Optimal Post-Placement Voltage Island Generation under Performance Requirement

Abstract

High power consumption not only leads to short battery life for handheld 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 trade-off. In this work we propose a methodology on top of a set of algorithms to exploit non-trivial voltage island boundaries for optimal power versus design cost trade-off 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.

1 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 in a constant fight with balancing power and performance throughout the chip design cycle.

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

Power consumption generally breaks down into two sources: dynamic and static powers [5]. While static power comes from leakage current, dynamic power P_d is the result of device's switching activities, which can be represented by

$$P_d = kcv^2 f, \tag{1}$$

where k is switching rate, c is load capacitance, v is supply voltage, and f is clock frequency. Dynamic power dominates the total power consumption in today's logic design. 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 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 supply voltage, reducing supply voltage can significantly reduce active power consumption. Multi-supply voltage (MSV) is introduced to provide finer-grain power and performance trade-off. There are two types of MSV. In "row-based" type, there are interleaving high and low supply voltage standard cell placement rows. In "region-based" approach, circuits are partitioned into "voltage island" (or "power domain") where each voltage island occupies a contiguous physical space and operates at a supply voltage that meets the performance requirement [9, 7, 2].

Region-based design in current state of art is largely done manually and is primarily based on 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 non-optimal boundaries for supply voltages.

Although in theory only timing critical device needs high supply voltage, this naive thinking for maximum power reduction is not practical. When voltage islands become fragmented, there is an overhead in voltage shifting devices. Moreover, it is high cost to implement fragmented power networks as implementing such complex power network is not only a tedious work but will also take a lot of precious routing resource from design, which is not a good idea when per-metal-layer manufacturing cost is soaring as process migrates. We therefore wish to reduce this *design cost*.

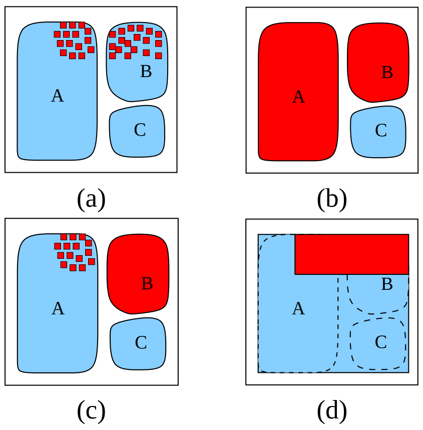


Figure 1: (a) Design with timing critical cells (small darker cells). (b) Power consumption too high. (c) Timing requirement not met for small cells in module A. (d) A solution with non-logical boundary

The example in Figure 1 shows how “natural” logical boundaries fails to provide a solution that meets all design requirements. There are three modules in the design, each of them only contains leaf cells. Both module A and B have timing critical cells that require high voltages (Figure 1(a)). If we want to guarantee performance with practical number of voltage islands while using only logical boundaries, then we will have a solution as shown in Figure 1(b), whose power consumption may exceed some given bound (suppose, for example, at most half chip area is allowed for high voltage). On the other hand, if we want to keep power consumption within given bound without generating fragmented voltage islands, then we will have to assign module A to a low-voltage voltage island, as shown in Figure 1(c), thereby sacrificing performance on those timing critical cells in A. However, if we use a non-logical boundary based on placement result, as shown in Figure 1(d), we can satisfy both the power consumption bound and the timing requirement at the same time without generating fragmented voltage islands.

In this work we propose a methodology on top of a set of algorithms to exploit the “non-natural” (non-logical) boundary in a design for optimal supply voltage partitioning capturing power versus design cost trade-off under performance (timing) requirement. Depending on each designer’s specific needs, the optimal trade-off can be explored by either one of the two dual optimization problems: maximally reduce power consumption within given number of voltage islands bound; or create minimally fragmented voltage islands within given power consumption bound. 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 contribution can be summarized as follows: To our best knowledge, we are the first to consider power versus design cost trade-off under timing requirement for voltage island generation problem. In particular, we exploit non-trivial voltage island boundaries to balance power consumption and power network fragmentation. We formulate this problem as voltage-partitioning problem (Section 2). This voltage-partitioning problem is NP-complete and we thus study approximation algorithms (i.e., algorithms with optimality guarantee) 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-size $p \times q$ gridding (Section 3). We show (Section 4) that our method is efficient and practical, as well as produces near-optimal voltage islands for a wide selection of industry data. Compared to the approach using simple 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.

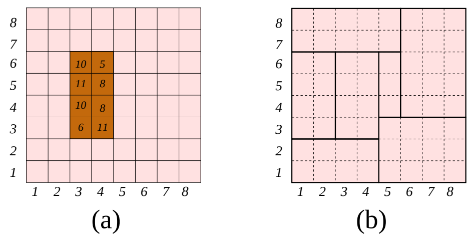


Figure 2: (a) A 9 by 9 array A, with a subarray (shadowed region) $R = \mathrm{A}[2 \cdots 4, 2 \cdots 6]$; $\mu(R) = 11$ and $\omega(R) = 19$. (b) A partitioning of A with seven rectangles.

2 Voltage-partitioning Problem

2.1 Problem definition

Let A be an n by m array, and $\mathrm{A}[x][y]$ the value of the element at position (x, y). We sometimes may also refer to an array as a *rectangle*. A subarray $\mathrm{A}[l \cdots r, b \cdots u]$ is simply the rectangular region in A with (l, b) (resp. (r, u)) as the bottom-left (resp. upper-right) vertex. 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 this partitioning. See Figure 2 for an illustration. Let $\mu(R) = \max_{(x,y) \in R} R[x][y]$ be the maximum value of all elements in a rectangle R. The *weight of* R is defined as

$$\omega(R) = \sum_{(x,y)\in R} (\ \mu(R) - R[x][y]\ ),$$

and the *weight of* Π, $\omega(\Pi)$, is simply the sum of the weights of all R_i's, for $1 \le i \le k$.

In the voltage partitioning problem, we can consider each standard cell placement region as a two-dimensional array A, induced by the underlying placement grid. $\mathrm{A}[x][y]$ is simply the *square* of the required voltage at the corresponding physical grid cell. We wish to subdivide the placement region into a small number of voltage islands (i.e., find a partitioning for A), where every cell in the same voltage island will eventually receive the same voltage. To make sure that the resulting voltage partitioning satisfies the timing requirement, we require that the voltage value assigned to a voltage island R should be at least the voltage associated with each grid cell contained in this voltage island (i.e., at least $\sqrt{\mu(R)}$). As dynamic power is proportional to the square of supply voltage (Eqn 1), $\sum_{(x,y)\in R} R[x][y]$ can be considered as the minimum power needed for all cells contained in R; while after merging them into one voltage island, the power consumption of R is increased by some value, which is its weight. To keep the overall power cost low, it is desirable to have the power increase (i.e., weight of the partitioning) below some threshold. We thus extract and formally define the voltage-partitioning problem as follows.

Definition 1 (Voltage-partitioning Problem, or VPP) *Given an $n \times m$ array A and an error threshold δ, among all partitions whose weight is smaller than δ, find the one with smallest size. Let $\kappa(\mathrm{A}, \delta)$ be the size of this optimal partitioning.*

The dual version of this problem (DVPP) can be defined similarly by minimizing the total weight with a bound on the size of the partitioning. We focus on the former due to lack of space.

2.2 Algorithms with guarantees

While no previous work has been done for problem VPP, some variants of it are well studied. In particular, if we define the weight of a rectangle R as the sum of all $R[x][y]$'s, and the weight of a partitioning Π as the maximum weight of rectangles in Π, we have a variant of problem VPP, sometimes called the *RTILE* problem [8]. The RTILE problem is NP-complete [8], and the proof of its NP-completeness can be extended to work for problem VPP (we omit the proof here due to lack of space).

Given the hardness of this problem, we thus shift our focus to approximation algorithms which provide a guarantee on the output size. There is a 2-approximation algorithm for problem VPP, which finds a partitioning Π of a given array A so that $\omega(\Pi) \le \delta$ and $|\Pi| \le 2\kappa(\mathrm{A}, \delta)$. We first introduce a nicely structured class of partitionings that underlies this 2-approximation algorithm.

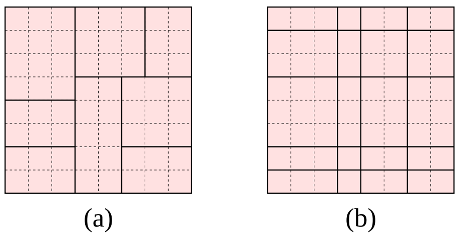


Figure 3: (a) A slicing partitioning of size 8. (b) A grid partitioning of size 4×5. (This picture may not display correctly in soft version. Printed version is fine.)

Slicing model. Given an input rectangle A, we can slice it either with a vertical or a horizontal cut. Each cut will divide its parent rectangle into two, and we then slice the resulting two children rectangles recursively. An example is shown in Figure 3 (a). A partitioning obtained this way is called a *slicing partitioning* [12, 14], or a *binary space partitioning* (BSP) [3]. Let $\rho(\mathrm{A}, \delta)$ denote the size of the optimal slicing partitioning of A with weight smaller than δ, and $\kappa(\mathrm{A}, \delta)$ denote that of the optimal arbitrary partitioning. The following lemma follows from a classic result in computational geometry about BSP [3].

Lemma 1 $\kappa(\mathrm{A}, \delta) \le \rho(\mathrm{A}, \delta) \le 2\kappa(\mathrm{A}, \delta)$.

The above lemma implies that an optimal slicing partitioning for A is a 2-approximation for problem VPP, and it can therefore become our focus now. Dynamic programming is a classic method to handle slicing structure [13, 3]. Below, we will give DP-Alg, a dynamic programming based 2-approximation algorithm for VPP. It follows the same framework as the one in [10], which was developed for the generalization of the RTILE problem.

Dynamic programming approach. First, we show that the dual problem DVPP can be solved by dynamic programming.

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

PROOF. Let $\omega_s^*(l \cdots r, b \cdots u)$ be the minimum weight of any rectangle (subarray) R in A with at most s subrectangles, where $R = \mathrm{A}[l \cdots r, b \cdots u]$. The ultimate goal is to compute $\omega_k^*(1 \cdots n, 1 \cdots m)$. If $s = 1$, or if $\omega(R) = 0$, we simply set $\omega^*_s(l\cdots r, b\cdots u) = \omega(R)$. Otherwise, we have:

$$\omega^*_s(l\cdots r, b\cdots u) = \min_{1\le t<s}\{$$

$$\min_{\substack{l\le i<r\\ b\le j<u}} \left\{ \begin{array}{l} \omega^*_t(l\cdots i, b\cdots u) + \omega^*_{s-t}(i+1\cdots r, b\cdots u), \\ \omega^*_t(l\cdots r, j\cdots u) + \omega^*_{s-t}(l\cdots r, j+1\cdots u) \end{array} \right\}\}.$$

Roughly, the second minimization term enumerates all vertical and horizontal cuts, while the first term ($1 \le t < s$) enumerates all possible division of rectangle numbers between the two separated parts. The weight of any subarray, $\omega(\mathrm{A}[l\cdots r, b\cdots u]$, can be computed in $O(1)$ time after $O(nm)$ time/space preprocessing by extending prefix sum algorithm [8] to our weight function. It then follows that we can build a table of size $O(kn^2m^2)$ to store $\omega^*_s(l\cdots r, b\cdots u)$, for all $1 \le s \le k$, $1 \le l < r \le n$ and $1 \le b < u \le m$, in time $O((n+m)kn^2m^2)$. Thus proves the lemma. ◻

We can now guess the optimal number of rectangles $\kappa^* = \kappa(\mathrm{A}, \delta)$ in a binary manner, starting with $\kappa^* = 1$. By Lemma 1 and 2, we conclude with the following result.

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

3 Fast Heuristic Algorithm

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

3.1 Algorithm overview

Ideally, we would like to have some guarantee on the size of the rectangles output by TS-Alg, while keeping the complexity of the algorithm low. DP-Alg as introduced in last section produces near-optimal solutions. Unfortunately, it is too slow to be practical. In fact, the large space requirement limits the size of the input rectangle to merely around 100×100, 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, before we feed it to DP-Alg. This motivates us to design the following two step approach for problem VPP, referred to as TS-Alg.

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

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

We also refer to a $p \times q$ array G a *grid partitioning* of size $p \times q$ for A (see Figure 3 (b)). Note that both the quality and quantity of the first step directly affect the performance of the second step: On 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 cut any cell in G (i.e., a rectangle in the final output will be a combination of some cells from G), cells from G should be 'good'. We will see in what follows that although TS-Alg does not guarantee that the output size will approximate $\kappa(\mathrm{A}, \delta)$ within some constant factor, we have a control in the quality in each of the two steps. The experimental results from next section further demonstrate its performance both in efficiency and in output quality.

3.2 Size reduction

One straightforward approach for `Step 1` of TS-Alg is to simply subdivide A evenly into $p \times q$ rectangles (so all pq number of cells in G are congruent). This method is completely oblivious to the cell value distribution in A. Therefore, it may produce large p and q in order to satisfy that $\omega(G) \le \epsilon$, and the 'wrong' boundaries induced by G limit the possible solutions from DP-Alg. The natural question to ask is then: given an error threshold ϵ, what is the grid partitioning of smallest size (i.e., $p \times q$), and whose weight is at most ϵ. Variants of this problem have already been studied in the theory and algorithm field [11, 4], and we can modify the algorithm from [11] to obtain the following result.

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

On the high level, the grid partitioning problem can be reduced to a Set Cover problem with small VC dimension. It is shown in [4] that such problems can be efficiently approximated using ϵ-nets and an elegant analysis by Clarkson in [6]. The algorithm is easy to implement. However, its description and analysis are somewhat involved. We therefore omit it from current version of the paper. Interested readers can refer to [11] and references within.

3.3 Putting everything together

Given a $p \times q$ grid partitioning G, let A_T denote the subarray from A covered by some rectangle T of G. For the second step, we modify the definition of the weight of T as follows:

$$\omega(T) = \sum_{(x,y)\in \mathrm{A}_T} (\mu(\mathrm{A}_T) - \mathrm{A}[x][y]).$$

In other words, we still use the original underlying array A to compute the weight, although the dynamic programming table is now built upon cells from G. Furthermore, we can preprocess A in roughly $O(nm)$ time into a structure of size $O(nm)$, so that $\mu(R)$ for any subarray $R \subset$ A can be computed in $O(1)$ time. Therefore, the second step can be implemented in $O(nm + k^2(p+q)p^2q^2 \log k)$ time and $O(nm + p^2q^2k)$ space. Putting everything together, we have:

Theorem 2 *The TS-Alg runs in $\tilde{O}(nm + k^2(p+q)p^2q^2)$ time and $O(nm+p^2q^2k)$ space, where $\tilde{O}$ hides some logarithmic terms.*

The size of p and q depends on the parameter $\epsilon \le \delta$ in the first step. In practice, if $p \ll n$, $q \ll m$, and k is small. Then our TS-Alg can run in roughly $O(nm)$ time.

4 Experimental Results

4.1 Experiment setup and snapshots of our 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 as follows:

We use the Cadence's commercial tool SoC Encounter [1] to do timing-driven placement, timing optimization and timing analysis. Then we assign voltage to each cell according to its worst slack.

We transform the standard cell placement and associated voltage requirement into the input array as stated earlier.

We calculate the maximum power increase, which is the total power increase when all cells are raised to the highest required voltage on the entire chip.

We give some reasonable bounds on the total amount of power increase, each corresponding to a certain percentage of the maximum power increase, and apply our TS-Alg to generate minimum number of voltage islands within each power increase 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. Figure 7 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 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.

4.2 Comparison with other approaches

To demonstrate the efficiency of our TS-Alg, we compared it with two alternative approaches.

Outline of alternative approaches. The first one is rather straightforward: It is the logical boundary based approach, where each grouped module or cell in the logical hierarchical tree forms an individual voltage island (Figure 4(a)). Currently this is the approach commonly used in practice (as mentioned in the example in section 1).

This approach is often very inefficient due to the high fanout of modules in the logical hierarchy tree. For the example in Figure 4(b), module A_1 has been marked high voltage due to its internal timing critical cells. The rest of cells in the sub-tree rooted at A are non-critical. If we don't group module A, each child of A will become an individual voltage island, causing too many fragments. If we group module A, the entire module will be raised to the high voltage that is required on A_1, causing too much power consumption. Such case is very common in real designs.

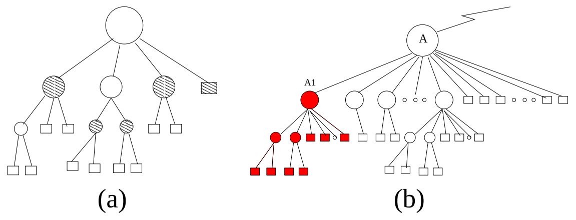


Figure 4: Logical hierarchical tree. (a) shaded modules correspond to one possible partitioning. (b) A node A with high fanout.

A natural way to improve the logical-boundary based approach is to substitute the high-fanout hierarchical tree with non-logical boundary based, standard quad-tree. The leaves of the quad-tree are the 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 increase, 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 increase resulted from combining the four subregions into R. Now in order to compute a partitioning, we start with a tree where all non-leaf nodes are white. At any time, we choose the candidate with smallest cost (by using a priority tree). The process will terminate when the overall weight of the resulting partitioning exceeds the given upper bound δ. We refer to this algorithm QT-Alg. The overall time complexity is $O(nm \log(nm))$ and space complexity is $O(nm)$. Figure 5 illustrates the quad-tree algorithm.

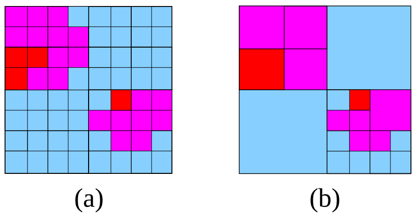


Figure 5: Quadratic tree. (a) The input array. (b) After some combinations

Comparison on the same design with different power increase bound. Figure 6 compares the output size of our TS-Alg with the logical tree and quad-tree algorithms on one industry design with different power increase bounds. Clearly our TS-Alg outperforms the other two significantly and constantly (by an order of 10 and 2, respectively). Between the two alternatives, quad-tree algorithm is significantly better. Therefore we will only compare with quad-tree algorithm in the rest of our experiment.

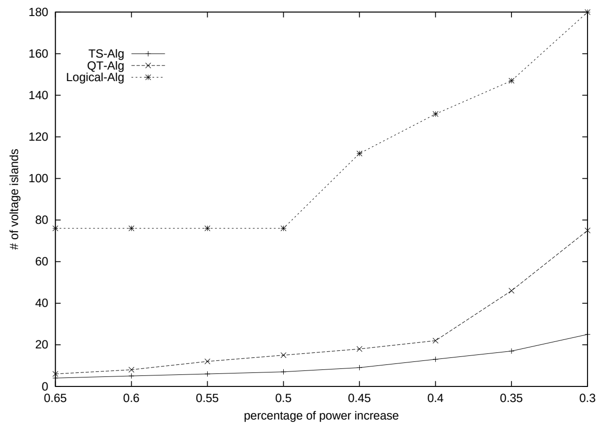


Figure 6: comparing different algorithms on the same design

Comparison on different designs. We extend the experiment to a wide selection of industry designs with different sizes. For each design, we compare the two algorithms against different power increase bounds, and obtain similar results in output size as in Figure 6. Due to the limited space, we omit them here. Instead, for each design, we pick a particular power increase bound such that the number of voltage islands being generated is within a desired range [1]. We selected the range to be around 20, which is roughly an upper bound in current practical designs. Table 1 shows the comparison between our TS-Alg and the quad-tree based QT-Alg. The designs are listed with increasing array size (the input to both algorithms), which is roughly proportional to the number of leaf cells. It can be clearly seen that our TS-Alg outperforms QT-Alg significantly on all designs in terms of the number of voltage islands obtained.

The time complexity of our algorithm has two terms: $O(nm)$ and $\tilde{O}(n + m + k^2(p + q)p^2q^2)$. The first term comes from the preprocessing of the input array into a table for later looking up the weight for any subarray of A. Since this step is needed by both QT-Alg and TS-Alg, we omit it from the running time presented [2]. Other than this preprocessing time, the running time of TS-Alg is output-sensitive, mainly depending on p, q, and k, while QT-Alg still has a $O(nm \log(nm))$ running time. This explains why the first three small designs in Table 1 has larger running time: as $p \times q$ after `Step 1` in their cases is larger than that of later cases. On the other hand, for all practical data we test and for all practical k, p and q are small regardless of the size of the input design (see, for example, $p \times q$ does not increase in Table 1 with the input size). This means that our algorithm scales well with increasing size of the input design! Furthermore, as k decreases, the running time of our algorithm decreases as well, while that of QT-Alg remains roughly the same (as it is not output-sensitive). This is demonstrated in Table 2, where we choose k around 10, which is probably a more practical number in current designs). Note that TS-Alg beats QT-Alg significantly and consistently in terms of both runtime and quality.

References

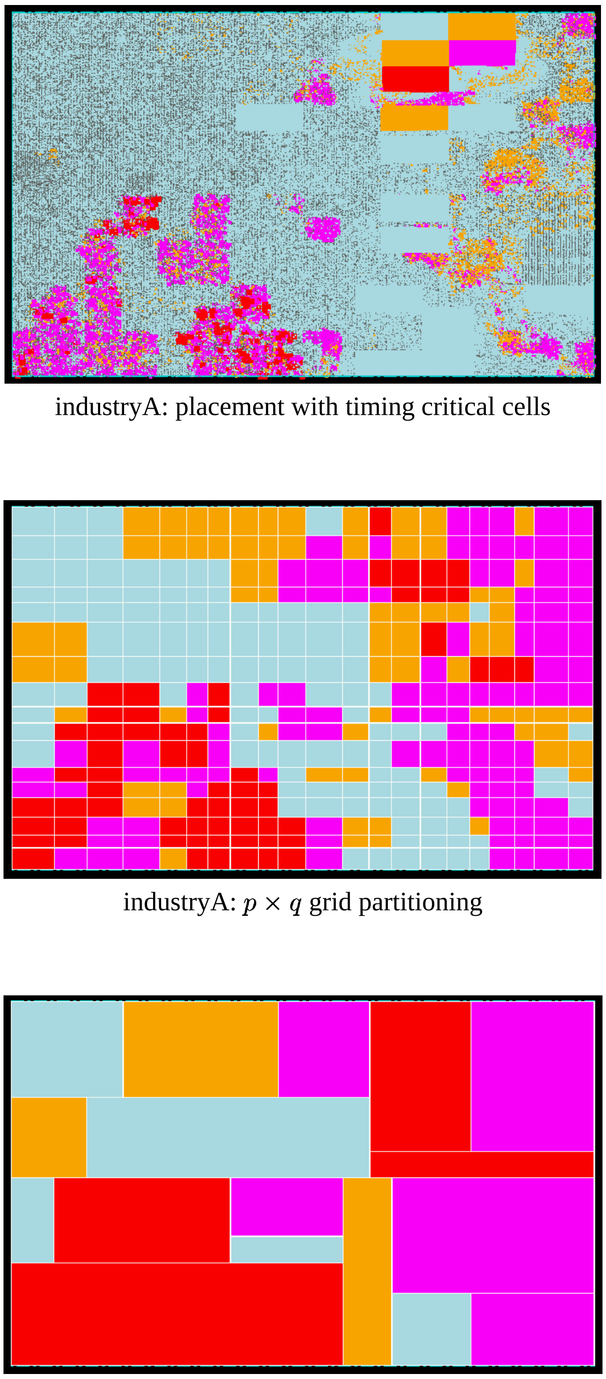
[1] Cadence software manual: Soc encounter gps. http://www.cadence.com/products/digital_ic/soc_encounter/index.aspx.

[2] Power islands: The evolving topology of soc power management. http://www.us.design-reuse.com/articles/article9150.html.

[3] P. Berman, B. Dasgupta, and S. Muthukrishnan. Exact size of binary space partitionings and improved

[1]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 increase, however it can be adapted to directly solve the other problem with faster runtime.

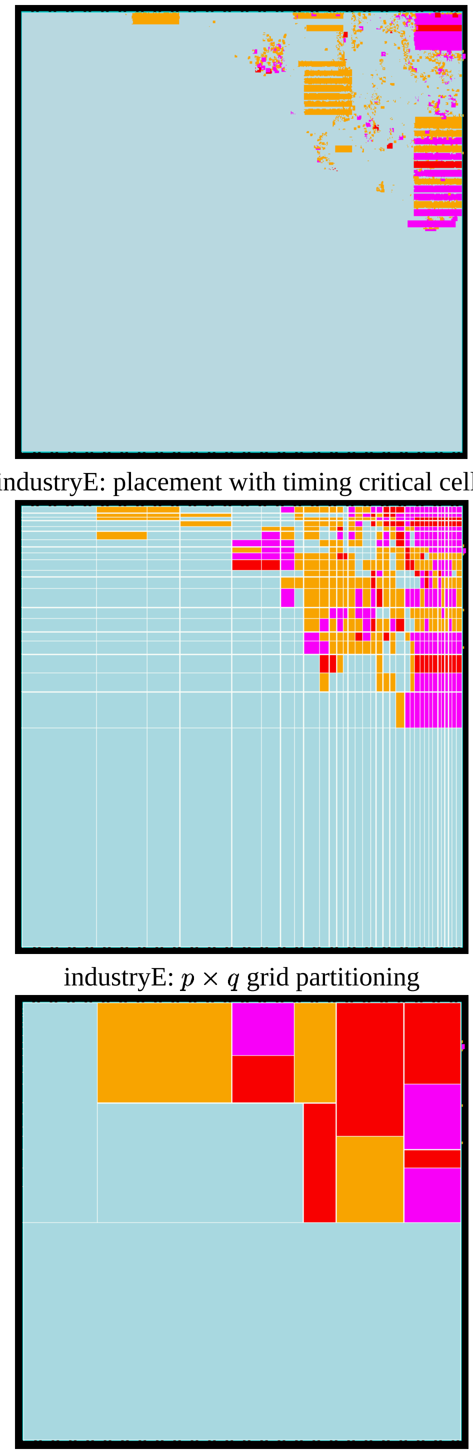
[2]Also since the preprocessing time is $O(nm)$, it will be much smaller than the $O(nm \log(nm))$ runtime for QT-Alg, especially for large design, this log-factor can be quite large! So omitting it will not change our comparison, while helping to clarify things.


industryA: placement with timing critical cells

industryA: $p \times q$ grid partitioning

industryA: voltage islands

(a)


industryE: placement with timing critical cells

industryE: $p \times q$ grid partitioning

industryE: voltage islands

(b)

Figure 7: (a) design industryA (b) design industryE

Design			Power	Grid Size	# of VI$\{k\}$		Runtime(s)	
Name	# of Cells	Array Size$\{n \times m\}$	Bound	$\{p \times q\}$	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 1: Comparison of TS-Alg and QT-Alg with output # of voltage islands around 20

Design	Power	Grid	# of VI		Runtime(s)	
Name	Bound	Size	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

Table 2: Comparison of TS-Alg and QT-Alg with output # of voltage islands around 10

rectangle tiling algorithms. *SIAM J. Discrete Math*, 15(2):252–267, 2002.

[4] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete Comput. Geom.*, 14:463–479, 1995.

[5] J. Buurma and L. Cooke. Low-power design using multiple V_{TH} ASIC libraries. http://www.sinavigator.com/Low_Power_Design.pdf.

[6] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.

[7] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu. Architecting voltage islands in core-based system-on-a-chip designs. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 180–185, 2004.

[8] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, 1998.

[9] 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 *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design table of contents*, pages 195–202, 2002.

[10] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *ICDT '99: Proceeding of the 7th International Conference on Database Theory*, pages 236–256, 1999.

[11] S. Muthukrishnan and T. Suel. Approximation algorithms for array partitioning problems. *Journal of Algorithms*, 54:85–104, 2005.

[12] R. H. Otten. Automatic floorplan design. In *Proceedings of the 19th ACM/IEEE conference on Design automation*, pages 261–267, 1982.

[13] L. P. P. P. van Ginneken and R. H. J. M. Otten. Optimal slicing of plane point placements. In *Proceedings of the conference on European design automation*, pages 322–326, 1990.

[14] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proceedings of the 23rd ACM/IEEE conference on Design automation*, pages 101–107, 1986.