

An Efficient Algorithm for Library-Based Cell-Type Selection in High-Performance Low-Power Designs

Li Li*, Peng Kang*, Yinghai Lu[†] and Hai Zhou*

*Electrical Engineering and Computer Science, Northwestern University, USA

[†]Analog Mixed Signal Group, Synopsys Inc, USA

Abstract—In this paper, we present a complete framework for cell-type selection in modern high-performance low-power designs with library-based timing model. Our framework can be divided into three stages. First, the best design performance with all possible cell-types is achieved by a Minimum Clock Period Lagrangian Relaxation (Min-Clock LR) method, which extends the traditional LR approach to conquer the difficulties in discrete scenario. Min-Clock LR fully leverages the prevalent many-core systems as the main body of its workload is composed of independent tasks. Upon a timing-valid design, we solve the timing-constrained power optimization problem by min-cost network flow. Especially, we identify and address the core issues in applying network flow technique to library-based timing model. Finally, a power prune technique is developed to take advantage of the residual slacks due to the conservative network flow construction. Experiments on ISPD 2012 benchmarks show that on average we can save 13% more leakage power on designs with fast timing constraints compared to start-of-the-art techniques. Moreover, our algorithm shows a linear empirical runtime, finishing the largest benchmark with one million cells in 1.5 hours.

I. INTRODUCTION

Power consumption is one of the most important issues in modern IC design. There are two main incentives behind the imperative demand on reducing the power budget: one is the widespread usage of portable electronic devices, which makes battery lifetime a critical metric of product quality; on the other hand, power dissipation will continue be a bottle-neck for next-generation workstations and supercomputers. For emerging technologies with extremely high transistor density, such as 3D-IC, effective power saving techniques have become indispensable in design process, as the heating problem can be devastating to chip performances.

Cell-type selection is well-known as one of the most effective ways to save power in circuit/gate level of CAD design. Different cell types originate from the combination of two cell parameters, gate size and threshold voltage (vt). The cell-type selection problem is to find the optimal mapping of standard library cells to a design while keeping the original logic composition of the netlist and meeting all timing constraints. Researchers have been attacking the gate sizing and vt assignment problem with various techniques, such as linear programming [1], [2], [3], [4], convex programming [5], [6], [7], [8], network flow [9], [10], [11], Lagrangian relaxation [12], [9],

[13], [14], [15], [5], dynamic programming (DP) [16], [17], [15], multi-direction search [18], branch-and-bound [19], sensitivity-based heuristics [20], [21], [22], [23] and hybrid approaches [19], [5], [17], [9]. However, many of these techniques operate in continuous domain and therefore cannot be directly applied to the CAD problem in real life which is discrete by nature. Their formulations are often derived from simple delay model such as piecewise-linear model [14], [2] and Elmore delay model [11], [5], [13], [9], [10], while lookup table based timing model has become the industrial standard as it is capable of handling more complex timing factors that arise in practice. It has been pointed out in [15] that there exists a substantial gap between the simplified timing models usually used in academia and the realistic ones actually used in industry.

Discrete cell-type selection has been proved to be NP-hard [24]. Thus the attempt of achieving optimality usually incurs intensive computations [25]. On the other hand, relatively fast greedy heuristics [22], [23] is prone to be trapped in local optimal solutions. Another popular approach to discrete cell-type selection is DP, which may produce optimal solution in a reasonable amount of time. However, DP is known to suffer from fanout reconvergence problem [16] if the circuit is not of tree-structure. A recent break-through in discrete cell-type selection is [15]. It considers comprehensive realistic timing models and decouples timing calculation from power optimization by a new LR formulation. The LR subproblem is solved by a modified DP with critical tree extraction. However, [15] does not discuss how to separately handle rise and fall delays in the realistic timing model, which is not trivial.

In this paper, we present a fast cell-type selection technique which adopts industrial library-based timing models and facilitates the design flow. Our work is primarily motivated by the following issues which are not well discussed in previous works: the optimization approach should be able to take advantage of the emerging many-core systems to effectively reduce the design cycle time for ultra-scale modern designs; the designer should have a clear picture of the performance limit of design, as well as the possible power-performance trade-off. In the rest of paper, we will develop an optimization flow to properly address the above issues with delicate algorithms. Our major contribution in this paper includes:

- We propose a Minimum Clock Period Lagrangian Relaxation (Min-Clock LR) method to generate the highest-performance design with all possible cell-types. The rise and fall delays are explicitly formulated in LR for the first time. We boost the convergence speed of Min-Clock LR by a modified subgradient method. Moreover, a novel parallel-friendly approach to LR subproblem (LRS) is presented under library-based timing model.
- We propose min-cost network flow based method to optimize the power consumption while maintaining the feasibility of the design. In particular, we enhance the min-cost flow algorithm to handle discrete non-convex timing models.
- We propose a greedy heuristic to further prune power by

*This work is supported by the NSF under CCF-0811270 and CCF-1115550.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IEEE/ACM International Conference on Computer-Aided Design (IC-CAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright © 2012 ACM 978-1-4503-1573-9/12/11... \$15.00

maximally utilizing the slacks which is falsely underestimated in network flow construction. In order to speed up the heuristic, a fast and coarse-grained timing update technique is developed to replace the accurate but expensive timing update.

- Experimental results show that our algorithm can achieve 13% more power savings in designs with fast timing constraints compared to state-of-the-art techniques [26]. In addition, our algorithm exhibits a near-linear empirical runtime, with processing rate from 1 to 1.5 hours per million cells.

The rest of the paper is organized as follows. In the next section, we briefly introduce the modern lookup table based timing model. Section III describes the algorithm flow. Section IV present the Min-Clock LR and the speed-up technique. Section V presents the min-cost network flow based method to optimize the power under timing constraints. Section VI presents a power prune technique that further saves power. Section VII presents the results of the experiments, comparing the performance and solution quality of our algorithm with state-of-the-art algorithms. Section VIII concludes the paper.

II. LIBRARY-BASED TIMING MODEL

In a typical modern cell library, cell delays and slews (transitions) are defined by lookup tables (LUT). Each timing arc represents an “internal” timing path from an input pin to an output pin of a particular cell. For each timing arc of a cell, there are two delay tables (rise and fall) in the form of $delay[output_load, input_slew]$, two slew tables in the form of $slew[output_load, input_slew]$ and a timing sensitization, which indicates how the output toggles with respect to different input transitions. The timing arc is positive/negative/non-unate if a rising transition on the input causes the output to be rise/fall/either rise or fall. For example, if a timing arc is negative unate, then we should use the fall slew of the input when looking up the rise delay and slew of an output. Since each delay/slew table only contains a few sample points of delay and slew, linear interpolation is used to estimate between the nearest table entries. Note that although in practice delay and slew can have linear or near-linear dependence near a data point in the table, in this paper we make no assumption about the content of delay/slew tables. There are three types of possible timing violations in industrial standard, namely slack violation, transition violation and capacitance violation. Slack violation indicates that there is a negative slack on a primary output. For practical reasons, there is usually an upper bound of slew allowed in the design, any slew exceeds the bound will be regarded as a transition violation. A capacitance constraints defines the maximum capacitance load allowed on an output pin of cell, which is determined by the driving strength of the cell.

III. ALGORITHM FLOW

Our algorithm flow comprises three basic stages: 1) Achieving best timing of the design. 2) Optimizing power of the design under given timing constraints. 3) Power pruning. Satisfying timing constraints in complex timing model is very hard since no explicit formula can be used as in simple timing model. For example, in the popular power-aware LR approaches [15], [13], large total negative slacks (TNS) may exist in most iterations even with rigid timing constraints. The problem can be even worse if appropriate initial Lagrangian multipliers and footsteps can not be found. Thus these approaches may impose extra burden on designers as fixing large TNS itself can be a full-fledged optimization problem in industrial CAD flow. In our problem, no timing violation in any form will be tolerated as long as we start power optimization. To achieve this goal, we first use a Min-Clock LR method to push timing to the limit with all

possible cell-types. As we will show in Section VII, the performance limit is a very important design parameter that provides guidance for deciding power-performance trade-off. In the second stage, we model the timing-constrained power optimization problem into a network flow problem. An accurate timing change estimator is developed to consider the complex effect of swapping a cell in library-based timing model. The network flow construction and timing change estimation is in a conservative way that guarantees valid timing, so in the third stage we further design a fast heuristic to utilize the residual slacks. Note that the three different techniques proposed can be used independently under different conditions. The Min-Clock LR method can be used to fix setup timing violations. The network flow based method can be used for fine-grained power optimization when low-level timing information is available such as rise and fall delay of each timing arc. The power prune method will be useful for coarse-grain power optimization when only high-level timing information such as cell slacks are available through a commercial timer.

IV. MINIMUM CLOCK PERIOD LAGRANGIAN RELAXATION

In this section, we show how to find the highest-performance design with all possible cell-types. It is obvious that all cells in the highest-performance design will use lowest vt (fast speed), thus in the following part of this section we will only consider gate sizing. There are two main reasons that we do not take power into account in this stage. First, we believe that designers should have clear understanding of the performance limit. Without this knowledge, power optimization with tentative clock period will often be inefficient. Secondly, the effectiveness of power-aware LR is strongly dependent on the selection of initial Lagrangian Multipliers (LMs) and the subgradient footsteps. Although some efforts have been dedicated to alleviate this problem [12], there is still lack of robust method that guarantees fast convergence, especially with library-based timing models. The backbone of our Min-Clock LR approach can be easily implemented in parallel, which makes it amenable to CAD flow on the prevalent many-core systems.

The Min-clock LR problem considering both rise and fall delays can be formulated as:

$$\begin{aligned} \text{Min} \quad & T \\ \text{s.t.} \quad & a_{po}^r + d_{po} \leq T \\ & a_{po}^f + d_{po} \leq T \\ & a_u^{s(uv)} + d_{uv}^r \leq a_v^r \\ & a_u^{\bar{s}(uv)} + d_{uv}^f \leq a_v^f \end{aligned} \quad (1)$$

where T is the clock period, a_{po}^r and a_{po}^f are the rise and fall arrival times of the primary outputs po . d_{po} is the delay of po , u and v are input pin and output pin of a cell, uv represents an timing arc from u to v and d_{uv} is the delay on timing arc uv . a_u^r and a_u^f denote the rise and fall arrival times on pin u . $s(uv) = r$, $\bar{s}(uv) = f$ when timing arc uv is positive unate and vice versus. For simplicity, in the rest of the paper we assume that each cell has only one output pin. Our model can be extended to multi-output situation in a straightforward way. Introducing two Lagrangian multipliers μ_{uv}^r and μ_{uv}^f for the rise and fall delay of each timing arc uv , we obtain the Lagrangian-Relaxed Subproblem (LRS) $\min L_\mu(\mathbf{a})$, where

$$\begin{aligned} L_\mu(\mathbf{a}) = & T + \sum_{po} \mu_{po}^r (a_{po}^r + d_{po} - T) + \sum_{po} \mu_{po}^f (a_{po}^f + d_{po} - T) \\ & + \sum_{uv} \mu_{uv}^r (a_u^{s(uv)} + d_{uv}^r - a_v^r) + \sum_{uv} \mu_{uv}^f (a_u^{\bar{s}(uv)} + d_{uv}^f - a_v^f) \end{aligned} \quad (2)$$

According to Karush-Kuhn-Tucker condition, when the solution is

optimal LMs should satisfy:

$$\sum_{po} \mu_{po}^r + \sum_{po} \mu_{po}^f = 1 \quad (3)$$

$$\sum_{b \rightarrow n} \mu_{ab}^r = \sum_{n \rightarrow c, s(cd)=r} \mu_{cd}^r + \sum_{n \rightarrow c, s(cd)=f} \mu_{cd}^f + \sum_{po \in n} \mu_{po}^r \quad (4)$$

$$\sum_{b \rightarrow n} \mu_{ab}^f = \sum_{n \rightarrow c, s(cd)=f} \mu_{cd}^r + \sum_{n \rightarrow c, s(cd)=r} \mu_{cd}^f + \sum_{po \in n} \mu_{po}^f \quad (5)$$

where n is any net in the design, $b \rightarrow n$ indicates that b is a driver pin of net n while $n \rightarrow c$ means that c is a sink pin of net n . With Equation (3)-(5), the LRS can be further reduced to

$$\min L_\mu(\mathbf{a}) = \sum_{uv} \mu_{uv}^r d_{uv}^r + \sum_{uv} \mu_{uv}^f d_{uv}^f \quad (6)$$

Note that all LMs μ are fixed in LRS, which can be regarded as some sort of “weight” for each timing arc.

The optimal delay and arrival time of the original LR problem (1) will also be the optimal solution to the LRS (6) when the LMs μ is the optimal solution to the Lagrangian Dual Problem (LDP) that

$$\begin{aligned} \text{Maximize} \quad & \min L_\mu(\mathbf{a}) \\ \text{s.t.} \quad & (3), (4), (5) \end{aligned} \quad (7)$$

Subgradient method [27] is used to solve the LDP (7). A successful subgradient method strongly relies on a good starting point of μ and a good step size for updating μ . One advantage of Min-clock LR is that finding a reasonable initial vector of μ becomes much easier compared to power-aware LR due to Constraint (3). However, finding a good step size still remains a hard problem as the traditional approach using linear step size [13] is very sensitive and may need lots of artificial tuning. In section IV-C, we propose a novel approach to updating the LMs μ which uses adaptive non-linear step size that is merely dependent on the computed delay and arrival time from the LRS.

The overall flow of Min-Clock LR is shown in Algorithm 1. The LRS (6) is approximately solved in the inner for loop. Then in the outer while loop, we try to solve the LDP (7). Since the cost of incremental timing update with realistic timing model is so huge that it is infeasible to consecutively resizing all cells as done in the continuous approach. In section IV-B, we will present a heuristic for the LRS which thoroughly circumvent incremental timing update. A highlight of our heuristic is that all resizing tasks are completely independent and thus multi-threading techniques can be easily applied to speed up solving the LRS, which accounts for majority of the workloads in our problem. Global timing update is a linear-time operation that basically includes propagating arrival time in topological order and propagating require time in reverse topological order. As there has been many sophisticated commercial timers, in this paper we will not further explore the possible optimization on global timing update. The algorithm terminates when the relative improvement of clock period is less than a given threshold in the last iteration. The threshold is set to 0.2% in our implementation.

A. Initial Sizing

Our Min-clock LR solver starts from a design without capacitance or slew violation (capacitance or slew violation is not related to clock period). If a proper initial design is not given, we can generate an initial design in the following way: $\alpha \leq 1$ is an empirical parameter to control the cap-ratio between driver cell and driven cells. If α is too small, it may eventually force the driver cells to be too large. When $\alpha = 1$, the produced circuit will be the slowest design without any capacitance or slew violation. In other words, if $\alpha = 1$ leads to an initial design with capacitance or slew violations, then

Algorithm 1 Min-Clock LR

```

1: Find an initial sizing without capacitance and slew violations.
2: while clock period improved in last iteration is larger than a
   threshold do
3:   for each gate  $g$  in design do
4:     if topological_level( $g$ ) % 3 == iteration_count % 3 then
5:       Resize  $g$  to minimize  $L_\mu(\mathbf{a})$ .
6:     end if
7:   end for
8:   Update global timing.
9:   Update  $\mu$ .
10: end while

```

Algorithm 2 Initial Sizing

```

1: for each gate  $g$  in reverse topological order of the circuit do
2:   Find the smallest-size alternative cell-type  $c_1$  for  $g$  that satisfies
    $\alpha * \max\_capacitance(c_1) > output\_load(g)$ .
3:   Find the smallest-size alternative cell-type  $c_2$  for  $g$  that satisfies
    $output\_slew(g) < \max\_transition$  assuming that the input
   slews of  $g$  are  $\max\_transition$ .
4:   Set  $g$  to cell-type  $\max(c_1, c_2)$ .
5: end for

```

it will be impossible to produce any violation-free design. With the performance requirement in practice, we incline to set α as low as it will not result in violations because there will be fewer number of iterations in LR with faster initial design. In our experiments, we set $0.6 \leq \alpha \leq 0.8$. Upon an initial design without capacitance or slew violation, we do not have to especially fix capacitance or slew violation in the optimization process as long as we pre-validate the cell-type candidate before swapping a cell. This further avoids the redundant design cycles to eliminate new slack violations caused by fixing capacitance or slew violations.

B. Solving Lagrangian-Relaxed Subproblem

In continuous LR method [13], LRS (6) is solved by iteratively resizing all gates to achieve local optimal. We inherit this basic idea from continuous LR and extend it to a discrete scenario. A simple example in Figure 1 is used to show the comprehensive effect of resizing a cell. Cell A and F are connected to cell B through net

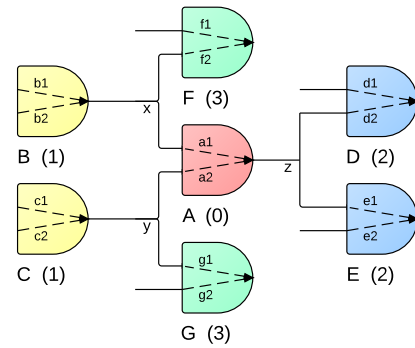


Fig. 1. An Example of Resizing Effect

x . Cell A and G are connected to cell C through net y . Cell D and E are connected to cell A through net z . Suppose we want to resize A to minimize $L_\mu(\mathbf{a})$ while all other cells remain unchanged. We first filter out sizes that may lead to any capacitance or slew violation, then the effect of resizing A to all remaining candidate sizes is estimated. Since the load on nets x, y are changed, the delays on timing arcs $b1, b2, c1, c2$ are re-estimated. The new slews on nets

x and y can be calculated as well. The delays on timing arcs $f2$ and $g1$ are updated according to the new input slews. The delays on timing arcs $a1$ and $a2$ are updated with the new input slews and new delay tables. After the new slew on net z is calculated, we obtain the new delays on timing arc $d2$ and $e1$. Now that we know the precise effect of resizing cell A on $L_\mu(\mathbf{a})$. The size among all candidates that minimizes $\Delta L_\mu(\mathbf{a}) = \mu_{b1}^r \Delta d_{b1}^r + \mu_{b1}^f \Delta d_{b1}^f + \mu_{b2}^r \Delta d_{b2}^r + \mu_{b2}^f \Delta d_{b2}^f + \mu_{c1}^r \Delta d_{c1}^r + \mu_{c1}^f \Delta d_{c1}^f + \mu_{c2}^r \Delta d_{c2}^r + \mu_{c2}^f \Delta d_{c2}^f + \mu_{a1}^r \Delta d_{a1}^r + \mu_{a1}^f \Delta d_{a1}^f + \mu_{a2}^r \Delta d_{a2}^r + \mu_{a2}^f \Delta d_{a2}^f + \mu_{g1}^r \Delta d_{g1}^r + \mu_{g1}^f \Delta d_{g1}^f + \mu_{f2}^r \Delta d_{f2}^r + \mu_{f2}^f \Delta d_{f2}^f + \mu_{e1}^r \Delta d_{e1}^r + \mu_{e1}^f \Delta d_{e1}^f + \mu_{d2}^r \Delta d_{d2}^r + \mu_{d2}^f \Delta d_{d2}^f$ will be chosen as the new size of cell A . In the order of importance to timing change estimation, we classify the cells to four levels. Cell A (resized cell) is level 0. Cell B and C (driver cells) are level 1 as it is directly influenced by the capacitance change. Cell D and E (driven cells) are level 2 as they are directly influenced by the slew change. Cell F and G (peer cells) are level 3 since they are only indirectly influenced by the slew change of cell B and C . Usually considering level 2 cells is necessary for accurate timing change estimation. In more complex situations, considering level 3 may help improve estimation quality, e.g. when cell A has a large number of peer cells like cell F and G .

Note that discrete library-based resizing can very be computationally intensive compared with continuous resizing, as there can be a few size candidates and each candidates may require many table inquiries. Fortunately, one resizing task will not affect others as long as the resized cells are far away enough to avoid significant capacitance or slew effect. However, detecting if another resizing is executed in neighborhood can be difficult and expensive. We solve this problem by an efficient heuristic that insulates simultaneous resizing. In each iteration, we only resize the cells satisfy that their topological level number mode 3 is equal to that the iteration number mode 3. The heuristic can not guarantee all resized cells are at least two levels away (slew effect will completely fade out after two level's propagation) but it significantly reduces the possibility of inaccurate estimation due to adjacent resizing.

Another difference between discrete and continuous resizing is the outdated timing information. Incremental timing update after resizing each cell is not affordable in library-based timing models. As the resizing tasks proceed, the old slews gained from last global timing update will become more and more inaccurate. Thus our discrete resizing process will not continue to convergence, instead, we will only run one iteration and then go ahead to update global timing and LMs. The early termination of solving the LRS will also significantly accelerate the global convergence.

C. Improved Subgradient Search for Lagrangian Multipliers

One major drawback of traditional LR subgradient search is that a global footstep is used to decide next μ . We propose a new procedure that adjusts sub-gradients according to local delay and arrival time, as shown in Algorithm 3.

β is a footstep parameter. When β is infinite large, we are only considering most critical paths. Excessively large β can be harmful in discrete problems because it may lead to a deadlock, resizing among paths with similar criticality. In our experiments, we find that $\beta = 2$ is a proper trade-off between stability and fast convergence.

V. GENERALIZED NETWORK FLOW BASED CELL OPTIMIZATION

In this section, we start from the timing-valid design obtained by Min-Clock LR and strive to achieve the minimum power consumption while meeting all timing constraints. Omitting the transition types,

Algorithm 3 Updating μ

```

1: for each timing endpoints  $po$  in design do
2:    $\mu_{po}^r = \mu_{po}^r (a_{po}^r + d_{po})^\beta$ .
3:    $\mu_{po}^f = \mu_{po}^f (a_{po}^f + d_{po})^\beta$ .
4: end for
5: for each timing arc  $uv$  in design do
6:    $\mu_{uv}^r = \mu_{uv}^r (a_{uv}^{s(uv)} + d_{uv}^r)^\beta$ .
7:    $\mu_{uv}^f = \mu_{uv}^f (a_{uv}^{s(uv)} + d_{uv}^f)^\beta$ .
8: end for
9: for each timing endpoints  $po$  in design do
10:   $cur\_sum = \sum_{po} \mu_{po}^r + \sum_{po} \mu_{po}^f$ .
11:   $\mu_{uv}^r = \mu_{uv}^r / cur\_sum$ .
12:   $\mu_{uv}^f = \mu_{uv}^f / cur\_sum$ .
13: end for
14: for each gate  $g$  in reverse topological order of the circuit do
15:   $cur\_sum^r = \sum_{uv \in g} \mu_{uv}^r$ ,  $cur\_sum^f = \sum_{uv \in g} \mu_{uv}^f$ .
16:   $opt\_sum^r = \text{Equation (4)}$ ,  $opt\_sum^f = \text{Equation (5)}$ , where
     $n$  is output net of  $g$ .
17:  for each timing arc  $uv$  in gate  $g$  do
18:     $\mu_{uv}^r = \mu_{uv}^r * opt\_sum^r / cur\_sum^r$ .
19:     $\mu_{uv}^f = \mu_{uv}^f * opt\_sum^f / cur\_sum^f$ .
20:  end for
21: end for

```

the optimization problem can be generally formulated as follows.

$$\begin{aligned}
& \text{Min} \quad \sum_{uv} P_{uv}(\mathbf{d}) \\
& \text{s.t.} \quad a_{po} - T \leq a_{pi} \\
& \quad \quad a_u + d_{uv} \leq a_v
\end{aligned} \tag{8}$$

Where u and v are nodes in the network flow graph with an directed edge uv from u to v . Each edge uv is associated with a delay-power curve $P_{uv}(\mathbf{d})$. Note that in previous works each node in the network flow graph may not exactly correspond to each pin in the circuit since each cell can not have multiple delay-power curves. When $P_{uv}(\mathbf{d})$ is a convex separable function, i.e. $P_{uv}(\mathbf{d}) = P_{uv}(d_{uv})$, Problem (8) can be formulated as a min-cost network flow dual problem and is well studied in the research community [28], [29], [30]. Network flow based solver is very powerful when handling circuit optimization problem because it gives a natural representation of the topology, sensitivity and criticality of the circuit elements [9]. However, in the real world, the cost function $P_{uv}(\mathbf{d})$ from a library-based cell model is a different animal, which prevents us from directly adopting network flow to solving Problem (8). There are three major differences.

- There are multiple timing arcs in a cell with different delay and slew tables. Moreover, the rise and fall delay of a timing arc can be very different. Therefore each cell can not be simply represented by a single delay-power curve $P_{uv}(\mathbf{d})$ any more, which is the presumption of formulation (8).
- The delay of adjacent cells are strongly correlated due to capacitance and slew effects. $P_{uv}(\mathbf{d})$ can no longer be assumed as a separable function.
- Even with the same input slews and output load, $P_{uv}(\mathbf{d})$ can be non-convex.

In this paper, we reinforce the network flow based solver to handle these difficulties with library-based cell models. Since network flow as a technique for power optimization has been extensively studied [28], [29], [30], [9], we will focus on how to address the above difficulties in applying network flow to library-based timing model. More specifically, we will demonstrate how to construct a timing-improved network flow graph with proper delay-power curves as in

formulation (8).

A. Incorporating Multiple Timing Arcs and Rise and Fall Delays

Since different timing arc within the same cell may have different delay-power curve, there does not exist an uniform delay-power curve for a cell. In order to address this problem, some extra nodes and edges are added in the network flow graph, as illustrated in Figure 2. For each pin in the circuit, we create a node in the network flow

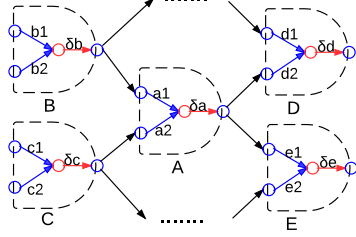


Fig. 2. An Example of Network Flow Construction

graph. All pin nodes are properly connected according to the netlist. In addition, we introduce a dummy node for each cell. All input pins are connected to the dummy node to represent the corresponding timing arcs. Each timing arc has a fixed delay which is the maximum in rise and fall delay in the current design. The dummy node is connected to the output pin to mimic an additional delay arc, e.g. δa in cell A. The dummy delay arc is associated with the unique delay-power curve of the cell. Point (0,0) in the delay-power curve is used to represent the current cell-type. When all the zero points in the delay-power curve are chosen, the network flow corresponds to the current design. If we recompute the timing of the network flow graph, then the arrival time on any output-pin node is strictly greater than the maximum in the rise and fall arrival time of the real pin. In other words, if the network flow graph has no timing violation then the design must be timing valid, but the opposite may not hold. Such construction of the network flow graph is generally pessimistic, but it is the only way that rigidly guarantees timing validity through the optimization procedure. In the next subsection, we will show how to generate the complete delay-power curve that keeps the timing validity of network flow.

B. Decoupling Timing Dependency between Adjacent Cells

For simplicity, we assume that all other cells remain unchanged when the effect of swapping a cell is being evaluated. The insulation technique described in Section IV-B can be used to enforce this assumption. Again, we use Figure 1 as an example, in which all timing arcs are negative unate. When cell A to swapped to its cell-type candidate, the worst rise slack change of its output pin can be estimated as $-\Delta slack_r(A_{out}) = \max(\Delta d_{a1}^f + \max(\Delta d_{b1}^f, \Delta d_{b2}^f), \Delta d_{a2}^r + \max(\Delta d_{c1}^f, \Delta d_{c2}^f)) + \max(\Delta d_{d2}^f, \Delta d_{e1}^f)$, and the worst fall slack change has the similar form. Our estimation considers the worst condition when the timing arc with the worst delta delay is right on the worst path through the output pin of cell A. As before, let (0,0) in the delay-power curve represent the current cell-type. Then we can use $(\Delta d_{\delta a}, \Delta p_{\delta a})$ to represent the cell-type candidate, where $\Delta d_{\delta a} = \max(-\Delta slack_r(A_{out}), -\Delta slack_f(A_{out}))$ and $\Delta p_{\delta a}$ is the power difference between current and candidate cell-type of cell A. In addition, we only consider downsizing in network flow solver to avoid unpredictable timing degradation on level 3 cells (cell F and G). In next subsection, we show how to determine the best cell-type candidate. It is not hard to verify that if choosing delay-power point

Algorithm 4 Power Pruning

```

1: while stop criteria not met do
2:   Find all active gates and calculate their sensitivities.
3:   Sort all active gates in the order of their sensitivity from high
   to low.
4:   for each gate  $g$  in the active pool do
5:     if  $slack(g) + \Delta slack(g) > 0$  then
6:       for each gate  $c$  in the fanin or fanout cone of gate  $g$  do
7:         if  $slack(g) + \Delta slack(g) < slack(c)$  then
8:            $slack(c) = slack(c) + \Delta slack(g)$ 
9:         end if
10:      end for
11:    end if
12:  end for
13: end while

```

of a given cell-type will not lead to timing violation in the network flow graph, then swapping to this cell-type must also not lead to timing violation in the design. Note that we usually have multiple choices of cell-type candidates at any time. If we compute all the delay-power points in the aforementioned way and put them in the same delay-power curve, the curve may not even be convex due to the complex nature of library-based timing model.

C. Dealing with Non-Convex Delay-Power Curves

We propose successive network flow refinement to handle non-convex DP curves. We choose the cell-type with the largest sensitivity as the “best” candidate. In this way, the delay-power curve will only contain two points, thus the convexity of the delay-power curve is trivially guaranteed. The sensitivity of a cell is defined as the slope of its delay-power curve, i.e. $\Delta p_{\delta a} / \Delta d_{\delta a}$. Note that we do not need to estimate all available cell-types because many of them violate capacitance or slew constraints. Since we limit the input delay-power curves of the network flow solver to a local segment, multiple calls to the network flow solver will be invoked to gradually reduce the power. After each call of the network flow solver, the cell-type for each cell is updated by rounding the result of the solver as is done in [29]. Global timing is updated and new delay-power curves are generated according to the latest timing information. The iteration ends when there are not enough number of cells to be updated.

VI. POWER PRUNING

In order to construct a consistent network flow graph, we use the worse delta delay in rise and fall of an timing arc as well as the worst delta delay among all timing arcs. This conservative approach helps to maintain valid timing but may neglect some cells with positive slacks. In this section, we propose a greedy heuristic to maximally utilize the residual slacks missed by the network flow solver.

For each gate g in the design, we find the best cell-type candidate as done in Section V. We define that a gate is active if it has enough slack to be swapped and it is not lowest-power cell-type. We expect that a large portion of gates in the design have become inactive after going through the previous stage. We collect all active gates in the order of sensitivity from high to low and attempt to swap them to the selected candidate one by one. In a straightforward implementation, timing update is necessary after each swap because the later swapped cells have to know if they have enough slacks left. However, accurate updating timing will be too slow even under incremental mode. To solve this problem, we simplify the accurate timing updates to a conservative but fast version. We define the slack of a gate g as the worse in rise and fall slack of its output pin that $slack(g) = \min(slack_r(g_{out}), slack_f(g_{out}))$. And

$\Delta slack(g) = \min(\Delta slack_r(g_{out}), \Delta slack_f(g_{out}))$, where the rise and fall delta slack can be obtained from timing change estimation as in Section V. After swapping a gate g , we update the slack of any gate c in its fanin or fanout cone of gate g in the following way: if $slack(g) + \Delta slack(g) < slack(c)$, then we update $slack(c) = slack(g) + \Delta slack(g)$; otherwise we keep $slack(c)$ unchanged. The core idea here is that if $slack(g) + \Delta slack(g) < slack(c)$, then g may be on the critical path that determines the slack of c . Otherwise g must not be on the c 's critical path. It is also worth mentioning that we use worst slack instead of separate rise and fall slack because the relation of rise and fall slack between two distant cells can be complex due to timing sense (unateness). Due to the conservative nature of our approach, the pruning process need to go through a few iterations to eventually squeeze out all useful slacks. The details of our power prune algorithm are shown in Algorithm 4.

VII. EXPERIMENTAL RESULTS

A. Experimental Setup and Benchmarks

We use the sequential circuits from the ISPD 2012 benchmark suite [26] as our test cases. Generally speaking, the ISPD 2012 benchmark suite is very similar to real designs in industry. It contains a standard cell library with lookup table based timing model considering both rise and fall delays. Each benchmark is composed of three files: 1) *.v*, the verilog netlist file; 2) *.sdc*, the timing constraints file; 3) *.spef*, the interconnection parasitics file. For simplicity, ISPD 2012 benchmarks only contain data path and assume zero clock skews. The interconnection parasitics are modeled as lumped capacitance with zero resistance. The statistics of the benchmark is listed in Table I.

TABLE I
STATISTICS OF BENCHMARKS

Name	#Input	#Output	#Comb cell	#Seq cell	#Total cell
DMA	683	276	23109	2192	25301
pci_bridge32	160	201	29844	3359	33203
des_perf	234	140	102427	8802	111229
vga_lcd	85	99	147812	17079	164891
b19	22	25	212674	6594	219268
leon3mp	254	79	540352	108839	649191
netcard	1836	10	860949	97831	958780

The proposed algorithms are implemented in C++ code, compiled by GCC 4.4.3 and run on a Linux workstation with six 2666 MHz two-socket cores and 72 GB memory. Experiments show the actual memory requirement of our program is about 4 GB per million gates. Intel TBB [31] is used for multi-threading.

B. Comparison of Solution Quality

In Table II, we compare our final leakage power with the best reported in [26]. [26] lists the detailed results of ISPD 2012 discrete gate sizing contest, thus we believe it will be an excellent reference for state-of-the-art cell-type selection techniques. The timing of our results have been verified in the commercial timer PrimeTime and no violation is found, which ensure the comparison is fair. Each benchmark is tested under fast and slow timing constraints (clock periods). The actual clock period of each test case shown in the column "Act-clock". The minimum clock period obtained by our Min-clock LR method is listed in column "Min-clock". The percentage power reduction compared to [26] is listed in column "Sav". On average, our algorithm can reduce 13% more leakage power with fast timing constraints and 0.4% more leakage power with slow timing constraints compared to state-of-the-art techniques. It is worth pointing out that the majority of power reduction comes from network flow solver. Even without power prune heuristic, the

TABLE II
RESULTS COMPARISON BETWEEN [26] AND OUR ALGORITHM

Design	Min-clock (ps)	Act-clock (ps)	Leakage Power (W)			Runtime (h)
			[26]	Ours	Sav (%)	
DMA_fast	637	770	0.312	0.281	9.3	0.01
DMA_slow		900	0.147	0.153	-4.2	0.01
pci_bridge32_fast	640	660	0.168	0.167	0.6	0.02
pci_bridge32_slow		720	0.115	0.111	3.5	0.02
des_perf_fast	596	735	2.32	1.93	16.8	0.11
des_perf_slow		900	0.674	0.671	0.4	0.10
vga_lcd_fast	526	610	0.580	0.460	20.7	0.17
vga_lcd_slow		700	0.378	0.375	0.8	0.13
b19_fast	1960	2100	1.18	0.784	33.6	0.20
b19_slow		2500	0.614	0.604	1.6	0.17
leon3mp_fast	1335	1500	2.02	1.64	18.8	0.91
leon3mp_slow		1800	1.42	1.40	1.4	0.73
netcard_fast	977	1200	2.01	2.18	-8.5	1.48
netcard_slow		1900	1.77	1.78	-0.6	0.80

optimized power consumption will increase by no more than 1% in tested benchmarks. Since power prune heuristic is designed to prevent over pessimism in timing estimation of network flow, it turns out that the gap between the estimated worst timing in network flow and the exact timing is very small in most cases.

C. Runtime and Workload Analysis

The runtime of our algorithm on all benchmarks are listed in Table II. We demonstrate in Figure 3 that the empirical runtime of our algorithm is close to linear, for designs with up to 1 million cells. The

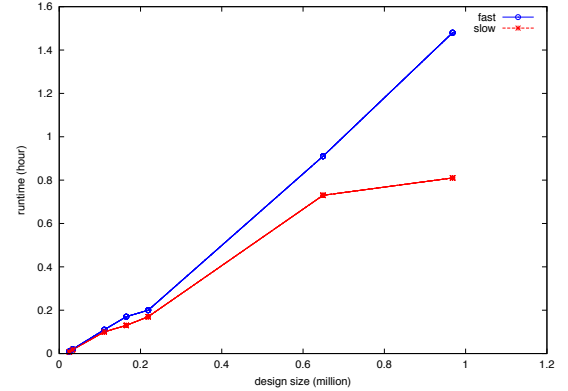


Fig. 3. Runtime Scalability

average processing rate of our algorithm on ISPD 2012 benchmarks is 1 to 1.5 hours per million cells. The exact runtime of techniques in [26] is not reported, and we did not succeed to gain these data from the organizer. However, according to partial data from some statistic graphs in [26], our algorithm is at least 5-10X faster than start-of-the-art techniques with comparable solution quality.

In Figure 4, we demonstrate the composition of different workload types in our algorithm. "Timing Estimation" includes resizing tasks in Min-Clock LR and sensitivity estimation in network flow solver, which is the multi-threaded part in our program. "Network Flow" indicates the time used to solve the transformed min-cost flow problem. "Timing Update" includes all global timing updates. "Power Prune" includes sensitivity sorting and fast incremental timing updates in the power prune stage, but not includes the global timing update in the end of each prune iteration. "Others" includes other non-dominant tasks in our program, such as LMs update. The runtime analysis is done by running the single-threaded program. The multi-threaded version uses 8 threads, which has approximately 7X speed-up on the

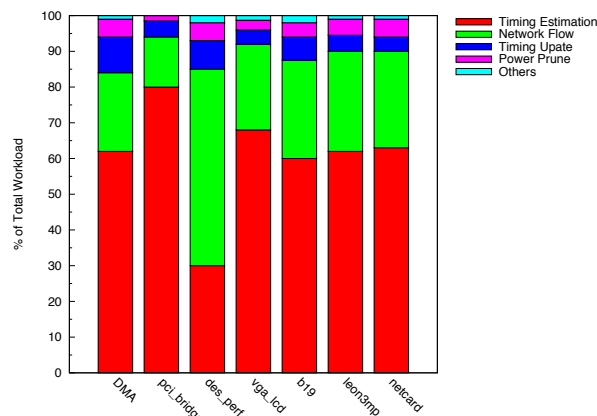


Fig. 4. Distribution of Different Workload Type in all Benchmarks

“Timing Estimation” part. As for the importance of different workloads, the top 3 workload types are “Timing Estimation”, “Network Flow” and “Timing Update”, which account for 60%, 25% and 6% of the whole program, respectively. With 8 independent threads on “Timing Estimation” part, the average speedup of our program is 2.2X. Note that “Timing Update” may take more time in industrial CAD flow due to various timing issues such as coupling effect. In this paper, we only focus on the parallelization on the “Timing Estimation” part. However, we would like to point out that there exists parallel techniques for network flow [32] and global timing update [33]. In summary, our algorithm is parallel-friendly in nature and has the potential to significantly accelerate cell-type selection on the emerging many-core systems.

VIII. CONCLUSION

In this paper, we propose algorithms for the cell-type selection problem in modern high-performance low-power designs. In particular, we provide techniques to address the difficulties that the traditional continuous approaches encountered when applied to real life designs. The experiments show that our parallel-friendly approach achieves better leakage power than state-of-the-art approaches with much less runtime.

REFERENCES

- [1] W. Chuang, S. Sapatnekar, and I. Hajj, “Timing and area optimization for standard-cell VLSI circuit design,” *IEEE Transactions on Computer Aided Design*, vol. 14, no. 3, pp. 308–320, 1995.
- [2] M. Berkelaar and J. Jess, “Gate sizing in MOS digital circuits with linear programming,” in *Proc. DATE: Design Automation and Test in Europe*, 1990, pp. 217–221.
- [3] D. Chinnery and K. Keutzer, “Linear programming for sizing, vth and vdd assignment,” in *Proc. International Symposium on Low Power Electronics and Design*, 2005, pp. 149–154.
- [4] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw, and V. Zolotov, “Discrete vt assignment and gate sizing using a self-snapping continuous formulation,” in *Proc. Intl. Conf. on Computer-Aided Design*, 2005, pp. 705–712.
- [5] H. Chou, Y.-H. Wang, and C. C.-P. Chen, “Fast and effective gate-sizing with multiple-vt assignment using generalized lagrangian relaxation,” in *Proc. Asian and South Pacific Design Automation Conference*, 2005, pp. 381–386.
- [6] S. Roy, W. Chen, C. Chung-Ping Chen, and Y. H. Hu, “Numerically convex forms and their application in gate sizing,” *IEEE Transactions on Computer Aided Design*, vol. 26, no. 9, pp. 1637–1647, 2007.
- [7] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, “A new class of convex functions for delay modeling and its application to the transistor sizing problem,” *IEEE Transactions on Computer Aided Design*, vol. 19, no. 7, pp. 779–788, 2000.
- [8] S. Sapatnekar, V. Rao, P. Vaidya, and S.-M. Kang, “An exact solution to the transistor sizing problem for cmos circuits using convex optimization,” *IEEE Transactions on Computer Aided Design*, vol. 12, no. 11, pp. 1621–1634, 1993.
- [9] J. Wang, D. Das, and H. Zhou, “Gate sizing by lagrangian relaxation revisited,” in *Proc. Intl. Conf. on Computer-Aided Design*, 2007, pp. 111–118.
- [10] H. Ren and S. Dutt, “A network-flow based cell sizing algorithm,” in *The International Workshop on Logic Synthesis*, 2008, pp. 7–14.
- [11] V. Sundararajan, S. Sapatnekar, and K. Parhi, “Fast and exact transistor sizing based on iterative relaxation,” *IEEE Transactions on Computer Aided Design*, vol. 21, no. 5, pp. 568–581, 2002.
- [12] H. Tennakoon and C. Sechen, “Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step,” in *Proc. Intl. Conf. on Computer-Aided Design*, 2002, pp. 395–402.
- [13] C.-P. Chen, C. Chu, and D. Wong, “Fast and exact simultaneous gate and wire sizing by lagrangian relaxation,” *IEEE Transactions on Computer Aided Design*, vol. 18, no. 7, pp. 1014–1025, 1999.
- [14] H. Tennakoon and C. Sechen, “Efficient and accurate gate sizing with piecewise convex delay models,” in *Proc. of the Design Automation Conf.*, 2005, pp. 807–812.
- [15] M. Ozdal, S. Burns, and J. Hu, “Gate sizing and device technology selection algorithms for high-performance industrial designs,” in *Proc. Intl. Conf. on Computer-Aided Design*, 2011, pp. 724–731.
- [16] S. Hu, M. Ketkar, and J. Hu, “Gate sizing for cell-library-based designs,” *IEEE Transactions on Computer Aided Design*, vol. 28, no. 6, pp. 818–825, 2009.
- [17] Y. Liu and J. Hu, “A new algorithm for simultaneous gate sizing and threshold voltage assignment,” in *Proc. International Symposium on Physical Design*, 2009, pp. 27–34.
- [18] O. Coudert, “Gate sizing for constrained delay/power/area optimization,” *IEEE Transactions on Very Large-Scale Integrated Systems*, vol. 5, no. 4, pp. 465–472, 1997.
- [19] M. Rahman, H. Tennakoon, and C. Sechen, “Power reduction via near-optimal library-based cell-size selection,” in *Proc. DATE: Design Automation and Test in Europe*, 2011, pp. 1–4.
- [20] S. Lin, M. Marek-Sadowska, and E. Kuh, “Delay and area optimization in standard-cell design,” in *Proc. of the Design Automation Conf.*, 1990, pp. 349–352.
- [21] J. Fishburn and A. Dunlop, “Tilos: A posynomial programming approach to transistor sizing,” in *Proc. Intl. Conf. on Computer-Aided Design*, 1985, pp. 326–328.
- [22] A. Srivastava, D. Sylvester, and D. Blaauw, “Power minimization using simultaneous gate sizing, dual-vdd and dual-vth assignment,” in *Proc. of the Design Automation Conf.*, 2004, pp. 783–787.
- [23] L. Wei, K. Roy, and C.-K. Koh, “Power minimization by simultaneous dual-vth assignment and gate-sizing,” in *Proc. Custom Integrated Circuits Conf.*, 2000, pp. 413–416.
- [24] W. Ning, “Strongly NP-hard discrete gate-sizing problems,” *IEEE Transactions on Computer Aided Design*, vol. 13, no. 8, pp. 1045–1051, 1994.
- [25] P. K. Chan, “Algorithms for library-specific sizing of combinational logic,” in *Proc. of the Design Automation Conf.*, 1990, pp. 353–356.
- [26] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, “The ISPD-2012 discrete cell sizing contest and benchmark suite,” in *Proc. International Symposium on Physical Design*, 2012, pp. 161–164.
- [27] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. John Wiley & Sons, 1993.
- [28] R. K. Ahuja, R. K. Ahuja, D. S. Hochbaum, D. S. Hochbaum, J. B. Orlin, and J. B. Orlin, “Solving the convex cost integer dual network flow problem,” *Management Science*, vol. 49, pp. 31–44, 1999.
- [29] Q. Ma and E. Young, “Network flow-based power optimization under timing constraints in MSV-driven floorplanning,” in *Proc. Intl. Conf. on Computer-Aided Design*, 2008, pp. 1–8.
- [30] L. Li, J. Sun, Y. Lu, H. Zhou, and X. Zeng, “Low power discrete voltage assignment under clock skew scheduling,” in *Proc. Asian and South Pacific Design Automation Conference*, 2011, pp. 515–520.
- [31] Intel Threading Building Blocks. [Online]. Available: <http://www.threadingbuildingblocks.org>
- [32] Y. Lu, H. Zhou, L. Shang, and X. Zeng, “Multicore parallelization of min-cost flow for CAD applications,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 10, pp. 1546–1557, 2010.
- [33] A. Holder, C. Carothers, and K. Kalafala, “Prototype for a large-scale static timing analyzer running on an IBM blue gene,” in *IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2010, pp. 1–8.