)), c(F_1(s)), \ldots, \ldots, v(F_k(s)), c(F_k(s))$ (recall that $v(F_i(s))$ and $c(F_i(s))$ compute the maximum voltage and sum of capacitance over all the functional units in $F_i(s)$, respectively). Given two solutions s_1 and s_2, s_1 is redundant with respect to s_2 and thus can be pruned if $v(F_i(s_1)) = v(F_i(s_2))$ and $c(F_i(s_1)) = c(F_i(s_2))$ for every $1 \leq i \leq k$. [8] shows that this pruning will not impact the optimality since any further partitioning solution on the remaining functional units derived from s_1 can be applied to s_2, which will not lead to larger peak power. It [8] also proves that the runtime of the algorithm is proportional to the number of non-redundant solutions. Precisely, in the partition i the number of combinations on $v(F_i(s))$ and $c(F_i(s))$ is given by the multiplication of the number of distinct sum of capacitances and the number of distinct maximum voltage. In our context, the number of distinct maximum voltage is easily bounded to be m while the number of distinct sum of capacitances is bounded by $O(\frac{k^2}{\epsilon^4})$ according to Lemma 2. Thus, it is $O(\frac{mk^2}{\epsilon^4})$ per partition. Subsequently, there can be totally $O((\frac{mk^2}{\epsilon^4})^k)$ non-redundant solutions over all k partitions. Including the runtime for performing the pruning using a radix-sort style procedure as in [8], the total runtime for dynamic programming is $O(k^2 \cdot (\frac{mk^2}{\epsilon^4})^k)$. We reach the following lemma.

Lemma 3: The dynamic programming algorithm on F_{big} can return $O((\frac{mk^2}{\epsilon^4})^k)$ solutions and run in $O(k^2 \cdot (\frac{mk^2}{\epsilon^4})^k)$ time.

Note that this runtime is independent of n. The output of the dynamic programming algorithm on F_{big} is a set of partial candidate voltage partitioning solutions, where none of them is redundant with respect to each other since the redundant solution has been pruned. The candidate voltage partitioning solution is called partial since the solution only contains the partitioning on big functional units.

We are to claim that for any voltage partitioning solution on F_{big}, there is always a voltage partitioning solution with small power increase at *every partition* returned by our dynamic programming. Given any voltage partitioning solution on F_{big}, one can always modify the capacitances to be rounded capacitances using our rounding process while keeping the assignment of each functional unit. This capacitance modified solution, denoted by s', can be either returned by our dynamic programming or pruned during the dynamic programming. For the former, there is no power increase. For the latter, one can find the solution which prunes it. If it is also pruned later, one finds the one pruning it. In this way, one can always trace the solution in the end of the pruning chain, which is returned by the dynamic programming. We are to investigate how much power increase there can be between this solution and s'.

First note that the main reason for the rounding procedure to accelerate the dynamic programming is that it can aggressively prune solutions. For example, suppose that $m = 1$ and $k = 2$ (i.e., two partitions and one voltage level). Consider two solutions s_1 and s_2. Suppose that before rounding, $c(F_1(s_1)) = 1.2, c(F_2(s_1)) = 2.7$ and $c(F_1(s_2)) = 1.1, c(F_2(s_2)) = 2.8$. Clearly, they are not redundant. However, suppose that after rounding, $c(F_1(s_1)) = 1, c(F_2(s_1)) = 2.5$ and $c(F_1(s_2)) = 1, c(F_2(s_2)) = 2.5$. One of them is pruned which could be s_1. Approximation error is introduced when s_1 is actually part of the optimal solution (and its peak power is determined by the second partition). Suppose that s_2 is returned by the dynamic programming, then the capacitance of its second partition is actually larger than that of s_1, which means that the peak power of s_2 is larger than that of s_1. Fortunately, it can only be slightly larger, which is controlled by the rounding step $\epsilon^2 L'$. From the above description, it is clear that one needs to bound the maximum rounding error on capacitance and then the maximum power increase can be computed as multiplying it by v_{max}^2. More importantly, since which partition has the peak power in the optimal solution is not known, one needs to guarantee that the power increase at every partition is always bounded by a small amount.

The capacitance of each functional unit has rounding error at most $\epsilon^2 L'$ in our rounding down process. At any partition, since we have only $\frac{2kv_{max}^4}{\epsilon v_{min}^4}$ functional units in F_{big} according to Equation (7), the total rounding error is at most $\epsilon^2 L' \cdot \frac{2kv_{max}^4}{\epsilon v_{min}^4} = \frac{2k\epsilon L v_{max}^2}{v_{min}^4} \leq \frac{2kv_{max}^2}{v_{min}^4} \cdot \epsilon p(OPT)$. Subsequently, the maximum power increase at any partition is bounded by $\frac{2kv_{max}^4}{v_{min}^4} \cdot \epsilon p(OPT)$. We reach the following lemma.

Lemma 4: Given any voltage partitioning solution on big functional units, there exists a dynamic programming solution with at most $\frac{2kv_{max}^4}{v_{min}^4} \cdot \epsilon p(OPT)$ power increase in every partition, for any $0 < \epsilon < 1$.

TABLE II
AN EXAMPLE FOR COMBINING SOLUTIONS OF F_{big} AND F_{small}

Functional units	F_{big}		F_{small}		
	u_1	u_2	u_3	u_4	u_5
Capacitance	220	100	22	25	2
Voltage	1.0	1.5	1.0	1.5	1.5

D. Target voltage enumeration for linking the solutions on F_{big} and F_{small}

The output of the dynamic programming algorithm on big functional units is a set of partial candidate voltage partitioning solutions. We are to grow each partial voltage partitioning solution to be the complete voltage partitioning solution. That is, starting from each partial solution s, assign each functional unit in F_{small} to a partition in s. This will be accomplished in two steps. The first step is to approximately assign the small functional units to partitions through linear programming, which computes some continuous solutions. The second step is to discretize the best continuous solution (i.e., the one with the minimum peak power) to compute the final voltage partitioning solution.

Given a partial candidate voltage partitioning solution s, adding small functional units into s could increase the maximum voltage $v(F_i(s))$ at some partitions, which will significantly increase the peak power and thus make the approximation ratio difficult to control. In other words, *even if one computes the optimal solution for F_{big} and the optimal solution for F_{small}, patching these two solutions together could still lead to much worse result*. One can use the example shown in Table II to see this.

To tackle this difficulty, we propose a *maximum voltage enumeration procedure* on determining $v(F_i(s))$ for each partition i before assigning F_{small}. The enumeration procedure will produce all possible combinations on the maximum voltages for all partitions. Given each enumeration result, the maximum voltage of each partition is determined and we call such voltage *target voltage* of that partition. The enumeration procedure can be performed efficiently and run in time independent of n. Since there are only m available voltage levels, each of the k partitions can have at most $m + 1$ choices of voltage levels. The last one accounts for the case that there is no functional unit in a partition and thus its maximum voltage is zero. Totally there can be up to $(m + 1)^k$ combinations of the maximum voltages for all partitions. Of course, the actual number could be smaller since some enumeration results are not feasible. For example, suppose that there are four available voltage levels, namely, $\{1, 1.1, 1.2, 1.5\}$. If after handling F_{big}, $F_1(s)$ already has the maximum voltage of 1.2, then after handling F_{small}, its maximum voltage can only be 1.2 or 1.5. It means that the target voltage for $F_1(s)$ cannot be from $\{1, 1.1\}$.

Given a partial solution s (which is a solution from dynamic programming), for each feasible enumeration result, the target voltage of each partition in s is determined. The following procedure will be performed to assign the functional units of F_{small} to s such that the peak power is minimized subject to the constraint that the target voltage on each partition is achieved. Let $v_{target}(F_i(s))$ denote the target voltage for the partition i in s. We first decompose F_{small} into m sub-categories such that each sub-category, denoted by $F_{small,j}$, contains all the small functional units with voltage level v_j, $1 \leq j \leq m$. As before, let $c(F_{small,j})$ denote the sum of capacitances of all functional units in $F_{small,j}$ and $v(F_{small,j})$ denote the maximum voltage of all functional units in $F_{small,j}$. Recall that $v_1 < v_2 \ldots < v_m$. Thus,

$$v(F_{small,1}) < v(F_{small,2}) \ldots < v(F_{small,m}). \quad (11)$$

For the convenience of illustration, without of loss of generality we assume that $\{F_1(s), F_2(s), \ldots, F_k(s)\}$ are ordered such that

$$v_{target}(F_1(s)) \leq v_{target}(F_2(s)) \ldots \leq v_{target}(F_k(s)). \quad (12)$$

The ordered partitions are illustrated in Figure 4. The key observation is that *the functional units in $F_{small,j}$ can only be put into the partitions with target voltage greater than or equal to v_j*. Otherwise, the target voltage is not achievable. This is in fact the benefit of our enumeration procedure. To obtain the observation, for example a functional unit with voltage level v_5 (i.e., it is in $F_{small,5}$) is assigned to a partition with target voltage level v_4. This partition will then have the maximum voltage level at least v_5 since $v_5 > v_4$, making its target voltage level not achievable.

Given a $v(F_{small,j})$, let $v_{target}(F_{w_j}(s))$ denote the first voltage greater than or equal to $v(F_{small,j})$ in the ordered list shown in Equation (12). This means that the functional units in $F_{small,j}$ can only be put into the partitions in the range of $\{F_{w_j}(s), F_{w_j+1}(s), \ldots, F_k(s)\}$. For the example in Figure 4, suppose that $V = \{1.0, 1.2, 1.4, 1.5\}$, then $w_4 = 3$ since $F_{small,4}$ (the small functional units with voltage $v_4 = 1.5$) can only be assigned to $\{F_3(s), F_4(s)\}$. Otherwise, assigning $F_{small,4}$ to $F_1(s)$ and $F_2(s)$ will make their target voltages not achievable. Similarly, $w_3 = 3$, $w_2 = 2$ and $w_1 = 1$. The index w_j is a critical parameter in our linear programming.

E. Linear programming for small functional units

Based on the above key observation, one can construct the following linear programming formulation which computes a continuous fractional voltage partitioning solution on F_{small}. One can view the continuous solution as splitting some functional units into pieces and assigning them into different partitions. Let c_j denote $c(F_{small,j})$, and let $c_{j,z}$ denote the amount of capacitances assigned from c_j to the partition z of a partial voltage partitioning solution s. Clearly, $w_j \leq z \leq k$. We also have $\sum_{z=w_j}^{k} c_{j,z} = c_j$. Let $\tilde{v}_i$ denote $v_{target}(F_i(s))$. The power for a partition i will be $p(F_i(s)) + \tilde{v}_i^2 \cdot \sum_{w_j \leq i} c_{j,i}$ after adding the small functional units to s. Let p_i denote $p(F_i(s))$, which is the power of partition i in s.

$$\begin{array}{ll} \min & p_{max} \\ s.t. & \sum_{z=w_j}^{k} c_{j,z} = c_j, j \in \{1, 2, \ldots, m\} \\ & p_i + \tilde{v}_i^2 \cdot \sum_{w_j \leq i} c_{j,i} \leq p_{max}, i \in \{1, 2, \ldots, k\} \end{array} \quad (13)$$

In the above linear programming formulation, $c_{j,z}$ and p_{max} are the variables and there can be at most $mk + 1$ variables. There are $m+k$ constraints. Since they are constant numbers, solving the linear program takes only constant time. For example, the standard simplex based linear program technique runs in $O((mk+m+k+1)^{m+k})$ time while the famous Karmarkar's algorithm runs in $O((mk+1)^{3.5})$ time. As an example, the linear programming formulation corresponding to Figure 4 is shown as follows.

$$\begin{array}{ll} \min & p_{max} \\ s.t. & c_{4,4} + c_{4,3} = c_4 \\ & c_{3,4} + c_{3,3} = c_3 \\ & c_{2,4} + c_{2,3} + c_{2,2} = c_2 \\ & c_{1,4} + c_{1,3} + c_{1,2} + c_{1,1} = c_1 \\ & p_4 + 1.5^2 \cdot (c_{4,4} + c_{3,4} + c_{2,4} + c_{1,4}) \leq p_{max} \\ & p_3 + 1.5^2 \cdot (c_{4,3} + c_{3,3} + c_{2,3} + c_{1,3}) \leq p_{max} \\ & p_2 + 1.2^2 \cdot (c_{2,2} + c_{1,2}) \leq p_{max} \\ & p_1 + 1.0^2 \cdot c_{1,1} \leq p_{max} \end{array} \quad (14)$$

1) Solving the linear programming combinatorially: Due to the specific property of our problem, we propose a combinatorial algorithm to solve the above linear program without calling the generic linear programming solver. In theory, our algorithm runs in $O(mk \log k)$ time which is more efficient than the above $O((mk + 1)^{3.5})$ runtime bound. In practice, the time due to data transfer between the linear programming solver and our program is saved as well.

Since the linear programming only needs to compute a continuous voltage partitioning solution, our algorithm uses the total capacitance of $c(F_{small,j})$ for each j. Given a partial solution s and a fixed

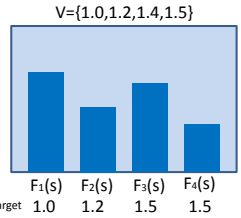

Fig. 4. A partial solution s with the assigned target voltages from the enumeration procedure.

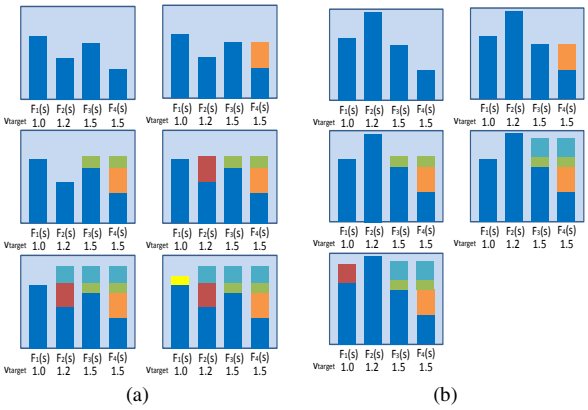

Fig. 5. Two examples of combinatorial algorithm.

combination of target voltage on each partition in s, our algorithm proceeds as follows. Recall that all the partitions $F_i(s)$ have been ordered according to the target voltage levels as in Equation (12). Start with $c(F_{small,m})$, find the smallest q such that each partition in $\mathcal{F}_m = \{F_q(s), F_{q+1}(s), \ldots, F_k(s)\}$ has target voltage equal to v_m. Sort the partitions in $\mathcal{F}_m$ according to the power in the non-decreasing order (note that we do not actually need to move them). Consider the following steps for handling $c(F_{small,m})$. Starting from the first partition in the sorted order, add some of $c(F_{small,m})$ until its power reaches that of the second partition. At that moment, simultaneously add some of $c(F_{small,m})$ to the first two partitions in the fashion that the power for both partitions are always equal, until the power reaches that of the third partition. Subsequently, simultaneously add some of $c(F_{small,m})$ to first three partitions to keep the power on each partition to be the same until it reaches the power of the fourth partition. Repeat this procedure until $c(F_{small,m})$ is used up. Note that the above description is only for illustration purpose. The exact amount of capacitance at each step (e.g., the amount of capacitances from $c(F_{small,m})$ to be added to first partition in order for its power to reach that of the second partition) can be computed in constant time. This finishes the processing on $c(F_{small,m})$. It takes $O(k \log k)$ time.

We next proceed to $c(F_{small,m-1})$. As above, find the smallest q such that each partition in $\mathcal{F}_{m-1} = \{F_q(s), F_{q+1}(s), \ldots, F_k(s)\}$ has target voltage greater than or equal to v_{m-1}. Sort the partitions in $\mathcal{F}_{m-1}$ according to power in the non-decreasing order. Starting from the first partition in the sorted order, add some of $c(F_{small,m-1})$ until its power reaches that of the second partition, then simultaneously add some of $c(F_{small,m})$ to the first two partitions in the fashion that the power for both partitions are equal until the power reaches that of the third partition. Repeat this procedure until $c(F_{small,m-1})$ is used up. This finishes the processing on $c(F_{small,m-1})$. In the similar fashion, the above procedure will be performed to $c(F_{small,m-2}), \ldots, c(F_{small,1})$. Handling each $c(F_{small,j})$ takes $O(k \log k)$ time and the whole process takes $O(mk \log k)$ time. Refer to Figure 5 for the illustration of the algorithm for two difference cases. We are to show that this simple greedy algorithm computes an optimal solution to the linear program, i.e., it returns an optimal continuous voltage partitioning solution. In a voltage partitioning solution, we call the partition achieving the peak power *peak power achieving partition* and note that there can be more than one peak power achieving partitions. Due to the space limitation, the proof of the following lemma is omitted.

Lemma 5: There exists an optimal continuous voltage partitioning solution to the linear program such that for the peak power achieving

partition with the lowest target voltage, denoted by F_q, one of the following two cases holds.

- No capacitances from F_{small} are added to F_q.
- Some capacitances from F_{small} are added to F_q, then all of $F_q, F_{q+1}, \ldots, F_k$ are peak power achieving partitions, i.e., they have the same power which is equal to the peak power.

Lemma 6: There exists an optimal continuous voltage partitioning solution such that none of the peak power achieving partitions use any capacitance from $c(F_{small,j})$ with $v_j \leq v_{target}(F_{q-1})$, where F_q is the peak power achieving partition with the lowest target voltage.

Our greedy algorithm explores the property shown in Lemma 5 and Lemma 6. Since one does not know the lowest target voltage of peak power achieving partition, our algorithm processes $F_{small,j}$ in the order of non-increasing voltage level v_j. Given a v_j, our algorithm treats it as that lowest target voltage and attempts to achieve the same power for all the partitions with voltage level greater than or equal to it. This is why our algorithm actually computes an optimal continuous solution of the linear programming formulation.

2) Linking enumeration with linear program solving: For a partial solution s (from dynamic programming), there are up to $O((m+1)^k)$ possible combinations on setting target voltages for all partitions of s, and there is one linear program corresponding to each combination. After solving all of these $O((m+1)^k)$ linear programming formulations, one can easily find the solution with the best peak power. This solution is actually the optimal continuous solution growing from s.

Let us denote the optimal continuous solution growing from s by $OPT_{s,continuous}$. In a similar fashion, one carries out the process to all other partial solutions. Let $OPT_{continuous}$ denote the solution with the best peak power over all the computed solutions from partial solutions, which is in fact the optimal continuous voltage partitioning solution. Thus,

$$p(OPT_{continuous}) = \min_s p(OPT_{s,continuous}). \tag{15}$$

The structure of $OPT_{continuous}$ is that F_{big} is partitioned in a discrete fashion, while F_{small} is partitioned in a continuous fashion and thus there could be fractional partitioning. Clearly, $p(OPT_{continuous}) \leq p(OPT)$.

To bound the runtime of the above process, for each partial candidate voltage partitioning solution s, there can be at most $(m+1)^k$ combinations computed by the enumeration procedure, and thus the runtime is bounded by $O(mk \log k \cdot (m+1)^k)$. Since there can be at most $O((\frac{mk^2}{\epsilon^4})^k)$ partial candidate voltage partitioning solutions according to Lemma 3, the total runtime is bounded by

$$O((\frac{mk^2}{\epsilon^4})^k \cdot mk \log k \cdot (m+1)^k), \tag{16}$$

which is independent of n.

F. Discretizing the best continuous solution

The best continuous solution $OPT_{continuous}$ is obtained. We are to discretize it using a simple greedy procedure as follows. Note that we only need to discretize this single continuous solution which is quite computationally efficient. Since $OPT_{continuous}$ is computed based on a partial solution from the dynamic programming, let s^c denote this partial solution. In the linear programming solution (computed using our combinatorial algorithm), the value of each $c_{j,z}$ is computed. Take discretizing c_1 as an example. One is to assign the functional units in $F_{small,1}$ to each partition of s^c according to $c_{1,z}$. Let $F_{small,1,i}$ denote the set of functional units assigned to $F_i(s^c)$. The assignment needs to be performed such that $c(F_{small,1,i})$ is close to $c_{1,i}$ for each i since the optimal continuous solution is a lower bound on the optimal solution. For this, arbitrarily order the functional units in $F_{small,1}$. Maintain a cumulative sum which is initialized to zero. Starting with the first partition in s^c (in the more general case it is the partition w_j), put the first functional unit to it, update the cumulative sum through adding the capacitance of this functional unit, and remove it from $F_{small,1}$. One then similarly handles the current first functional unit (the second functional unit in the original list), i.e., put it to the first partition, update the cumulative sum and remove it. This process is iterated until the cumulative sum is greater than or equal to $c_{1,1}$ or all the functional units have been removed. All the removed functional units form $F_{small,1,1}$. Subsequently, set cumulative sum to zero and move to the second partition in s^c to compute $F_{small,1,2}$. Perform the same procedure to all other partitions. After that, we will discrete $c_2, c_3, \ldots, c_m$ in the same way.

To measure the error (capacitance difference) between the discretized solution and the continuous solution, note that $c(F_{small,1,1}) - c_{1,1}$ is bounded by the largest capacitance in F_{small}. Given any partition, there can be at most m times of involved discretization process since there are only m sets of $F_{small,j}$ (note that at any partition differently colored capacitance can be actually from the same $F_{small,j}$ in Figure 5). Thus, at any partition the total capacitance increase over the same partition of $OPT_{continuous}$ is at most $m \cdot \max_{u \in F_{small}} c(u)$. This leads to the maximum power increase for the same partition to be at most $m \cdot \max_{u \in F_{small}} c(u) \cdot v_{max}^2$. On the other hand, due to our choice of threshold in categorizing F_{big} and F_{small}, the largest capacitance of F_{small} is bounded by $\epsilon L'$. Thus, the maximum power increase at any partition is bounded by

$$m \cdot \epsilon L' \cdot v_{max}^2 = m\epsilon L. \tag{17}$$

The runtime of this greedy algorithm based discretization process is linear in terms of the number of functional units in F_{small} which is bounded by $O(n)$.

G. Approximation ratio and runtime

We are to compute the approximation ratio of the whole algorithm. There are two types of errors which need to be handled. The first one is the rounding error in dynamic programming and the second one is the error in discretization of the continuous solution. Let ALG denote our discretization solution to the continuous voltage partitioning solution $OPT_{continuous}$. According to Equation (17), one has $p(ALG) \leq p(OPT_{continuous}) + m\epsilon L$.

According to Lemma 4, for any voltage partitioning solution (including the optimal voltage partitioning solution), there is always a dynamic programming solution (called corresponding dynamic programming solution) with maximum power increase at any partition bounded by $\frac{2kv_{max}^4}{v_{min}^4} \cdot \epsilon p(OPT)$. Let us denote the dynamic programming solution corresponding to the optimal voltage partitioning solution OPT by s^*. If one optimally grows s^* (i.e., adding F_{small} to it in an optimal fashion), the resulting solution would have the peak power at least $p(OPT_{s^*,continuous})$, since the optimal continuous solution (computed using linear programming formulation) gives a lower bound on any solution growing from s^*. Thus, the difference between $p(OPT_{s^*,continuous})$ and $p(OPT)$ is only bounded by the rounding error in dynamic programming. One has $p(OPT_{s^*,continuous}) \leq p(OPT) + \frac{2kv_{max}^4}{v_{min}^4} \cdot \epsilon p(OPT)$.

We next include the discretization error. Since $OPT_{continuous}$ is the minimum peak power solution over all $OPT_{s,continuous}$, one has

$$\begin{aligned} p(ALG) &\leq p(OPT_{continuous}) + m\epsilon L \\ &\leq [1 + (\tfrac{2kv_{max}^4}{v_{min}^4} + m)\epsilon] \cdot p(OPT) \end{aligned} \tag{18}$$

Setting $\epsilon' = (\frac{2kv_{max}^4}{v_{min}^4} + m)\epsilon$, ALG is a $(1+\epsilon')$ approximation to the optimal voltage partitioning solution. One can choose reasonable ϵ to satisfy $0 < \epsilon' < 1$ without contrary to assumption $0 < \epsilon < 1$.

We are to bound the total runtime of our algorithm as follows. It takes linear time to compute the threshold. The dynamic programming on big functional units takes $O(k^2 \cdot (\frac{mk^2}{\epsilon^4})^k)$ time. The runtime for linear programming over all partial voltage partitioning solutions is bounded by $O((\frac{mk^2}{\epsilon^4})^k \cdot mk \log k \cdot (m+1)^k)$ (since computing continuous solutions only needs the total capacitance for each voltage category in F_{small}). The runtime for discretization is $O(n)$ since we only discretize the best continuous solution. Therefore, the total runtime is bounded by $O(n + (\frac{mk^2}{\epsilon^4})^k \cdot mk \log k \cdot (m+1)^k)$. Plugging $\epsilon = \epsilon'/(\frac{2kv_{max}^4}{v_{min}^4} + m)$, the runtime is $O(n + (\frac{mk^2(\frac{2kv_{max}^4}{v_{min}^4}+m)^4}{\epsilon'^4})^k \cdot mk \log k \cdot (m+1)^k)$. Since all of the parameters other than ϵ' are constants, it can be simplified to $O(n + \frac{1}{\epsilon'^{O(1)}})$. We reach the following theorem.

Theorem 7: A $(1+\epsilon)$ approximation to the voltage partitioning for peak power minimization problem can be computed in $O(n + 1/\epsilon^{O(1)})$ time for any $0 < \epsilon < 1$, where n is the number of functional

TABLE III
COMPARISON OF OLDFPTAS WITH SOLUTION RESTRICTION OF 10 SOLUTIONS AND NEWFPTAS ON THE TESTCASES WITH $k = 5$ PARTITIONS. CPU REFERS TO RUNTIME IN SECONDS.

Test cases		Total Power Minimization		OLDFPTAS w/ Solution Restriction (with $\epsilon = 5\%, k = 5$)			NEWFPTAS (with $\epsilon = 5\%, k = 5$)		
Index	# units	Peak Power	Total Power	Peak Power	Total Power	CPU(s)	Peak Power	Total Power	CPU(s)
1	500	3933.6	11843.4	3214.3	12355.3	24.5	3207.6	12123.5	0.3
2	1000	6487.7	23804.6	6095.8	28039.0	86.1	5635.3	25175.3	0.3
3	2000	18925.0	52767.4	12838.1	55376.5	339.6	12131.2	54693.1	0.3
4	4000	38564.8	99372.3	26001.3	116394.7	1809.7	23818.7	105872.4	0.3
5	5000	43272.8	129222.1	32463.0	144561.8	3072.5	30267.3	135732.6	0.3
Average		22236.8	63402.0	16122.5	71345.5	1066.5	15012.0	66719.4	0.3

units, assuming that the number of partitions k and the number of voltage levels m are constants.

IV. EXPERIMENTAL RESULTS

The proposed linear time approximation scheme for the voltage partitioning problem is implemented in C++ and tested on an Intel machine with $1.8GHz$ CPU and $2GB$ memory. Our technique is compared to the previous technique in [8], and the same experimental setup, testcases and technology library from [8] are used in this paper. Precisely, it includes 10 small testcases with 50 to 100 functional units, 5 large testcases with 500 to 5000 functional units, and a technology library consisting of five voltage levels. In addition, we also perform experiments on the standard GSRC floorplanning benchmark circuits. Let OLDFPTAS denote the algorithm in [8] and let NEWFPTAS denote our algorithm.

TABLE IV
COMPARISON OF OLDFPTAS (WITHOUT SOLUTION RESTRICTION) AND NEWFPTAS ON A SET OF 10 SMALL TESTCASES ON $k = 2$ PARTITIONS. # UNITS REFERS TO THE NUMBER OF FUNCTIONAL UNITS (BLOCKS) AND CPU REFERS TO RUNTIME IN SECONDS.

Testcases		OLDFPTAS (with $\epsilon = 5\%, k = 2$)		NEWFPTAS (with $\epsilon = 5\%, k = 2$)	
Index	# units	Peak Power	CPU(s)	Peak Power	CPU(s)
1	50	648.9	48.4	652.4	8.6
2	55	708.0	83.4	716.5	5.4
3	60	789.3	128.0	799.9	5.4
4	70	932.7	208.0	945.1	4.6
5	75	1003.4	283.4	1018.2	4.6
6	80	1068.3	397.7	1092.7	4.6
7	85	1136.6	600.5	1153.6	1.3
8	90	1205.5	798.7	1223.5	0.2
9	95	1264.3	1009.5	1282.8	0.2
10	100	1325.3	1325.0	1340.7	0.1
Average		1008.2	488.3	1022.5	3.5

Due to the space limitation, we choose to present the results for $\epsilon = 5\%$, i.e., the computed solution has peak power at most 5% over the optimal solution. This is desired as in practice, one would like to compute the solution sufficiently close the the optimal solution while still running fast. We first compare NEWFPTAS with OLDFPTAS on small testcases and the results are summarized in Table IV. We make the following observations.

- Due to the theoretical guarantee, both algorithms return the solutions satisfying the target approximation ratio. However, NEWFPTAS runs much faster than OLDFPTAS. The main reason is that OLDFPTAS performs a time consuming dynamic programming algorithm involving all of the n functional units. In contrast, NEWFPTAS performs the dynamic programming involving only 4 functional units even for the largest testcase where $n = 100$ for handling F_{big} (i.e., $|F_{big}| = 4$). The remaining process of NEWFPTAS for computing the continuous solutions on F_{small} does not involve n at all, while the last discretization process only needs a simple and efficient linear time algorithm.
- It is interesting to see that NEWFPTAS runs faster for larger testcases. The reason is that the number of big functional units $|F_{big}|$ decreases with the increase of the testcase size n. For example, $|F_{big}|$ is 25 when $n = 50$ while it is 4 when $n = 100$. The reason for this is that the threshold L/v_{max}^2 increases with n since the lower bound L increases with n (as in Equation (2) $L \geq c(F_{min}) \cdot v_{min}^2$ and $c(F_{min})$ increases with n).

Due to the large time and memory complexity of $\Omega(n^{2k})$, OLDFPTAS cannot finish for handling either moderately large n or moderately large k (e.g., $n = 500$ or $k = 5$). In our experiments, in these cases OLDFPTAS runs out of memory. Thus, the solution restriction technique needs to be used. Basically, it limits the maximum number of solutions which can be maintained during dynamic programming. If the actual number of non-redundant solutions is greater than this value, the solutions will be pruned aggressively even if they are not redundant. Due to this, OLDFPTAS with solution restriction losses the theoretical guarantee. In fact, [8] shows that they need to limit the number of solution to only 10 in order to make it run in a reasonable time, which is certainly quite aggressive. We perform the experiments on large testcases with $k = 5$. The results are summarized in Table III. We make the following observations.Comparing the solution quality, OLDFPTAS returns the solution significantly worse (on average 7.4% worse) than that of NEWFPTAS. In fact, it does not achieve theoretical guarantee on approximation ratio due to the solution restriction. NEWFPTAS runs much faster than OLDFPTAS even if OLDFPTAS uses the solution restriction. For the testcase with $n = 5000$, NEWFPTAS runs more than $10000\times$ faster and reduces the runtime from about 1 hour to less than 1 second. The reason is still due to the time consuming dynamic programming process. For all of these large testcases, $|F_{big}| = 0$ since the thresholds become quite large as the lower bounds L are high for $n \geq 500$. Thus, in fact no dynamic programming procedure is performed in NEWFPTAS while still satisfying the theoretical guarantee. Comparing NEWFPTAS with the total power minimization algorithm [3], [4], one can see that the NEWFPTAS significantly reduces peak power with little increase in total power.

To demonstrate the impact of our NEWFPTAS to voltage island shutdown for energy reduction, as in [8] the NEWFPTAS is integrated into a simulated annealing based floorplanning technique to compute the floorplan with voltage island generation. We compare the resulting voltage islands with the one from a natural greedy frequency based voltage partitioning technique used in [8]. The greedy algorithm used in [8] can be briefly described as sorting blocks in the decreasing order according to the frequency of a block being idle (called block idle frequency) followed by cutting the list to form the voltage partitions. The results of energy consumption considering voltage island shutdown effect for GSRC benchmark $n100$ are summarized in Table V. One can see that the greedy partitioning performs well for the sequence used in computing the idle frequency. However, its results on other sequences are usually much worse especially comparing to NEWFPTAS since the greedy algorithm is not optimized for them. Summing over all of five sequences, each voltage island has similar shutdown frequency. Since peak power driven voltage partitioning balances the power distribution over all partitions, NEWFPTAS can achieve large energy reduction (about 25%) through voltage island shutdown. This demonstrates the effectiveness of the proposed algorithm for energy reduction through voltage island shutdown.

TABLE V
RESULTS OF ENERGY CONSUMPTION THROUGH PERFORMING VOLTAGE ISLAND SHUTDOWN ON GSRC $n100$.

Algorithms	Seq1	Seq2	Seq3	Seq4	Seq5	Total	Ratio
Seq1 Greedy	219550	244080	214835	226411	228622	1133498	1.24
Seq2 Greedy	228467	239607	220336	228715	234180	1151305	1.26
Seq3 Greedy	222809	243428	214568	229664	233013	1143482	1.26
Seq4 Greedy	219283	238814	215560	221442	235232	1130331	1.24
Seq5 Greedy	226277	245177	216721	229164	232938	1150277	1.26
NEWFPTAS	179705	191670	171091	185353	182750	910569	1.00

REFERENCES

[1] R. Shelar and M. Patyra, "Impact of Local Interconnects on Timing and Power in a High Performance Microprocessor," in *Proceedings of ACM International Symposium on Physical Design*, 2010.
[2] D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould, and J. Cohn, "Managing Power and Performance for System-on-Chip Designs Using Voltage Islands," in *Proccedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 195 – 202, 2002.
[3] Z. Gu, Y. Yang, J. Wang, R. Dick, and L. Shang, "Taphs: Thermal-Aware Unified Physical-Level and High-Level Synthesis," in *Proceedings of IEEE Asia and South Pacific Design Automation Conference*, pp. 879 – 885, 2006.
[4] H.-Y. Liu, W.-P. Lee, and Y.-W. Chang, "A Provably Good Approximation Algorithm for Power Optimization Using Multiple Supply Voltages," in *Proceedings of IEEE/ACM Design Automation Conference*, pp. 887 – 890, 2007.
[5] T. Lin, S. Dong, B. Yu, S. Chen and S. Goto, "A revisit to voltage partitioning problem", in *Proceedings of the IEEE/ACM Symposium on Great Lakes Symposium on VLSI*, pp. 115 – 118, 2010.
[6] M. Hamada, Y. Ootaguro, and T. Kuroda, "Utilizing Surplus Timing for Power Reduction," in *Proceedings of IEEE Conference on Custom Integrated Circuits*, pp. 89 – 92, 2001.
[7] J. Wang and S. Hu, "The Fast Optimal Voltage Partitioning Algorithm For Peak Power Density Minimization", in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2010.
[8] J. Wang, X. Chen, C. Liao and S. Hu, "The Approximation Scheme For Peak Power Driven Voltage Partitioning", in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2011.
[9] H. Kellerer and U. Pferschy, *A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem*, Journal of Combinatorial Optimization, Vol. 3, No. 1, pp. 59-71, 1999.