Placement-Proximity-Based Voltage Island Grouping Under Performance Requirement

Huaizhi Wu, Martin D. F. Wong, *Fellow, IEEE*, I-Min Liu, and Yusu Wang

***Abstract*—High power consumption not only leads to short battery life for hand-held devices but also causes on-chip thermal and reliability problems in general. As power consumption is proportional to the square of supply voltage, reducing supply voltage can significantly reduce power consumption. Multi-supply voltage (MSV) has previously been introduced to provide finer grain power and performance tradeoff. In this paper, we propose a methodology on top of a set of algorithms to exploit nontrivial voltage island boundaries for optimal power versus design-cost tradeoff under performance requirement. Our algorithms are efficient, robust, and error-bounded and can be flexibly tuned to optimize for various design objectives (e.g., minimal power within a given number of voltage islands, or minimal fragmentation in voltage islands within a given power bound) depending on the design requirement. Our experiment on real industry designs shows a tenfold improvement of our method over current logical-boundary-based industry approach.**

***Index Terms*—Low power, optimization, timing, voltage island.**

I. Introduction

With the broadening market interests in sophisticated mobile applications, meeting aggressive power target on top of performance requirement in high-speed portable design is becoming a challenging task. As the design parameters for optimal power and optimal performance often contradict each other (for example, a lower supply voltage reduces power consumption but slows device speed), designers are constantly struggling to balance power and performance throughout the chip-design cycle.

High power consumption not only leads to short battery life for hand-held devices but also causes on-chip thermal and reliability problems in general. At a 90-nm process node, the vast amount of functionality integrated within SoC designs, compounded with much larger leakage current, is already leading to designs with power dissipation in the hundreds of Watts. As process technology is trending against power dissipation, this problem is expected to only get worse at the future process nodes.

Manuscript received December 30, 2005; revised May 30, 2006. This work was supported in part by the National Science Foundation under Grant CCR-0306244. This paper was recommended by Associate Editor C. J. Alpert.

H. Wu was with Cadence Design Systems, Inc., San Jose, CA 95134 USA. She is now with Atoptech, Inc., Santa Clara, CA 95054 USA (e-mail: linda@atoptech.com).

M. D. F. Wong is with the Department of Electrical and Computer Engineering, University of Illinois, Urbana–Champaign, IL 61801 USA (e-mail: mdfwong@uiuc.edu).

I-M. Liu is with Atoptech, Inc., Santa Clara, CA 95054 USA (e-mail: imliu@atoptech.com).

Y. Wang is with the Department of Computer Science and Engineering, Ohio State University, Columbus, OH 43210 USA (e-mail: yusu@cse.ohiostate.edu).

Digital Object Identifier 10.1109/TCAD.2006.888270

Power consumption generally breaks down into two sources: dynamic and static powers [1]. Static power related to CMOS devices includes reverse-biased junction leakage, gate induced drain leakage, direct-tunneling leakage, and subthreshold leakage [2]. With the current technology, subthreshold leakage is much larger than the other ones, due to reduced Vt. While static power comes from leakage current, dynamic power P_d is the result of a device's switching activities. It can be represented by

$$P_d = kcfV_{\mathrm{dd}}^2 \tag{1}$$

where k is the switching rate, c is the load capacitance, f is the clock frequency, and V_{dd} is the supply voltage. Static and dynamic powers are comparable in many of today's logic designs. Techniques to lower switching power are combinations of reducing switching activity, load capacitance, and supply voltage. For example, clock frequency can be set to zero by gating the clock to an inactive logic block. Load capacitance can be reduced by minimizing total wire length and by downsizing the gates.

As dynamic power is proportional to the square of the supply voltage V_{dd}, reducing V_{dd} can significantly reduce dynamic power. Meanwhile, when higher power-supply voltage is selectively available, the percentage of low Vt devices needed in a design become less, thus reducing leakage power also. However, the delay increase due to reduced V_{dd} degrades the circuit performance. Multi-supply voltage (MSV) can be used to provide finer grain power and performance tradeoff. In an MSV design, high V_{dd} is assigned to cells on critical paths while low V_{dd} is assigned to cells on noncritical paths, so that power can be saved without degrading the overall circuit performance [3].

Although, in theory, only timing-critical cells need high V_{dd}, this naive thinking for maximum power reduction is not practical. When low V_{dd} and high V_{dd} interleave heavily, there is a significant overhead in voltage-shifting devices between low V_{dd} and high V_{dd} cells to eliminate the undesirable static current that will otherwise flow. Moreover, it is expensive to implement the resulting fragmented power networks, as implementing such complex-power network is not only tedious human work but also takes a lot of precious routing resources, which is not good especially when per-metal-layer manufacturing cost is soaring as process technology migrates.

Previous efforts toward reducing the level-shifting overhead include: 1) clustered voltage scaling (CVS) [4] and 2) extended CVS (ECVS) [5]. In CVS, the cells driven by each power supply are grouped (clustered) together and level shifting is needed only at sequential element outputs. In ECVS, level shifting is

allowed anywhere within the combinational logic block using an asynchronous level shifter. The added flexibility in ECVS can provide greater power reduction than CVS. However, the delay penalty tends to be larger too.

Grouping cells of different supply voltages into a small number of "voltage islands" [6]–[8], where each voltage island occupies a contiguous physical space and operates at a single supply voltage that meets the performance requirement, can also effectively reduce the amount of level shifting, as well as reduce the cost of the power network. The current state-of-the-art design of voltage islands is largely done manually and is primarily based on the design's logic hierarchy. That is, designers partition circuits into a few groups based on their performance requirement and the connectivity between modules. Each group is then specified with a supply voltage. Logic boundaries are largely used in this grouping process mainly because they are the boundaries that designers are most familiar with. However, these "natural" boundaries in a design are almost always nonoptimal boundaries for supply voltages.

Fig. 1 illustrates why sticking to logical boundaries is limiting the solution space in producing optimal MSV. In the example, there are three modules, each of them contains only leaf cells, and both modules A and B contain some timing-critical cells that require high voltages [Fig. 1(a)]. Fig. 1(b) and (c) is a design based on logical boundary. While Fig. 1(b) guarantees the performance using high power, Fig. 1(c) reduces the power consumption without meeting the timing requirement. None of them are optimal MSV. By using placement proximity (instead of logical) information, the optimal MSV meets power and timing requirements at the same time while keeping the number of power domains small [Fig. 1(d)].

In this paper, we wish to reduce the cost of the power network in an MSV design by voltage-island grouping (we also call it voltage-island partitioning).[1] We propose a methodology on top of a set of algorithms to exploit the nonlogical boundary in a design for optimal voltage-island grouping that captures the power versus design-cost tradeoff under performance (timing) requirement. Depending on each designer's specific needs, the optimal tradeoff can be explored by either one of the two dual optimization problems: Maximally reduce power consumption within a given bound on a number of voltage islands or create minimally fragmented voltage islands within a given bound on power consumption. Our approach can handle both problems, with the latter having an extra $\log(k)$ factor in running time (k is the number of voltage islands), coming from a binary search for k. We will focus on the latter problem in the following discussion. However, all our results can be easily adapted to the former one.

Our contributions can be summarized as follows: To our best knowledge, we are the first to consider power versus design-cost tradeoff under timing requirement for the voltage-island-grouping problem. In particular, we exploit nontrivial voltage-island boundaries to balance power consumption and power-network fragmentation. We formulate this problem as a partitioning problem (Section II). This voltage-partitioning problem (VPP) is hard (some closely related variants with much simpler weight functions are NP-hard), and we, thus, study approximation algorithms (i.e., algorithms that guarantee solutions with good guarantees) and present one that runs in polynomial time. This algorithm, unfortunately, is not efficient enough for practical designs. We, therefore, design an efficient two-step heuristic algorithm, which combines dynamic programming with variable-sized $p \times q$ gridding (Section III). We show (Section IV) that our method is efficient and practical, as well as produces good-quality voltage islands for a wide selection of industry data. Compared to the current industry approach using logical boundaries within a design, our method generates about one tenth of voltage islands for the same amount of power reduction. The running time is small even for very large industry designs.

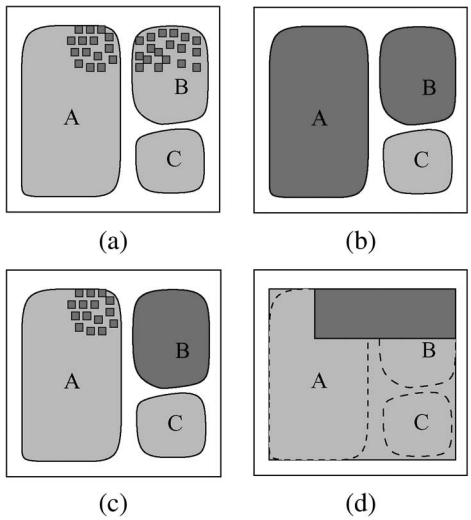


Fig. 1. (a) Design with timing-critical cells (small darker cells). (b) Power consumption too high. (c) Timing requirement not met for small cells in module A. (d) Placement-proximity-based solution with nonlogical boundary.

II. Voltage-Partitioning Problem

A. Problem Definition

Let A be an $m \times n$ array, and $A[i][j]$ the value of the element at position (i, j), $1 \le i \le m$, $1 \le j \le n$. A subarray $A[l \cdots r, b \cdots u]$ is the rectangular region in A with (l, b) [respectively (r, u)] as the bottom-left (respectively upper right) corner. We may also refer to an array (or a subarray) as a rectangle or a region. Let $\mu(R) = \max_{(i,j)\in R} A[i][j]$ be the maximum value of all elements in a rectangle R. The weight of a rectangle R is defined as

$$\omega(R) = \sum_{(i,j)\in R} (\mu(R) - A[i][j]) .$$

See Fig. 2(a) for an illustration.

A partitioning of A is a set of disjoint rectangles (subarrays) $\Pi = \{R_1, \ldots, R_k\}$ that cover A; $k = |\Pi|$ is called the size of

[1]We assume that power consumption is monotonic in the number of voltage islands, meaning that we consider device power and delay only depending on its own supply voltage but not on others. In theory, a device's power and delay are functions of its own supply voltage, as well as the supply voltages of other devices due to coupling. In this paper, we ignore such coupling effects, as they are secondary and can be optimized by other techniques if arisen.

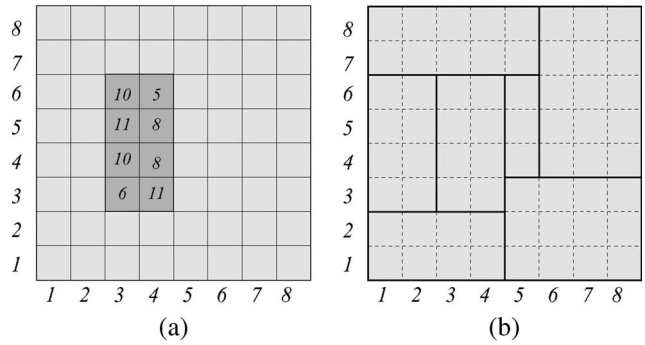


Fig. 2. (a) 9×9 array A, with a rectangle (shadowed region) $R = A[3 \cdots 4, 3 \cdots 6]$; $\mu(R) = 11$ and $\omega(R) = 19$. (b) Partitioning of A with seven rectangles.

this partitioning. See Fig. 2(b) for an illustration. The weight of a partitioning Π is defined as

$$\omega(\Pi) = \sum_{1 \le t \le k} \omega(R_t).$$

In the voltage-island-partitioning problem, we wish to subdivide the placement region into a small number of voltage islands (where every cell in the same voltage island will eventually receive the same voltage), while keeping the total power consumption low and timing requirement met. The latter means that the voltage assigned to each cell should be no lower than its required value. Therefore, the voltage value of each voltage island should be the maximum required voltage of all cells in this voltage island. Raising voltages of cells with lower requirement to this maximum value will result in an increase in the power consumption of this voltage island. To keep the overall power consumption low, it is desirable to have the total power penalty on all voltage islands below some threshold.

We can consider each standard-cell-placement region as a 2-D array A, induced by the underlying placement grid. As dynamic power is proportional to the square of the supply voltage (1), we let $A[i][j]$ be the square of the required voltage of the corresponding standard cell covering this grid. Then, it is easy to see that a partitioning of A corresponds to a voltage-island partitioning of the placement region, $\sqrt{\mu(R)}$ is the voltage value of each voltage island R, and the weight of the partitioning represents the total power penalty. We can, thus, formally define the VPP as follows.

Definition 1 (VPP): Given an $m \times n$ array A and an error threshold δ, among all partitionings whose weight is at most δ, find one with the smallest size. Let $\kappa(A, \delta)$ be the size of this optimal partitioning.

The dual version of this problem (DVPP) is defined as the problem of minimizing the weight of the partitioning with a bound on the size. We will focus on VPP in this paper. However, our algorithm can easily handle DVPP as well.

B. Algorithm With Guarantees

While no previous work has been done for the VPP problem, some variants of it are well studied. In particular, if we define the weight of a rectangle R as the sum of all $A[i][j]$, $(i, j) \in R$, and the weight of a partitioning Π as the maximum weight of all rectangles in Π, we have a variant of the VPP problem, previously referred to as the RTILE problem [9]. Intuitively, this max-sum weight function is much simpler than the one we consider in the VPP problem, and it has been shown that the RTILE problem is NP-hard [9]. Therefore, given the indication of the hardness of our problem,[2] we shift our focus to approximation algorithms, which provide a guarantee on the output size. Below, we first describe our two-approximation algorithm for the VPP problem, which finds a partitioning Π of a given array A so that $\omega(\Pi) \le \delta$ and $|\Pi| \le 2\kappa(A, \delta)$. This approximation algorithm is based on a nicely structured class of partitionings, which we introduce next.

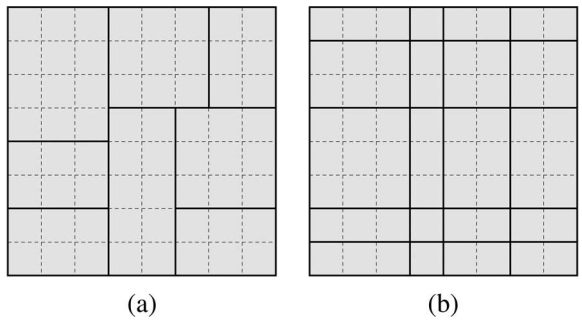


Fig. 3. (a) Slicing partitioning of size eight. (b) Grid partitioning of size 5×4.

Slicing model: Given an input region (array) A, we can slice it either with a vertical or a horizontal cut. Each cut will divide its parent region into two, and we then slice the resulting two children regions recursively. An example is shown in Fig. 3(a). A partitioning obtained this way is called a slicing partitioning of region A [10], [11]. In fact, it is the same as a special type of binary-space partitioning (BSP) induced by orthogonal cuts, called orthogonal BSP (see A for more details). It has been shown in [12] that given a set of m disjoint axis-aligned rectangles that cover a rectangular region A, one can construct an orthogonal BSP so that in the induced slicing partitioning of A, each region lies completely inside one of the m original rectangles, and the size of this slicing partitioning is at most $2m$. Let $\rho(A, \delta)$ denote the size of the optimal slicing partitioning of A with weight at most δ and $\kappa(A, \delta)$ denote that of the optimal arbitrary partitioning. We can infer the following lemma.

Lemma 1: $\kappa(A, \delta) \le \rho(A, \delta) \le 2\kappa(A, \delta)$.

Proof: The left inequality is obvious. For the right one, the optimal solution Π^* of the VPP gives $\kappa(A, \delta)$ disjoint rectangles that cover region A. By [12], there exists a slicing partitioning Π of A with at most $2\kappa(A, \delta)$ rectangles (regions). Since every rectangle in Π is completely inside a rectangle in Π^*, the weight of the slicing partitioning Π cannot increase, i.e., $\omega(\Pi) \le \omega(\Pi^*) \le \delta$. Thus, proves the right inequality. ■

The above lemma states that an optimal slicing partitioning for A is a two-approximation for the VPP. Therefore, we now only need to focus on solving the VPP under this slicing

[2]The VPP under $p \times q$ grid partitioning is NP-complete, which can be shown by a straightforward extension of the NP-completeness proof of the RTILE problem under $p \times q$ grid partitioning. Unfortunately, its hardness under arbitrary partitioning does not seem to be trivial. We are not able to prove its NP-completeness nor disprove it; although, we tend to believe that the former is true.

model. One standard method to handle the slicing partitioning is by using dynamic programming [12]. We first describe DP-Alg, a dynamic programming-based two-approximation algorithm for VPP.

Dynamic programming approach: First, we show that the dual DVPP under the slicing model can be solved optimally by dynamic programming. It follows a similar framework as the one in [13], which was developed for the RTILE problem and several other variants of it.[3]

Let $\omega_s^*(l\cdots r, b\cdots u)$ denote the minimum weight of any rectangle (subarray) R in A with at most s disjoint subrectangles, where $R = A[l\cdots r, b\cdots u]$. The ultimate goal is to compute $\omega_k^*(1\cdots m, 1\cdots n)$. At the base level, if $s = 1$, or if $\omega(R) = 0$, we set $\omega_s^*(l\cdots r, b\cdots u) = \omega(R)$. Otherwise, we have the recursion as shown at the bottom of the page. The second minimization term enumerates all horizontal and vertical cuts, and the first term $(1 \le t < s)$ enumerates all possible ways to assign the number of rectangles (t and $s - t$) for the two separated parts obtained by some horizontal or vertical cut.

For the base cases, the weight $\omega(R)$ of any rectangle R can be computed in $O(1)$ time after $O(nm)$ time/space preprocessing of the input array A, by extending the prefix-sum algorithm from [9] to our weight function (see B for more details). It then follows from the above recursion that we can build the remaining dynamic programming table in $O((n+m)n^2m^2k^2)$ time, where we spend $O((n+m)k)$ time to compute each of the $O(n^2m^2k)$ entries of the table. We, thus, have the following result.

Lemma 2: Given an $m \times n$ array A and an integer $k > 0$, we can compute in $O((n+m)n^2m^2k^2)$ time the minimum-weight slicing partitioning for A of size k. The space complexity is $O(n^2m^2k)$.

This provides an optimal solution for the DVPP under the slicing model. For the primal VPP under the same model, we can now guess the optimal number of rectangles $\bar{k} = \rho(A, \delta)$ in $O(\log \bar{k})$ time by a binary search, starting with $\bar{k} = 1$. Since $O(\bar{k}) = O(k^*)$, together with Lemmas 1 and 2, we conclude with the following result.

Theorem 1: Given an $m \times n$ array A and an error bound $\delta > 0$, a two-approximation of $k^* = \kappa(A, \delta)$ can be computed in $O((n+m)n^2m^2(k^*)^2 \log k^*)$ time and $O(n^2m^2k^*)$ space.

III. Fast Heuristic Algorithm

In this section, we describe TS-Alg, an efficient two-step heuristic algorithm for the VPP.

A. Algorithm Overview

Ideally, we would like to have some guarantee on the size of the output partitioning, while keeping the complexity of the algorithm low. DP-Alg, as introduced in last section, produces solutions with good guarantees. Unfortunately, it is too slow and memory inefficient to be practical. In fact, the large space requirement limits the size of the input array to merely around 100×100,[4] while the size encountered in practice can easily go up to $50\,000 \times 50\,000$. We, therefore, want to first reduce the size of the input array, before we feed it to DP-Alg. In particular, we can impose a second grid of lower resolution on the original array A, and obtain a "compressed" array G, where each element $G[u][v]$ corresponds to a rectangle (subarray) R_{uv} of A at grid (u, v). To guarantee meeting the timing requirement for the voltage islands, we let $G[u][v] = \mu(R_{uv}) = \max_{(i,j)\in R_{uv}} A[i][j]$. Note that G actually corresponds to a special type of partitioning of A: Suppose G is of size $p \times q$, we call the underlying partitioning Π_G a grid partitioning of size $p \times q$ for A [see Fig. 3(b)], and each rectangle R_{uv} in Π_G is referred to as a grid rectangle. To avoid excessive power penalty in such partitioning, we would like to have a bound on $\omega(\Pi_G)$, where $\omega(\Pi_G) = \sum_{1\le u\le p, 1\le v\le q} \omega(R_{uv})$.

This motivates us to design the following TS-Alg for the VPP, referred to as TS-Alg.

Step 1) Size reduction: produce a $p \times q$ array G with $\omega(\Pi_G) \le \epsilon$, for some $\epsilon \le \delta$.

Step 2) Approximate voltage partitioning: apply DP-Alg on G to compute a partitioning Π with $\omega(\Pi) \le \delta$.

Note that both the quality and quantity of the first step directly affect the performance of the second step. On the one hand, we hope that the quantity (i.e., the value of p and q) is small, so that DP-Alg is fast and practical. On the other hand, as DP-Alg will not further subdivide any grid rectangle in Π_G (i.e., a rectangle in the final output Π will be a combination of some grid rectangles in Π_G), grid rectangles in Π_G should be "good." We will see in what follows that although TS-Alg does not guarantee that the output size will approximate $\kappa(A, \delta)$ within some constant factor, there is a control on the quality in each of the two steps. The experimental results from next section further demonstrate the performance of TS-Alg both in efficiency and in output quality.

B. Size Reduction

One straightforward approach for Step 1) of TS-Alg is to simply subdivide A evenly into $p \times q$ grids (so that all pq grid rectangles in Π_G are congruent). This method is completely

[3]We remark that this dynamic programming paradigm is general and applies to different weight functions that are superadditive [13], which informally means that splitting a rectangle in any partitioning does not increase the overall weight. As aforementioned, our $\omega(\Pi)$ in the VPP is superadditive.

[4]We remark that there are some running-time/space improvement over the above DP algorithm that can be extended to our case as well. However, it increases the approximation factor greatly [13]. Our goal is to have an efficient algorithm that performs well in practice.

$$\omega_s^*(l\cdots r, b\cdots u) = \min_{1\le t<s} \left\{ \min_{l\le i<r, b\le j<u} \left\{ \begin{array}{l} \omega_t^*(l\cdots i, b\cdots u) + \omega_{s-t}^*(i+1\cdots r, b\cdots u) \\ \omega_t^*(l\cdots r, b\cdots j) + \omega_{s-t}^*(l\cdots r, j+1\cdots u) \end{array} \right\} \right\}$$

oblivious to the data distribution in A, thus, leading to a suboptimal result. It is desirable to subdivide A more intelligently so as to preserve the data distribution in A, by using more flexible variable-sized grids. In particular, we ask the following question for Step 1).

Definition 2 (Grid-Partitioning Problem, or GPP): Given an $m \times n$ array A and an error threshold ϵ, among all grid partitionings whose weight is at most ϵ, find one with the smallest size (i.e., $p \times q$).

Variants of this problem have already been studied in computational geometry [14], [15], and we modify the algorithm from [15] to obtain the following result.[5]

Lemma 3: Let $p^* \times q^*$ be the size of the optimal grid partitioning with weight at most $\epsilon/2$. One can compute in $O(nm + (n + m + \bar{p}\bar{q}) \cdot \bar{p} \log(nm))$ time and $O(nm)$ space a grid partitioning of weight at most ϵ and of size $\bar{p} \times \bar{q}$, where $\bar{p} \times \bar{q} \leq 17p^* \times q^*$.

On the high level, the GPP can be reduced to a special type of set cover problem, which can be efficiently approximated using techniques from randomized algorithms (in particular, ε nets [14] and an elegant analysis given by Clarkson in [16]. See C for more details). Roughly speaking, it uses the iterative doubling technique originally used for linear-programming problem [16]. It assigns a load to every possible separator for grid partitioning (i.e., all vertical and horizontal grid lines). The load was referred to as weight in previous studies. We change it here to avoid confusion with our weight function ω introduced earlier. Starting with unit loads, at each iteration, it chooses a subset of these separators according to their current loads, such that the resulting grid partitioning Π_G serves as an ε net. If Π_G already satisfies the error requirement (i.e., ϵ), the algorithm returns Π_G. Otherwise, it chooses a grid rectangle at random with probability proportional to its weight and doubles the loads of all separators that intersect this grid rectangle. The intuition is that if a grid rectangle has high weight, then those grid lines cutting it are more likely to be chosen for the grid partitioning. Careful analysis can show that the expected number of iterations can be bounded. Further details of the algorithm are omitted here. Interested readers can refer to [15].

C. Putting Everything Together

The size-reduction step produces a "compressed" $p \times q$ array G, with each element $G[u][v]$ representing a grid rectangle R_{uv} of the original array A, for any $1 \leq u \leq p$, $1 \leq v \leq q$, and $G[u][v] = \mu(R_{uv}) = \max_{(i,j) \in R_{uv}} A[i][j]$.

Let T be a subarray of G and $\mu(T) = \max_{(u,v) \in T} G[u][v]$. We define the compressed weight of T as

$$\hat{\omega}(T) = \sum_{(u,v) \in T} (\mu(T) - G[u][v]) \cdot |R_{uv}|$$

where $|R_{uv}|$ is the number of elements contained in the grid rectangle R_{uv} of the original array A. Note that this compressed weight is in similar form as the weight function $\omega(R)$ defined in Section II-A, except that there is an additional multiplier $|R_{uv}|$ here because unlike $A[i][j]$, which corresponds to single cell, $G[u][v]$ corresponds to group of cells contained in R_{uv}.

Let $\Pi = \{T_1, \ldots, T_l\}$ be a partitioning of G, it can also be considered as a partitioning over the original array A. Recall that Π_G is the grid partitioning of A induced by the compressed array G, it is easy to see that the compressed weight of Π, $\hat{\omega}(\Pi) = \sum_{1 \leq h \leq l} \hat{\omega}(T_h)$, is in fact the weight increase from $\omega(\Pi_G)$ to $\omega(\Pi)$, i.e., $\omega(\Pi) = \omega(\Pi_G) + \hat{\omega}(\Pi)$. To satisfy overall power penalty bound δ, we now require that $\omega(\Pi_G) + \hat{\omega}(\Pi) \leq \delta$, i.e., $\hat{\omega}(\Pi) \leq \delta - \omega(\Pi_G)$, where $\omega(\Pi_G)$ is output by Step 1).

We can now carry over the algorithm from Section II-B, running DP-Alg on the compressed $p \times q$ array G. The only difference is in how the base cases are computed, as the weight function is slightly different here.[6] However, as the weight $\hat{\omega}(T)$ of any subarray T of G can also be computed in $O(1)$ time, after a similar $O(pq)$ time/space preprocessing on array G, we can carry over the complexity result from Theorem 1 and implement the second step in $O(pq + (p + q)p^2q^2k^2 \log k)$ time and $O(pq + p^2q^2k)$ space. Together with Lemma 3, we have the following.

Theorem 2: The TS-Alg runs in $\tilde{O}(nm + (p + q)p^2q^2k^2)$ time and $O(nm + p^2q^2k)$ space, where $\tilde{O}$ hides some logarithmic terms. The value of p and q depends on the parameter $\epsilon \leq \delta$ in the first step. In practice, k is small, and $p \ll m$, $q \ll n$, in which case, our TS-Alg runs in roughly $O(nm)$ time.

IV. Experimental Results

A. Experiment Setup and Snapshots of Results

We perform our experiments with a set of industry designs on 64-bit Linux machines (CPU: 1.95 GHz, Memory: 11.7 GB). For each design, the experiment is carried out in the following steps.

We use the Cadence's commercial tool SoC Encounter [17] to do timing-driven placement, timing optimization, and timing analysis. Then, we assign a voltage to each standard cell according to its worst slack.[7] We give four different levels of voltages, each corresponding to a different range of worst slacks. The smaller the slack, the higher the voltage.

We transform the standard cell placement and associated voltage requirement into an input array, as described in Section II-A.

We calculate the maximum power penalty (denoted as $\delta_{\max}$), which is the total power penalty when all cells are raised to the highest required voltage on the entire chip.

[5] $O(nm)$ time/space is spent on preprocessing so that the weight $\omega(R_{uv})$ of any grid-rectangle R_{uv} can be computed in $O(1)$ time.

[6] We remark that the new weight function $\hat{\omega}(\Pi)$ is also superadditive.

[7] Strictly speaking, to reduce voltage on the noncritical cells without degrading the overall circuit performance, the voltage on each cell c should be assigned with regard to other cells on the same paths as c—because they share the slack with c—instead of being assigned in isolation. In fact, our recent work [18] directly addresses this problem and gives a voltage assignment algorithm that strictly respects the timing constraints. However, in this paper, since voltage assignment is not the focus—the main focus of this paper is that given a voltage assignment, how to form a voltage-island partitioning to reduce level shifting and design cost without introducing timing violations—so in the experiments, we only use the simple worst slack-based heuristic for the voltage assignment to produce input data for the voltage-island-partitioning algorithm. However, this heuristic has reasonably good correlation with a completely time-constrained solution so as to produce meaningful input data here.

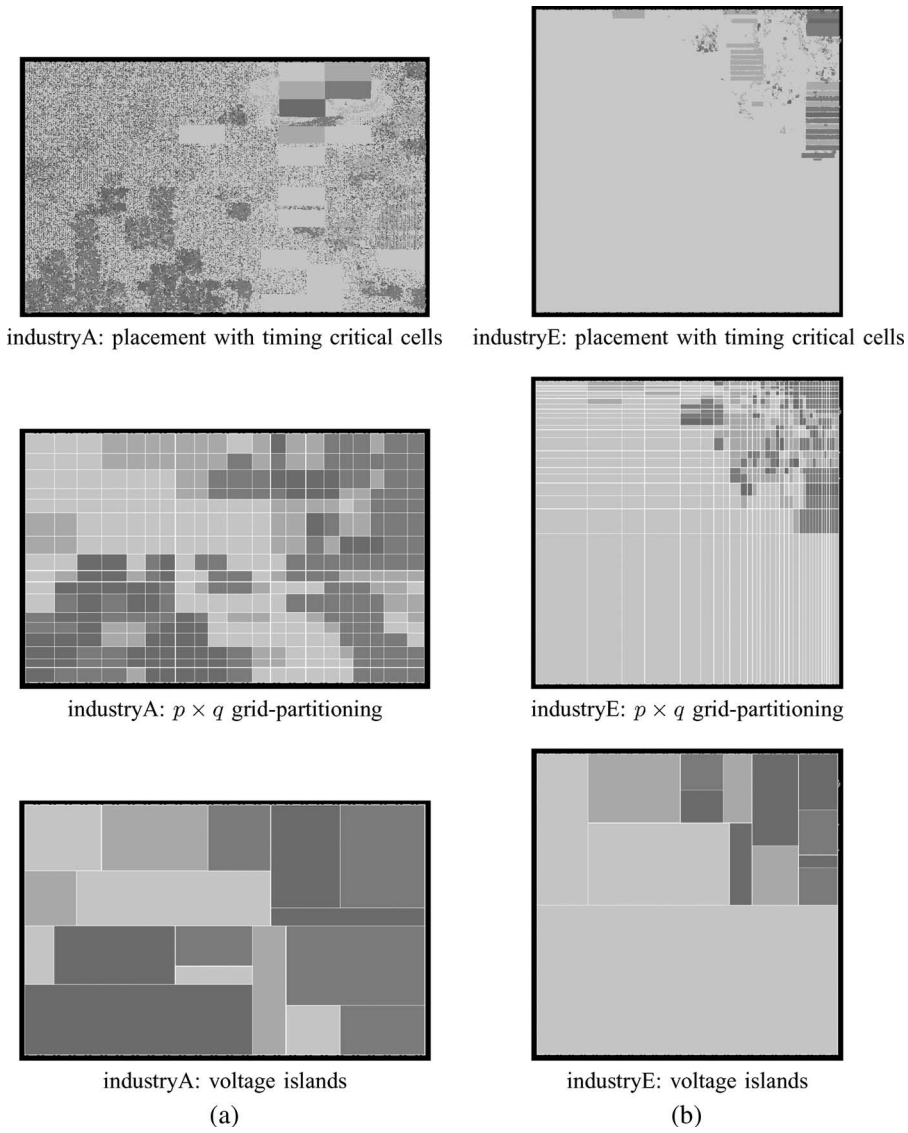


Fig. 4. Snapshots of the experiment results. (a) Design industryA. (b) Design industryE.

We give some reasonable bounds on the total amount of power penalty, each corresponding to a certain percentage of the maximum power penalty $\delta_{\max}$, and apply our TS-Alg to generate minimum number of voltage islands within each power-penalty bound.

Snapshots: First, we give some visual results of our TS-Alg to demonstrate its effectiveness. We will present quantitative results in the subsequent section. Fig. 4(a) shows the voltage islands generated from two industry designs. For each design (one column), the top picture shows the placement with timing-critical cells in dark colors (the darker a cell, the higher voltage is needed). The middle picture shows the $p \times q$ grid partitioning generated by the size-reduction step. Note that the voltage-distribution information is well preserved by the variable-sized grids. The last picture shows the generated voltage islands.

B. Comparison With Other Approaches

To demonstrate the efficiency of our TS-Alg, we compared it with two alternative approaches (one of them is commonly used in industry; the other is developed by us as a straightforward improvement over the industry approach).

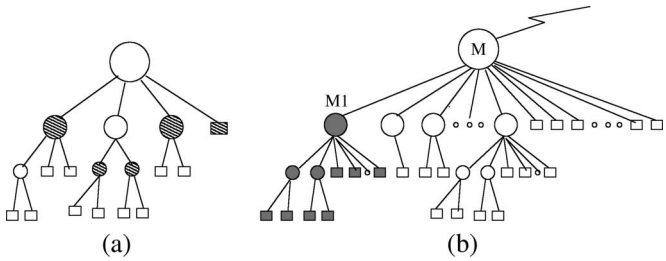


Fig. 5. Logical hierarchical tree. (a) Shaded modules/cells correspond to one possible partitioning. (b) Node M with high fan-out.

Outline of alternative approaches: The first one is the logical-boundary-based approach mentioned earlier in Fig. 1(b) and (c), where each grouped module or cell in the logical hierarchical tree forms an individual voltage island [Fig. 5(a)]. Currently, this approach is commonly used in practice.

This approach is often very inefficient due to the high fan-out of modules in the logical hierarchy tree. For the example in

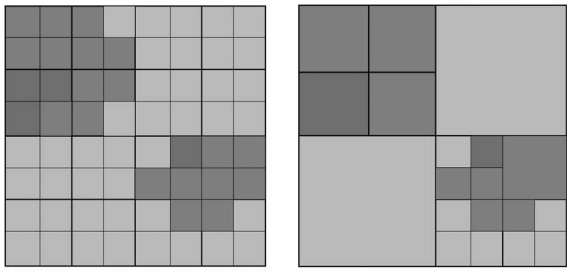


Fig. 6. Quadratic tree. (a) Input array (darker color indicates higher voltage value). (b) After some combinations.

Fig. 5(b), module M_1 has been marked high voltage due to its internal timing-critical cells. The rest of the cells in the subtree rooted at M are noncritical. If we do not group module M, each child of M will become an individual voltage island, causing too many fragments. If we group module M, the entire module will be raised to the high voltage that is required on M_1, causing too much power consumption. Such a case is very common in real designs.

A natural way to improve the logical-boundary-based approach is to substitute the high-fan-out hierarchical tree with nonlogical-boundary-based standard quad-tree. The leaves of the quad-tree are the grid cells in the input array, and each node in this tree corresponds to a possible region (a subarray). Given an upper bound δ for power penalty, the goal is to find a set of appropriate nodes from the tree that form a partitioning of the input array. The way to obtain such a partitioning is by a greedy bottom–up merging approach. In particular, we mark a node white if it has not been merged, black otherwise. A node is called a candidate for merging if it is white while all its four children are black. Furthermore, given a node R in quad-tree with its four children R_i, $i = 1, \ldots, 4$, define the cost of merging R_i to be $C(R) = \omega(R) - (\omega(R_1) + \omega(R_2) + \omega(R_3) + \omega(R_4))$. This corresponds to the power penalty resulted from combining the four subregions into R. Now, in order to compute a partitioning, we start with a tree where all nonleaf nodes are white. At any time, we choose the candidate with the smallest cost (by using a priority queue). The process will terminate when the total weight of the resulting partitioning exceeds the given upper bound δ. We refer to this algorithm as QT-Alg. The overall time complexity is $O(nm \log(nm))$ and space complexity is $O(nm)$. Fig. 6 illustrates QT-Alg.

Comparison with different power penalty bounds: In the left column of Fig. 7, we compare the output size of our TS-Alg with QT-Alg and the logical-tree-based algorithm on each industry design with different power-penalty bounds (the size of the designs are shown in Table I). Clearly, our TS-Alg outperforms the other two significantly and consistently on all designs in terms of the number of voltage islands obtained.[8] Between the two alternatives, QT-Alg is generally significantly better (with one exception on design industryH, where the logical hierarchy happens to become more advantageous than the physical hierarchy for the voltage-island merging). In the right column, we also show the actual amount of power penalty caused by the voltage-island grouping by each algorithm. All the values are below the given power bound. The amount from the TS-Alg is slightly closer to the bound, because it optimizes more aggressively toward the objective of the smallest number of voltage islands.

Comparison with selected power-penalty bound: As the output size of QT-Alg is much closer to our TS-Alg than the logical-tree-based algorithm, we compare TS-Alg with QT-Alg more explicitly in this section by listing some data from Fig. 7 in Table I. Here, for each design, we only pick a particular power-penalty bound such that the number of voltage islands being generated is within a desired range.[9] We let the range be around 20, which is roughly an upper bound in current practical designs.

Additionally, we also show the comparison of running time in the table. The time complexity of TS-Alg has two terms: $O(nm)$ and $\tilde{O}((p+q)p^2q^2k^2)$. The first term comes from preprocessing the input array for later computation of the weight of any subarray. Since this preprocessing step is needed by both TS-Alg and QT-Alg, we omit it from the running time presented.[10] Other than this preprocessing time, the running time of TS-Alg is only output-sensitive, depending on p, q, and k, while QT-Alg still has a $O(nm \log(nm))$ running time.[11] This explains why TS-Alg has larger running time on the first three small designs in Table I: Because $p \times q$ after Step 1) in these cases is larger than that of later cases. For all practical data, we test with all practical numbers of voltage islands, and p and q are small regardless of the size of the input design (see, for example, $p \times q$ does not increase in Table I with the input size). This means that TS-Alg scales well with increasing size of the input design! Furthermore, as k decreases, the running time advantage of TS-Alg over QT-Alg becomes even more significant, because the running time of TS-Alg decreases with k, while that of QT-Alg remains roughly the same (as it is not output sensitive). This is demonstrated in Table II, where we choose k around ten, probably a more realistic number in current designs. Note again that TS-Alg also beats QT-Alg significantly and consistently in terms of the quality of result.

C. Comparison of DP-ALG and TS-ALG

The DP-Alg from Theorem 1 is impractical for large data size, due to its huge memory requirement. Therefore, we are

[8]We do not obtain result from the logical-tree-based algorithm on a few designs (industryB, industryC, and industryD), because these designs do not have logical hierarchy (they are all flat).

[9]As stated earlier, our algorithm works for both of the dual-optimization problems. Our discussion and implementation is focused on minimizing number of voltage islands bounded by power penalty; however, it can be adapted to directly solve the other problem with $\log(k)$ faster running time.

[10]Also, since the preprocessing time is $O(nm)$, it will be smaller than the $O(nm \log(nm))$ running time of QT-Alg, especially for large design, this log-factor can be quite large! Therefore, omitting it will not change our comparison, while helping to clarify things.

[11]We remark that the logical-tree-based algorithm also has a $O(nm \log(nm))$ running time. Its base of the logarithm is much larger than that of the QT-Alg, due to the high fan-out of modules in the logical hierarchy tree. As a result, its running time is much smaller than the QT-Alg. Quantitatively, the logical-tree-based algorithm runs in less than 1 s on all our tested designs.

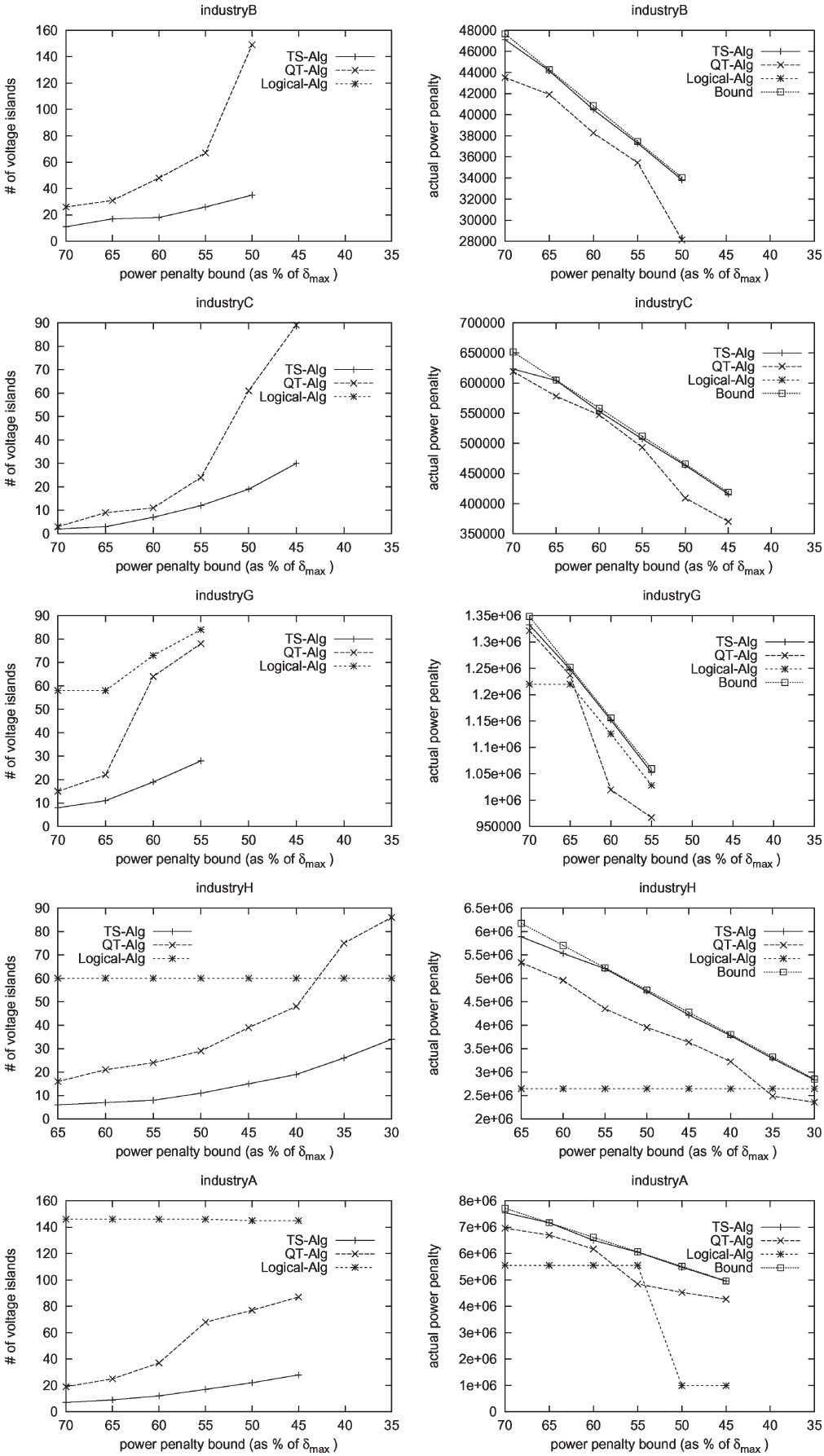


Fig. 7. Comparing different algorithms on the same design.

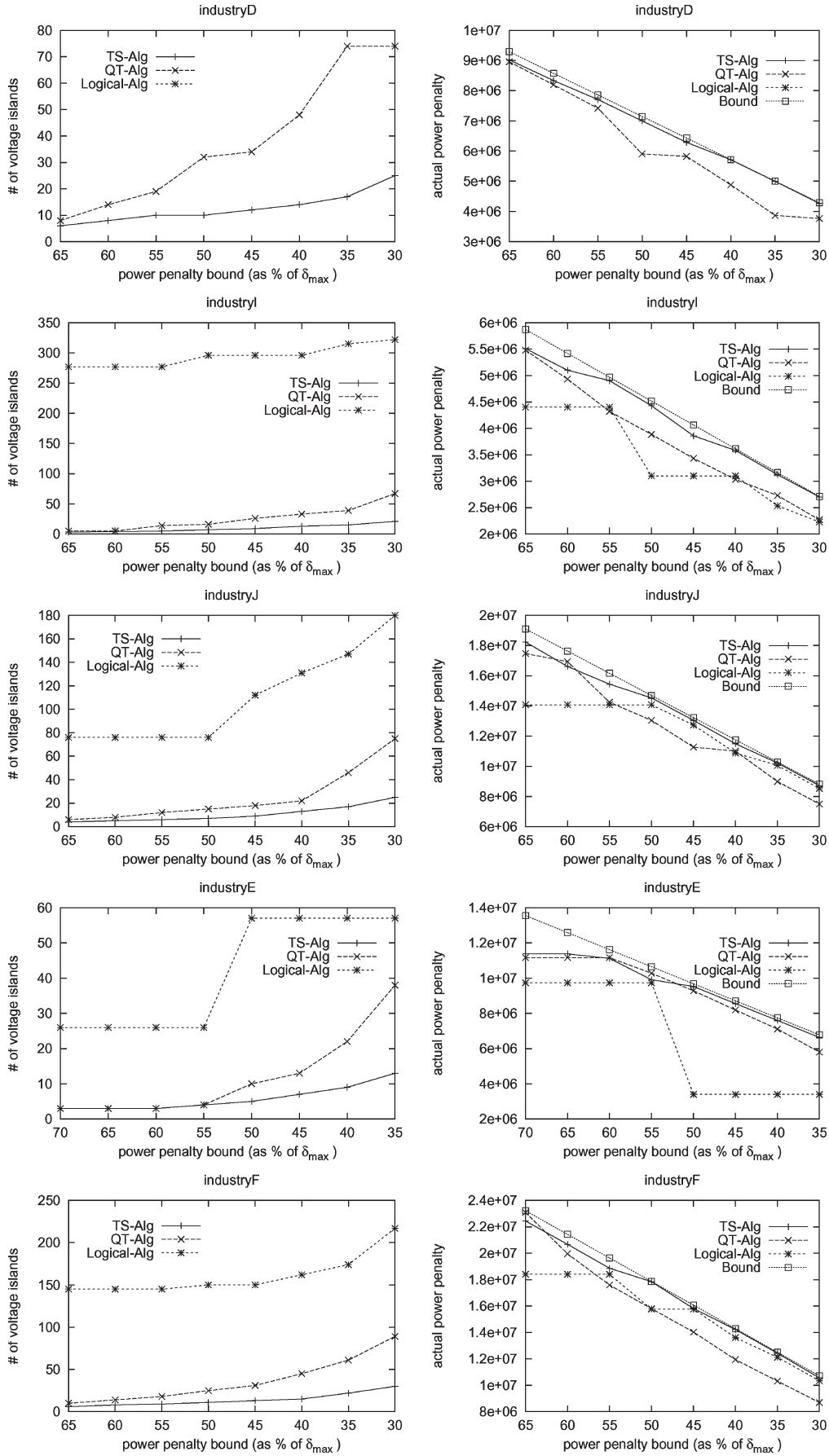


Fig. 7. *(Continued.)* Comparing different algorithms on the same design.

TABLE I
COMPARISON OF TS-ALG AND QT-ALG WITH OUTPUT # OF VOLTAGE ISLANDS AROUND 20

Design			Power	Grid Size	# of VI{k}		Runtime(s)	
Name	# of Cells	Array Size{$m \times n$}	Bound	in TS {$p \times q$}	TS	QT	TS	QT
industryB	5926	79 x 790	60%	26 x 26	18	48	58	1
industryC	43677	161 x 2860	50%	21 x 34	19	61	125	2
industryG	76406	230 x 3504	60%	26 x 26	19	64	105	3
industryH	243188	694 x 7852	40%	13 x 17	19	48	9	20
industryA	317752	732 x 8793	55%	17 x 21	17	68	16	25
industryD	397940	1300 x 8270	35%	21 x 26	17	74	49	33
industryI	342113	1199 x 8931	35%	17 x 21	15	39	11	36
industryJ	737555	1372 x 12350	35%	21 x 21	17	46	32	64
industryE	352060	2144 x 17159	35%	26 x 34	13	38	47	134
industryF	306326	2652 x 20978	40%	13 x 17	15	45	5	222

TABLE II
COMPARISON OF TS-ALG AND QT-ALG WITH OUTPUT # OF VOLTAGE ISLANDS AROUND TEN

Design	Power	Grid Size	# of VI		Runtime(s)	
Name	Bound	in TS	TS	QT	TS	QT
industryB	70%	13 x 17	11	26	2	1
industryC	55%	21 x 21	12	24	7	2
industryG	65%	21 x 21	11	22	6	3
industryH	50%	9 x 13	11	29	1	20
industryA	60%	13 x 17	12	37	3	25
industryD	50%	9 x 13	10	32	2	33
industryI	45%	9 x 13	9	26	2	35
industryJ	45%	9 x 13	9	18	2	63
industryE	40%	17 x 21	9	22	4	131
industryF	50%	9 x 13	11	25	2	211

unable to obtain its result on any of our tested designs. This is one of the main reasons why we developed the TS-Alg.

However, in order to study the gap between DP-Alg and TS-Alg, we make some special modification to the input data so that the result from DP-Alg can be obtained. We then apply TS-Alg on the modified data too and compare the results with those from DP-Alg.

In particular, we first test the memory of our machine and roughly find the upper limit of the input data size for DP-Alg to run[12]: $m = n = 40$, $k = 15$. We then partition the input array A by randomly generated 40×40 nonuniform grids (we will explain later why we use nonuniform grids) and obtain a corresponding 40×40 array S. Each element of S is the average or maximum[13] of all the elements of A in the corresponding grid rectangle. Then, we run DP-Alg on S to obtain the optimal slicing partitioning. The power bound is selected such that the number of voltage islands in the resulting partitioning is roughly between 10 and 15.

In general, a natural next step would be to apply TS-Alg on the same array S that we performed DP-Alg and compare their results. However, in this case, the size of S is too small for the comparison to be meaningful. As can be expected, not too much reduction of array size will happen in the first step, and the result from TS-Alg will be quite close to that of DP-Alg.

Nonetheless, we still wish to obtain some indication of how well TS-Alg performs with respect to DP-Alg. We, thus, conduct the following experiments. Instead of using the smaller array S to run TS-Alg, we "project" S back to the original array to obtain a modified array M of the original size. In particular, the value of each element $M[i][j]$ will be the value of the element of S which corresponds to the grid rectangle that contains $M[i][j]$. Due to the "identicalness" of data distribution between S and M, it is easy to see that the optimal slicing partitioning of the former is also the optimal for the latter. We now apply TS-Alg on this modified array M. Although M has a special data distribution that may bias the behavior of TS-Alg in the right direction, we believe that with the specific $p \times q$ gridding algorithm we currently use, the randomness and nonuniformity of the way we previously obtain S alleviate this problem. Hence, the results of TS-Alg on such modified array still reflect its performance on arbitrary arrays.

The comparison of the results of TS-Alg with DP-Alg is shown in Table III. For each design, we control the power bound of the size-reduction step so as to generate different $p \times q$ size (three sets). The largest $p \times q$ size is decided by the memory limit. From the table, clearly, the larger the $p \times q$ size, the better the result; the gap between the best TS-Alg result and the DP-Alg result is quite tolerable. In fact, with the availability of machines with larger memory, we may hopefully further increase the $p \times q$ size and reduce the gap. Thus, we can conclude that the $p \times q$ gridding does not greatly deteriorate the optimal solution and the TS-Alg is still able to find a close-to-optimal solution.

D. Study of the Effect of Level Shifters

A common issue in MSV designs is that level shifters need to be inserted at the boundaries between low $V_{\rm dd}$ and high $V_{\rm dd}$, causing extra area/delay/power penalty. This is one of the main reasons that motivated us to try to group the cells into a small number of voltage islands (the other reason is to reduce the design cost of the power network). Because as such, level shifters will only need to be inserted at the boundaries of the voltage islands. The demand can, thus, be significantly reduced. To further study the effect of the level shifters in MSV designs, we count the average number of level shifters needed per timing path after the voltage-island partitioning and calculate their delay penalty accordingly.

More specifically, for each design, we select the first 1000 worst timing paths (with different end points); we also generate a set of different voltage-island partitionings under different

[12]The actual limit also depends on the design size (# of cells and nets), because the design itself will take up certain amount of memory.

[13]We decide the function on a case-by-case basis so as to maximize the similarity of data distribution between A and S.

TABLE III
COMPARISON OF DP-ALG AND TS-ALG

Design	Power	DP-Alg		TS-Alg (set 1)			TS-Alg (set 2)			TS-Alg (set 3)		
Name	Bound	# VI	Power	$p \times q$	# VI	Power	$p \times q$	# VI	Power	$p \times q$	# VI	Power
industryB*	4.17	11	4.12	17 x 21	13	4.13	26 x 26	12	4.14	42 x 42	12	4.07
industryC*	1.55	13	1.47	26 x 26	17	1.54	26 x 34	16	1.53	34 x 42	15	1.51
industryG*	3.23	12	3.16	26 x 34	14	3.22	34 x 34	13	3.17	34 x 42	13	3.14
industryH*	3.16	10	3.08	21 x 34	11	3.03	34 x 34	11	3.03	34 x 42	11	3.00
industryA*	3.36	11	3.13	21 x 26	13	3.26	21 x 34	12	3.21	34 x 34	11	3.28
industryD*	9.23	12	8.67	26 x 26	15	8.74	26 x 34	14	9.23	34 x 34	14	8.68
industryI*	1.83	12	1.77	21 x 21	15	1.83	26 x 26	14	1.82	34 x 34	14	1.76
industryJ*	7.07	11	7.02	17 x 21	14	7.00	26 x 26	13	6.91	34 x 34	13	6.75
industryE*	3.19	10	3.09	17 x 26	13	3.00	26 x 26	12	3.11	34 x 34	11	3.06
industryF*	5.75	11	5.75	21 x 21	14	5.64	21 x 26	14	5.60	26 x 26	13	5.74

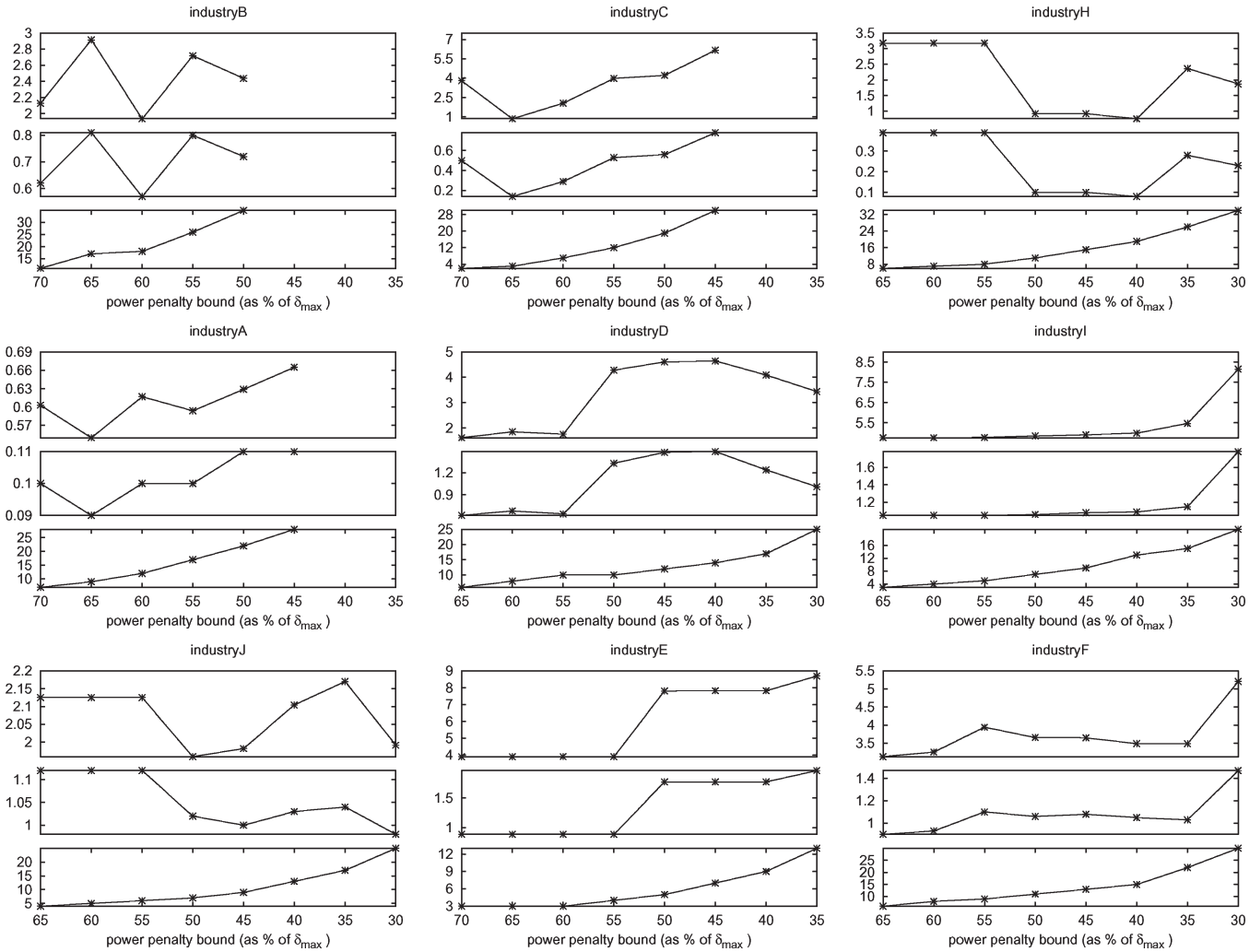


Fig. 8. Effect of level shifters. X-axis: Power-penalty bounds for voltage-island partitioning. Y-axis: Lower figure—the number of voltage islands in each partitioning; middle figure—the average number of level shifters per path; upper figure—the average percentage of the level-shifter delay in the path required time.

power-penalty bounds. Then, for each of the voltage-island partitionings, we count the number of level shifters needed on each selected timing path, which is the number of nets on the path that cross voltage-island boundaries (from low $V_{\rm dd}$ to high $V_{\rm dd}$). For each selected timing path, we also compute the average delay of buffers/inverters on the path. Then, assuming that the delay of each level shifter is twice the average delay of buffers/inverters, we calculate the total delay of all level shifters on the path and its percentage in the path-required time. We then compute the average over all selected paths. The results are shown in Fig. 8. For each design, each point along the x-axis corresponds to a different voltage-island partitioning generated under the given power-penalty bound. For each voltage-island partitioning, the lower figure shows the number of voltage islands, the middle figure shows the average number of level shifters per path, the upper figure shows the average percentage of the level-shifter delay in the path-required time.

From these figures, we can make the following observations.

First, in general, the average number of level shifters (and the corresponding delay) increases with the number of voltage islands, for example, industryA and industryC. Clearly, this is because the chance that a timing path crosses the boundaries of voltage islands becomes more. However, since we only count the first 1000 worst paths, the average number of level shifters counted also depends on where the voltage-island boundaries are with respect to these 1000 paths. This explains why the average number of level shifters does not correspond to the number of voltage islands in some designs, for example, industryB and industryJ. On the other hand, this suggests a future improvement for the voltage-islands partitioning to reduce the number of level shifters: Adding the number of times the timing paths cross the voltage-islands boundaries as a second-order minimization objective. This can be easily integrated into the DP-Alg.

Second, for most designs, the average number of level shifters is quite small and their total delay only takes a small portion of the path-required time.[14] This means that, as a future improvement, we can estimate such delay and reserve it in advance before doing voltage assignment and voltage-island partitioning. Besides, with the availability of more physical information of the level shifters to be used in each MSV design, their area and power penalty could also be estimated and reserved in advance for placement and for setting the power bound, respectively.

Third, some designs with a number of small voltage islands inside a small area, for example, industryE [see the upper right region of the bottom figure in Fig. 4(a) and (b)], tends to have a higher average number of level shifters. Intuitively, this is because the timing paths will cross the voltage-island boundaries more frequently in that area. As a future improvement, we may put some additional constraints on the voltage-island partitioning to avoid such cases. The DP-Alg can take such constraints fairly easily.

V. Conclusion

Reducing power consumption is essential in current chip designs. MSV has previously been used to reduce the overall power consumption without degrading the entire circuit performance, by applying high voltage on timing-critical cells and low voltage on noncritical cells. However, the resulting fragmented power network causes extra design cost, and grouping cells into a practical number voltage islands is, therefore, desired. Logical boundaries are largely used today in such groupings but they are almost always far from optimal for the voltage islands. In this paper, we initiated the study of the problem of balancing power consumption and power-network fragmentation. In particular, we investigated the problem of partitioning a placement region into a small number of voltage islands, while keeping the overall voltage supply low. We formulated this problem as a VPP and presented an efficient TS-Alg for solving it. We have implemented and applied our TS-Alg on a wide selection of real industry designs, and it has been shown to be fast, scalable, and produce superior results compared with the logical-boundary-based approach commonly used today.

[14]For the same number of level shifters, the percentage of their delay to the path-required time varies from design to design, depending on the path-required time, as well as the delay of the buffer that we use to estimate the level-shifter delay.

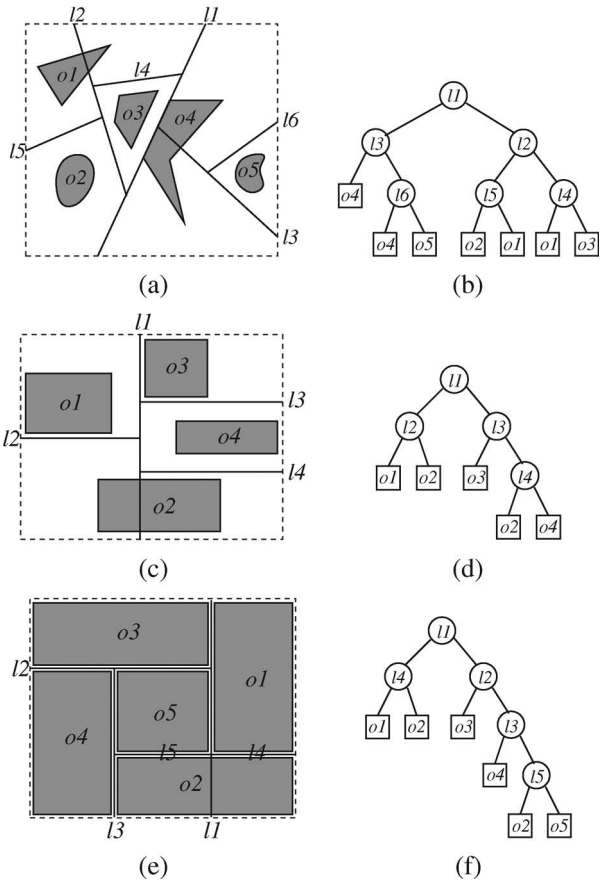


Fig. 9. (a) BSP. (b) BSP tree for (a). (c) Orthogonal BSP. (d) BSP tree for (c). (e) Orthogonal BSP whose rectangles cover the underlying space completely. (f) BSP tree for (e).

Appendix A
Orthogonal BSP

The definition of the BSP for a collection of disjoint geometric objects in the two-dimensional plane can be illustrated by Fig. 9(a) [19]. The BSP is obtained by recursively splitting the plane with a line: First, we split the entire plane with l_1; then, we split the half-plane above l_1 with l_2 and the half-plane below l_1 with l_3, and so on. The splitting lines not only partition the plane, they may also cut objects into fragments. The splitting continues until there is only one fragment left in the interior of each region. This process is naturally modeled as a binary tree (BSP tree), as shown in Fig. 9(b). Each leaf of this tree corresponds to a face (leaf region) of the final subdivision. Each internal node corresponds to a splitting line.

If the geometric objects are all rectangles whose boundaries are parallel to the x-axis or y-axis of the coordinate plane, we call them axis-aligned rectangles, and if the splitting lines are also all parallel to the x-axis or y-axis, then we call the corresponding BSP as orthogonal BSP. Fig. 9(c) shows an example. The corresponding BSP tree is shown in Fig. 9(d).

The size of a BSP is defined as the number of leaves in the BSP tree (which is also the number of leaf regions in the BSP). Reference [12] has proven the following upper bounds on the size of the orthogonal BSP: $3n-1$ in general and $2n-1$ if the rectangles cover the underlying space completely [an example is shown in Fig. 9(e)], where n is the number rectangles. For the orthogonal BSP in Fig. 9(c), $n=4$ and the size is five. For the orthogonal BSP in Fig. 9(e), $n=5$ and the size is six.

Appendix B
Constant Time Weight Computation

For a given array A, we first describe how to do a $O(nm)$ time/space preprocessing so that $sum(R)=\sum_{(i,j)\in R} A[i][j]$ for any rectangle R can be computed in $O(1)$ time.

Define two $m \times n$ arrays S and T

$$S[i][j] = \sum_{1\le c\le j} A[i][c], \text{for } 1\le i\le m, 1\le j\le n$$

$$T[i][j] = \sum_{1\le r\le i} S[r][j], \text{for } 1\le i\le m, 1\le j\le n.$$

Then, using the prefix-sum algorithm, both S and T can be computed in $O(nm)$ time/space as follows:

$$S[i][j] = \left\{ \begin{array}{ll} A[i][j], & j=1 \\ S[i][j-1]+A[i][j], & j>1 \end{array} \right\}$$

$$T[i][j] = \left\{ \begin{array}{ll} S[i][j], & i=1 \\ T[i-1][j]+S[i][j], & i>1 \end{array} \right\}.$$

Let $R=A[l\cdots r, b\cdots u]$ be a rectangle of A, then $sum(R)$ can be computed as

$$sum(R)=T[r][u]-T[l-1][u]-T[r][b-1]+T[l-1][b-1]$$
$$(T[i][j]=0 \text{ for } i=0 \text{ or } j=0).$$

Next, we describe how to extend it to compute the weight $\omega(R)$ of any rectangle R in $O(1)$ time/space, assuming that $A[i][j]$ only takes a limited number of discrete values (corresponding to the total number d of different voltage levels). Recall that

$$\omega(R)=\sum_{(i,j)\in R}(\mu(R)-A[i][j])=\mu(R)\cdot|R|-\sum_{(i,j)\in R}A[i][j]$$

where $\mu(R)=\max_{(i,j)\in R} A[i][j]$ and $|R|$ is the number of elements in A.

We only need to show the computation of $\mu(R)$ in $O(1)$ time/space. This can be done by creating an auxiliary array B_v for each voltage level v

$$B_v[i][j] = \left\{ \begin{array}{ll} 1, & A[i][j]=v \\ 0, & A[i][j]\neq v \end{array} \right\}.$$

Obviously, by doing similar preprocessing, $sum_{Bv}(R)=\sum_{(i,j)\in R} B_v[i][j]$ can be computed in $O(1)$ time/space for any rectangle R. The maximum v satisfying $sum_{Bv}(R)>0$ can be found by a $O(\log(d))$ time binary search. Because d is a constant, therefore, $O(\log(d))=O(1)$.

Appendix C
ε-Net and Analysis

A set system $(X, \mathcal{R})$ is a set X along with a collection $\mathcal{R}$ of subsets of X. A hitting set of a set system $(X, \mathcal{R})$ is a subset $H\subseteq X$ such that H has a nonempty intersection with every set S in $\mathcal{R}$. The hitting-set problem is to find a hitting set of the smallest size. It is equivalent to the set cover problem and they are both NP-complete.

A subset $H\subseteq X$ is an ε-net of $(X, \mathcal{R})$, if it has a nonempty intersection with every set S in $\mathcal{R}$ for which $|S|\ge\varepsilon\cdot|X|$. Note that an ε-net is a hitting set of the set system $(X, \mathcal{R}_\varepsilon)$, where $\mathcal{R}_\varepsilon=\{S\in\mathcal{R}||S|\ge\varepsilon\cdot|X|\}$. For a given additive weight function[15] $w: X\to\mathbb{R}^+$, a subset $H\subseteq X$ is an ε-net of $(X, \mathcal{R})$ with respect to weight w, if it has a nonempty intersection with every set S in $\mathcal{R}$ for which $w(S)\ge\varepsilon\cdot w(X)$. A net finder for $(X, \mathcal{R})$ is an algorithm that, given ε and a weight function w, returns an ε-net of $(X, \mathcal{R})$ with respect to weight w.

For a set system $(X, \mathcal{R})$ with finite VC-dimension (see [14] for the definition), [14] gives an algorithm to find a hitting set of almost optimal size in polynomial time. Suppose the size of the optimal hitting set of $(X, \mathcal{R})$ is c^*. The algorithm performs a doubling search for the value of c^*, staring from $c=1$. For each guess c of c^*, it tries to find a hitting set of size $s(2c)$ by using a strategy based on the notion of "survival of the fittest" from the evolutionary biology.[16] Intuitively, one wants to simulate the growth of a population where some elements are advantaged, because they hit more sets than others. This is done by putting weights on the elements of X, as follows. Initially, all weights are uniform (unit weights). Then, the algorithm iterates as follows: 1) Invoke the net finder[17] to select an ε-net H with respect to current weights (where $\varepsilon=1/2c$); 2) verify whether H is a hitting set; 3) if H is not a hitting set, then find a set S in $\mathcal{R}$ that is not hit by H, and double the weights of the elements in S.

Following the arguments of Clarkson [16], [14] proves that if there is a hitting set H of size c, then the number of iterations in the above algorithm will not exceed a certain bound, $4c\log(n/c)$, where $n=|X|$. Briefly speaking, the proof is based on the fact that $w(H)$ would grow faster than $w(X)$ through the iterations, and since $H\subseteq X$, clearly, $w(H)\le w(X)$. This implies that the algorithm has to terminate before $w(H)$ exceeds $w(X)$.

Hence, if the number of iterations exceeds this bound, then we know that there is no hitting set of size c; in other words, the current guess c is too small, and we will double its value. The doubling search stops until a hitting set is found for a certain value of c. Clearly, $c\le 2c^*$, so the hitting set returned by the algorithm is of size at most $s(4c^*)$.

[15] The term additive means that $w(Y)=\sum_{y\in Y} w(y)$.

[16] s is a nondecreasing function.

[17] A polynomial time net finder can be implemented by the algorithm of [20].

REFERENCES

[1] J. Buurma and L. Cooke, "Low-power design using multiple V_{TH} ASIC libraries," SoC central, Aug. 2004. [Online]. Available: http://www.sinavigator.com/Low_Power_Design.pdf

[2] F. Fallah and M. Pedram, "Standby and active leakage current control and minimization in cmos vlsi circuits," *IEICE Trans. Electron.—Special Section Low-Power LSI and Low-Power IP*, vol. E88-C, no. 4, pp. 509–519, Apr. 2005.

[3] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanazawa, M. Ichida, and K. Nogami, "Automated low-power technique exploiting multiple supply voltages applied to a media processor," *IEEE J. Solid-State Circuits*, vol. 33, no. 3, pp. 463–472, Mar. 1998.

[4] K. Usami and M. Horowitz, "Clustered voltage scaling technique for low-power design," in *Proc. Int. Symp. Low Power Des.*, 1995, pp. 3–8.

[5] C. Chen, A. Srivastava, and M. Sarrafzadeh, "On gate level power optimization using dual-supply voltages," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 5, pp. 616–629, Oct. 2001.

[6] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn, "Managing power and performance for system-on-chip designs using voltage islands," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2002, pp. 195–202.

[7] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu, "Architecting voltage islands in core-based system-on-a-chip designs," in *Proc. Int. Symp. Low Power Electron. and Des.*, 2004, pp. 180–185.

[8] Virtual Silicon Technology, Inc., "Power islands: The evolving topology of SoC power management," Design & Reuse, Jul. 2004. [Online]. Available: http://www.us.design-reuse.com/articles/article9150.html

[9] S. Khanna, S. Muthukrishnan, and M. Paterson, "On approximating rectangle tiling and packing," in *Proc. 9th Annu. ACM-SIAM SODA*, 1998, pp. 384–393.

[10] R. H. Otten, "Automatic floorplan design," in *Proc. 19th ACM/IEEE Conf. Des. Autom.*, 1982, pp. 261–267.

[11] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. 23rd ACM/IEEE Conf. Des. Autom.*, 1986, pp. 101–107.

[12] P. Berman, B. Dasgupta, and S. Muthukrishnan, "Exact size of binary space partitionings and improved rectangle tiling algorithms," *SIAM J. Discrete Math.*, vol. 15, no. 2, pp. 252–267, 2002.

[13] S. Muthukrishnan, V. Poosala, and T. Suel, "On rectangular partitionings in two dimensions: Algorithms, complexity, and applications," in *Proc. 7th ICDT*, 1999, pp. 236–256.

[14] H. Brönnimann and M. T. Goodrich, "Almost optimal set covers in finite vc-dimension," *Discrete Comput. Geom.*, vol. 14, no. 4, pp. 463–479, 1995.

[15] S. Muthukrishnan and T. Suel, "Approximation algorithms for array partitioning problems," *J. Algorithms*, vol. 54, no. 1, pp. 85–104, Jan. 2005.

[16] K. L. Clarkson, "A Las Vegas algorithm for linear programming when the dimension is small," *J. ACM*, vol. 42, no. 2, pp. 488–499, Mar. 1995.

[17] "Cadence software manual: SoC encounter RTL-to-GDSII system," [Online]. Available: http://www.cadence.com/products/digital_ic/soc_encounter/index.aspx

[18] H. Wu, M. D. Wong, and I-M. Liu, "Timing-constrained and voltage-island-aware voltage assignment," in *Proc. 43rd ACM/IEEE Conf. Des. Autom.*, 2006, pp. 429–432.

[19] M. de Berg, M. van Kreveld, O. Overmars, and O. Schwarzkopf, *Computational Geometry—Algorithms and Applications*. Berlin, Germany: Springer-Verlag, 1997.

[20] H. Brönnimann, B. Chazelle, and J. Matousek, "Product range spaces, sensitive sampling, and derandomization," *SIAM J. Comput.*, vol. 28, no. 5, pp. 1552–1575, Apr./May 1999.

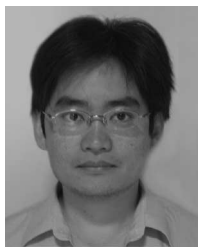
Huaizhi Wu received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1998, and the M.S. and Ph.D. degrees in computer engineering from the University of California, Santa Cruz, in 2000 and 2007, respectively.

She joined Silicon Perspective Corporation (SPC), Santa Clara, CA, in 2001. After SPC merged with Cadence Design Systems, Inc., San Jose, CA, in 2002, she was with the Digital IC Division until 2006. She is currently with Atoptech, Inc., Santa Clara. Her research interests include low-power design methodology, power/timing-driven placement, analytical placement, algorithm design, and combinatorial optimization.

Martin D. F. Wong (M'88–SM'04–F'06) received the B.Sc. degree in mathematics from the University of Toronto, ON, Canada, and the M.S. degree in mathematics from the University of Illinois, Urbana–Champaign. He obtained the Ph.D. degree in computer science from the University of Illinois, in 1987.

He was a Bruton Centennial Professor of computer sciences at the University of Texas, Austin. He is currently a Professor of electrical and computer engineering with the University of Illinois. His research interests are computer-aided design (CAD) of very large-scaled integrated circuits (VLSI), design and analysis of algorithms, and combinatorial optimization. He has published 300 technical papers and has graduated 32 Ph.D. students.

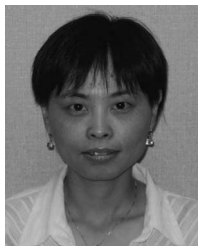
Dr. Wong is the recipient of the 2000 IEEE CAD Transactions Best Paper Award for his work on interconnect optimization. He is also the recipient of the best paper awards at Design Automation Conference (DAC), in 1986, and International Conference on Computer Design (ICCD), in 1995, for his work on floorplan design and routing, respectively. In 1994, his International Conference on Computer-Aided Design (ICCAD) paper on circuit partitioning has been included in the book "The Best of ICCAD—20 Years of Excellence in Computer Aided Design" published in 2002. He has served as Technical Program Chair, General Chair, and Steering Committee member for the annual Association for Computing Machinery International Symposium on Physical Design. He also regularly serves on the technical program committees of all major VLSI/CAD conferences (e.g., DAC, ICCAD, ISPD, DATE, ASPDAC). He has served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN from 2002 to 2005, and IEEE TRANSACTIONS ON COMPUTERS from 1995 to 2000. He has also served as a Guest Editor of four special issues for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. He is currently on the Editorial Board of Association for Computing Machinery TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS. He is an IEEE Distinguished Lecturer.



I-Min Liu received the B.S. and M.S. degrees in electronics engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1993 and 1995, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Texas, Austin, in 2000.

He was a Visiting Researcher at the University of California, Berkeley, and at Intel Corporation in the summer of 1998 and 1999, respectively. He joined Silicon Perspective Corporation (SPC), Santa Clara, CA, in 2000. Through the merger of SPC, he joined Cadence Design Systems, San Jose, CA, in 2002. Throughout his tenure with SPC/Cadence, he contributed to the development of physical prototyping and implementation technologies for multimillion transistor system-on-chip designs. Since 2005, he has been with Atoptech, Inc., a startup company in Santa Clara, to work on next generation electronic-design-automation solutions.

Dr. Liu is a member of Phi Tau Phi.



Yusu Wang received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1998, and the M.S. and Ph.D. degrees in computer science from Duke University, Durham, NC, in 2000 and 2004, respectively.

From 2004 to 2005, she was a Postdoctoral Fellow in the Geometric Computing Lab in Stanford University, Stanford, CA. She is currently an Assistant Professor in Department of the Computer Science and Engineering, Ohio State University, Columbus. Her research interests include computational geometry, computational topology, shape analysis, and visualization.

Dr. Wang is the recipient of the DOE Early Career Principal Investigator Award in 2006. She is a member of the Association for Computing Machinery Computer Society.