

PaRS: Fast and Near-Optimal Grid-Based Cell Sizing for Library-Based Design

Tai-Hsuan Wu, and Azadeh Davoodi
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
1415 Engineering Dr., Madison WI 53706
Email: {twu3,adavoodi}@wisc.edu

Abstract—We propose PaRS, a parallel and randomized tool which solves the discrete gate sizing (cell sizing) problem on a grid. PaRS is formulated based on an optimization framework known as Nested Partitions which uses parallelism and randomization from a novel perspective to better identify the optimization direction. It achieves near-optimal solutions for minimizing total power and area subject to meeting a delay constraint. The embarrassingly-parallel nature of PaRS makes it highly efficient. We show small algorithm run-times, in at most minutes for circuits with over 47,000 cells. We make comparison with the optimal solution generated by a custom and parallel branch-and-bound algorithm. Consequently, we are able to generate the optimal solution within hours. While the optimal algorithm uses up to 200 nodes in our grid, PaRS achieves its speedups and near-optimal solutions using only 20 nodes.

I. INTRODUCTION

Cell sizing for library-based designs is a popular technique which allows exploring trade-offs between delay versus area or power. Traditionally cell sizing is viewed as a continuous optimization problem [10], [1]. The continuous size solutions are typically rounded to their closest discrete options in the library. This results in deviation in the estimated delay and area/power of the continuous solution. This deviation might be low if the cell sizes are uniformly spaced. However now-a-days, to effectively explore the area/power versus delay tradeoffs, cell libraries are designed to be geometrically spaced [2]. Consequently the rounding errors become significant [9].

Previous works on cell sizing includes [8] which is based on multi-dimensional descent optimization. Recently Hu et al [9] propose a dynamic programming approach to cell sizing which can systematically search the solution space. The solution quality of [9] is shown to be better than [8]. However, the reported run times of [9] are very high—even though search is limited to those cell sizes that are close to the solution generated by continuous optimization. Furthermore, it is unclear how close the solution of [9] might be from the optimal.

In this paper, we propose PaRS, a parallel and randomized cell sizing algorithm. PaRS is fast and generates near-optimal solutions. It minimizes total power and area subject to meeting a delay constraint.

PaRS is formulated based on a parallel and randomized meta-heuristic known as Nested Partitions which we introduce in the CAD area. This optimization framework has been proposed in [7]. Here, we formulate it to solve the cell sizing problem. PaRS is an iterative algorithm. At each iteration the size combinations of multiple cells are explored simultaneously. The “most promising” assignment for these variables is then found with the aid of random sampling on the unassigned variables. Random sampling allows quick identification of the correct optimization direction (i.e., variable assignment). This is the main advantage of PaRS over a branch-and-bound alternative.

After choosing a promising assignment at an iteration, the remaining assignments are added to a region known as “complementary region” which we expect not to contain the optimal solution. At each iteration, we also evaluate the complementary region using random sampling. If we find a variable assignment with lower cost in the complementary region, we backtrack to a previous stage.

We also generate the provably optimal solution using a custom branch-and-bound (BB) algorithm. We develop custom strategies to significantly reduce the search time by avoiding the large-sized infeasible regions. We show parallel implementation of PaRS and custom BB on a grid of heterogeneous computers, and discuss the “Master-Worker” (MW) framework [4] for grid-based implementation.

While the number of computers in BB dynamically changes during optimization (reaches 200 in our experiments), in PaRS, we always fix the number of computers a-priori. We show only 20 computers are sufficient to generate near-optimal solutions (within 1%) of optimal. The runtime of PaRS is often in seconds while the runtime of parallel BB is in hours. The main advantage of PaRS over BB is incorporation of our variable selection at an iteration with our sampling scheme such that the impact of unassigned optimization variables can be accurately assessed. This helps efficient identification of the correct optimization direction (unlike BB).

While the fast runtime of PaRS is due to parallel implementation, we are emphasizing that many of the existing algorithms for CAD applications have serious limitations for parallelism. Consequently, we argue that comparing the runtimes of parallel PaRS with an alternative that can hardly be parallelized is fair. The embarrassingly parallel nature of PaRS is an advantage over many existing algorithms. Our reported runtimes are yet conservative because of Condor [11], the program that manages the use of computers on our grid. Condor’s philosophy is to use idle cycles of the machines in the network.

In the following sections, we start with an overview of cell sizing and Nested Partitions in Sections II and III. Sections IV, VI and discuss formulation of PaRS and BB for cell sizing, and their grid implementations. We end with simulation results and conclusions.

II. PROBLEM FORMULATION AND PREVIOUS WORK

Given a combinational circuit with n cells, let w_i denote the size of cell i . Assume $w_i \in Q_i$, where Q_i is the set of discrete sizes for cell i provided by the standard-cell library. Also, for each cell size, the library provides leakage and delay numbers. In the cell sizing problem, we would like to minimize a linear combination of total area and power of the circuit which can be expressed as a weighted summation of individual areas and powers of the cells. We require the total circuit delay D_{cir} , to not exceed a target delay constraint D_{tar} . This optimization problem is formulated below:

$$\begin{aligned} \min_{w_i} \sum_{i=1}^n C_i & \quad (1) \\ \left\{ \begin{array}{l} C_i = \alpha \cdot (a_i \cdot w_i) + \beta \cdot (P^{dyn}(w_i) + P^{stat}(w_i)) \quad \forall i \\ D_{cir}(\vec{w}) \leq D_{tar} \\ w_i \in Q_i \quad \forall i \end{array} \right. \end{aligned}$$

Other parameters such as slew rates and capacitive loadings are provided by the cell library. Wire delays are also incorporated and this combinational formulation can be extended to the sequential case.

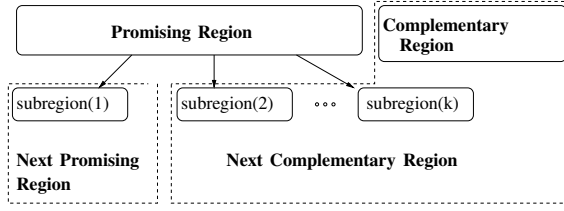


Fig. 1. Overview of optimization using Nested Partitions

To solve this optimization problem, many previous works first obtain continuous convex delay and power models for individual cells [6]. This can be done via fitting the data from the discrete cell-library to predefined convex forms and assuming w_i is continuous [6], [3]. The obtained formulation will then describe the continuous gate sizing problem which can efficiently and optimally be solved using convex programming techniques [1]. The continuous solution is then discretized to the cell sizes available by the library. Alternatively, [8], [9] directly solve the discrete problem. [9] uses an intuitive observation that the optimal cell sizes are close to the ones generated by solving the relaxed problem. In our simulations, we verify this to be a correct intuition. We also solve the discrete problem formulation.

III. OVERVIEW OF NESTED PARTITIONS

Nested Partitions [7] is a powerful optimization framework that combines adaptive global sampling with formal methods and local heuristic search. It uses a flexible partitioning method to divide the search space into regions that can be analyzed individually and then evaluates the results from each region to determine how to proceed with the search. The underlying idea of Nested Partitions is to focus the computation effort on a subset of search space that is deemed “promising”, while also keeping an eye on the rest. The search space is always viewed as the disjoint union of a promising region and a complementary region, as shown in Figure (1). This is in contrast to BB in which the inferior branches are pruned out based on generated bounds. At each iteration of Nested Partitions we apply four steps:

1. Partitioning: Current promising region is divided by k subregions.
2. Random Sampling: After defining the subregions, we randomly make feasible assignments to the unassigned optimization variables. Each such assignment is referred to as a sample. At each iteration, we sample from the entire solution space (i.e., each sub-region and the complementary region).

3. Find Promise Index: Using the sampling information, each sub-region is assessed for its potential to contain a “good” solution. Also the complementary region is assessed for the same criterion. The region that is assessed to have the highest potential to contain a good solution is designated as the next promising region, and remaining regions are bundled together to form the next complementary region.

4. Backtracking: If partitioning and assessment steps are effective, the next promising region is expected to often be among the subregions of the current one, resulting in the promising region to get smaller with each iteration (hence the word “nested partitions”). However, at a given iteration, it is possible that the complementary region is assessed to be more promising. In such a case, we backtrack to a previous optimization stage, for example to the previous iteration, and redo the partitioning with the information hitherto obtained.

To list some of highlighting features of Nested Partitions, first, the backtracking feature avoids getting trapped at local minima. Second, evaluation of different subregions as well as global sampling at each iteration can be done in parallel. Third, existing heuristics and formal methods can get incorporated in Nested Partitions—for example to define a partitioning or sampling strategy. This hybrid feature can help achieve parallel CAD because it can reuse the existing algorithms.

IV. PARS: NESTED PARTITIONS FOR CELL SIZING

We start by selecting a subset of the n variables at each iteration and consider all of their size combinations. Let k indicate the number of selected variables at each iteration. Also assume q size options are available for each cell. Therefore the number of subregions are always equal to q^k . Figure (2) shows an example. Here $k=5$ and $q=2$. We explore all 32 possibilities as 32 different subregions. With the aid of sampling, suppose we choose subregion i in the first iteration as the most promising. In the second iteration, we again choose 5 new w_i variables and explore all their size combinations. Also, we put together all remaining subregions as the complementary region.

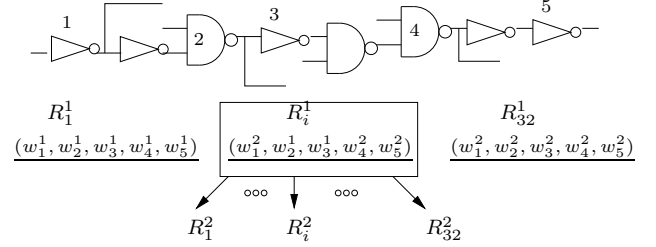


Fig. 2. Exploring size combinations of $k=5$ new cells at each iteration for $q=2$ sizes per cell and selecting the most promising one

Since we explore all size combinations, we know one of the subregions should be the promising one. However, given many unassigned variables (specially at the first few iterations), it would be difficult to choose the correct one. Therefore a good partitioning and sampling scheme should select w_i variables and evaluate each subregion to facilitate the identification of the promising assignment. Next, we define some notations to describe our scheme:

- Let X_i indicate the set of variables that are already “processed”. For example at iteration 2, we have variables w_1 to w_5 .
- Let Y_i indicate the “in-process” variables at iteration i . For example at iteration 2, w_6 to w_{10} might be in process ($|Y_i| = k$).
- Let Z_i indicate “not-processed” variables. So $|Z_i| = n - |X_i| - |Y_i|$.
- Let R_j^i indicate assignment of $X_i \cup Y_i$ variables at iteration i , defined by subregion j (see Figure (2)). R_j^i has two components:
 - Assignment of X_i is propagated by $R_b^{(i-1)}$ where b is the index of the promising region at iteration $i - 1$.
 - Assignment of Y_i is defined by subregion j at iteration i .

We now present the four steps of Nested Partitions formulated for cell sizing. We discuss our rationale to increase the likelihood of identifying the promising region, at the end of this section.

1. Partitioning: During partitioning at iteration i , we select the Y_i variables based on Algorithm 1. In summary, we choose Y_i s to be the “most critical” cell sizes of the “most critical” path, if all the remaining $w_j \notin X_i$ variables are set to their minimum sizes.

Algorithm 1: Partitioning at iteration i

Input: $X_i, R_b^{(i-1)}$ (processed variables and their assignment)

Output: Y_i : the k selected variables and the “Stop” flag

- Keep $X_i = R_b^{(i-1)}$; GStop=true.
- $\forall w_j \notin X_i$, temporarily set $w_j = w_j^{min}$.
- Identify Pl_i , the longest timing path at iteration i .
- If $Delay(Pl_i) > D_{tar}$:
 - Select Y_i to be the k size variables corresponding to the cells $j \in Pl_i$ with higher criticality factor $CF_j = \frac{\delta P_i}{\delta C_i} \times (1 - S_i^N)$;
 - GStop=false.
- return Y_i and the GStop flag.

We identify the longest path (denoted by Pl_i), if each $w_j \notin X_i$ takes its minimize size w_j^{min} and $X_i = R_b^{(i-1)}$. This gives the minimum cost assignment for $X_i = R_b^{(i-1)}$ which might be infeasible.

If the delay of Pl_i is infeasible, we choose Y_i to be the top k cells that have the higher value of criticality factor CF_i (note the partitioning step only selects the Y_i variables and does not explore their assignments). The CF_i expression is a variation of a popular sensitivity-based factor which has been used in several cell sizing papers such as [5]. In the CF_i expression δ_{D_i} and δ_{C_i} reflect the change in delay and cost of cell i for sizes w_i^{max} and w_i^{min} . Also S_i^N is the slack of cell i normalized to be between 0 and 1 over all cells in Y_i . CF_i is between 0 and 1 for each cell i and we look for the cell of highest criticality factor. The algorithm returns a global stop flag “GStop” which is used to define the stopping criteria. GStop is set to true only if the delay of Pl_i is feasible for assignment $X_i = R_b^{(i-1)}$ and minimum size Y_i . In this case, Y_i is the optimal assignment because it yields to a feasible solution with minimum cost. These X_i and Y_i assignments are possibly the global optimum, if they are not found to be sub-optimal during sampling of the complementary region. We discuss the rationale behind our partitioning in conjunction with sampling in Subsection (IV-A).

2. Random sampling: This step is applied after partitioning at each iteration. We apply two types of sampling: 1) sampling each of the q^k subregions, 2) sampling the complementary region. The objective of sampling is to get an estimate of minimum cost solution attainable at each subregion, as well as at the complementary region.

1) Sampling the Subregions: To sample the subregions, we first identify only a subset of the unprocessed Z_i variables for sampling. We denote this subset by S_i . The remaining Z_i variables will be set to their minimum size during sampling. Algorithm 2 shows selection of S_i . We first identify a set of primary outputs O , to which the current X_i and Y_i variables connect to. We then select all the nodes in the subtrees of O that are in Z_i .

Algorithm 2: Finding candidate cells S_i for sampling at iteration i
Input: X_i, Y_i, Z_i (processed, in-process, not-processed variables)
Output: S_i candidates selected for sampling

1. $S_i = \emptyset$.
 2. Find a set of primary outputs $\{O|j \xrightarrow{P} O, \forall w_j \in X_i \cup Y_i\}$.
 3. Find cells $\{l|l \xrightarrow{P} O\}$ and update $S_i = S_i \cup w_l$.
 4. $S_i = S_i \cap Z_i$.
-

The selected S_i variables for sampling represent the timing critical part of the circuit which affects the feasibility of the generated samples. This is because X_i and Y_i are chosen to be the most critical cells of the most timing critical path. The remaining cells in Z_i (and not in S_i) are set to minimum size during sampling to have the smallest contribution to cost without affecting feasibility. Algorithm 3 shows sampling of subregion j at iteration i . It returns an estimate of minimum attainable cost at region j (denoted by RC_j^i).

Algorithm 3: Sampling subregion j at iteration i
Input: X_i, Y_i, Z_i, S_i, j , and M number of samples
Output: RC_j^i (estimate of minimum cost solution of subregion j)

1. $X_i \cup Y_i = R_j^i$ and $RC_j^i = \infty$
 2. $w_l = w_l^{min}, \forall l \in Z_i$;
 3. For M steps

While $D_{cir}(\vec{w}) > D_{tar}$

Select $l \in S_i$ with maximum CF_l
 $r = \text{RAND}(0, 1)$
 If $(r \leq CF_l)$ //select cell size closest to random r
 $f = \text{round}(r \times q) \rightarrow w_l = w_l^f$
 $S_i = S_i - w_l$

 Compute $\text{Cost}(\vec{w})$ and $RC_j^i = \min(RC_j^i, \text{Cost}(\vec{w}))$
 4. Return RC_j^i
-

We first set $X_i \cup Y_i = R_j^i$, and Z_i to their minimum sizes. At this point, we have an infeasible assignment. We then apply sampling at M rounds. At each round, we randomly increase the size of the cells with highest criticality factor one by one and each time apply timing analysis until the resulting circuit timing becomes feasible. We return the minimum cost over all M samples.

2) Sampling from the Complementary Region: We first generate samples for X_i and Y_i . If these samples do not belong to any of the assignments of the subregions (samples $\notin \bigcup R_j^i$), we randomly generate samples for Z_i . We record minimum cost of M feasible samples. We denote this assignment by R_{CC}^i and its cost by RC_{CC}^i .

3. Promise Index: The promise index is given by the equation below:

$$b = \arg(\min_{j=\{1,2,\dots,k,CC\}} R_j^i) \quad (2)$$

If $b \neq CC$, we set subregion b as the next promising region. We combine the remaining subregions and complementary region as the next complementary region.

4. Backtracking: If $b = CC$, we backtrack to iteration $i - 1$. We set X_i as given by R_{CC}^i and move to a new partitioning step next.

5. Stopping Condition: If during an iteration, the “GStop” flag is set to true by Algorithm 2, it indicates that we have found the minimum cost assignment for given X_i . We then apply random sampling and find the promise index. If the promise index is not the complementary region, PaRS terminates. If the above never happens, we continue the iterations until all variables are processed. So, in the latter case, iteration i is the last iteration if $|Z_i|=0$.

A. Rationale

Recall we generate all the q^k assignments of Y_i variables at iteration i and require having a high likelihood to choose the optimal assignment for Y_i . Since we always have unassigned Z_i variables, we need the sampling scheme to have a high likelihood to correctly identify the promising region. Below we summarize properties of our scheme to discuss achieving the above-stated objective.

1) During partitioning, Y_i variables are chosen from the “most timing critical” path (if $w_l = w_l^{min}, \forall w_l \notin X_i$). Moreover, Y_i are the ones with higher criticality factor CF . So they have less slack but result in more decrease in delay and less increase in cost if they are upsized.

2) The objective of sampling of each subregion is to get an estimate of the attainable minimum cost feasible solution in that subregion in order to identify the most-promising one correctly. This should be done by sampling the unprocessed Z_i variables. However, to find the minimum cost at a subregion, we don’t need to explore all the possible Z_i samples. Instead, we start from the minimum cost assignment which is infeasible and use random sampling to turn this assignment into a feasible one with small increase in cost. So the cells in Z_i with higher CF are upsized first using random sampling. Every time we up-size a cell, we check for feasibility. We obtain M different estimates of the minimum cost of a subregion. All of these are good estimates and the random nature takes care of exploring several low cost feasible solutions. The smallest estimate is selected.

We would like to emphasize that even though we use heuristics for partitioning and sampling but these heuristics only help us 1) find an intelligent order for variable selection for partitioning, 2) get a good estimate of minimum cost feasible solution for a subregion. The latter is achieved by “fixing” a minimum cost infeasible solution to become feasible using random sampling on the critical cells. The random nature if combined with enough samples (M) gives a very good estimate. We show near-optimal solutions for $M=100$, even for largest benchmark with few backtracking steps. Overall, we use heuristics to facilitate the formal search of solution space.

V. BRANCH-AND-BOUND FOR CELL SIZING

Given a level i of the search tree, we compute a lower and an upper bound of the cost for each node at level i . We then prune the inferior nodes using these bounds. Given a partial variable assignment X corresponding to a node at level i in the search tree, the upper bound is found by finding any feasible solution for given X . The lower bound is found via the continuous cell sizing for given X .

Customized Rules for Effective Pruning: Similar to BB, we always select k cells and explore all their size combinations at each level i in the BB search tree. For example, in Figure (2), we select variables w_1 to w_5 at first. Here we give two rules on selection of the variables at each level of BB tree which are listed below:

1) *Longest Path Rule (LPR)*: Here we choose the k variables similar to partitioning step of PaRS given in Algorithm 1. We set all the unprocessed size variables to minimum size, and identify the most timing critical path Pl in the circuit. We then select the top k variables with highest criticality factor on this path. In the consecutive iteration, we select the next k variables from the same path Pl until all the variables on Pl are processed. This “depth-wise” variable ordering helps identifying the large infeasible portions of the search space faster. Note this variable selection scheme is not exactly the same as in the partitioning step of PaRS (PaRS dynamically updates and selects critical paths at each iteration).

2) *Nearest Neighbor Rule (NNR)*: After all cells on P_l are selected, we choose the cells that are “neighbors” of P_l which are $\{\forall w_l | fanin(l) \in P_l, l \notin P_l\}$. The reason is because at this point we already explored the feasible assignments on P_l assuming their neighbors have minimum size. Increasing the size of neighbors further makes many of those assignments to become infeasible and allows significant pruning. Once such neighbors are explored, we apply the *LPR* to find another critical path. *We tried other pruning rules that are based on breadth-wise ordering of size variables but did not find them useful for pruning of the infeasible regions.*

We evaluated our rules on some of the largest benchmarks. We applied them to an exhaustive search algorithm and did not apply BB. We compared the total search size and our search size when *LPR* and *NNR* are applied. When choosing $k=5$ variables using *LPR* and *NNR*, out of the q^k possibilities, we found only r_1 and r_2 (for *LPR* and *NNR* respectively) to be feasible at each node. The average percentage of $\frac{r_1}{q^k}\%$ and $\frac{r_2}{q^k}\%$ were both less than 0.1% for $q=2$. Note the nodes in lower levels of the branch-and-bound tree had a much lower percentage than the higher levels, to result in this average percentage. We observed that these rules for variable selection allow eliminating a large portion of the q^k assignments at each node. Detailed results are eliminated due to lack of space.

Efficient Checking for the Best Objective Value: At each node in the search tree we initially check if we can obtain a feasible solution if we set the unprocessed sizes to minimum size. If we find such minimum cost feasible solution we don’t branch on that node further.

VI. PARS AND BB FOR TWO CELL SIZES

As we verified by simulation, there is a high correlation between the sizes generated from continuous sizing and the optimal discrete cell sizes. This assumption is also made in [9]. In such case we consider only two sizes for each variable which are the ones that are immediately before and after the continuously generated size for that cell from the library. We apply BB-2 for our customized BB using two variables. For PaRS, we also considered the variation with two sizes. We modified our algorithms such that during partitioning and sampling of subregions we assumed two sizes for each variable. During sampling of complementary region we sampled all the sizes.

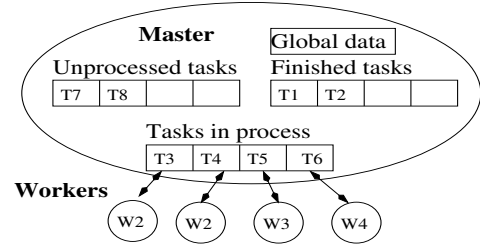


Fig. 3. Overview of the Master-Worker framework

VII. PARALLEL IMPLEMENTATION

We now discuss parallel implementation of PaRS and BB on a grid of heterogeneous computers. We first explain our framework for managing the machines in a heterogeneous grid. We then discuss specifics of implementation of BB and PaRS.

Condor System for High Throughput Computing: Condor system [11] manages collection of machines in a network to achieve high throughput computing. The default policy is to assign jobs to computers in the network while their owners are away. Consequently, a large computational power is available by exploiting these otherwise-unused cycles, without interrupting the machine owners. These features promote large-scale (and green) computing.

Master-Worker Enabling Grid-Based Optimization: Master-Worker (MW) is a software framework for parallel computing which is built on top of the Condor system [4]. The concept is that we first break up a large computation into several small tasks to be executed in parallel. Then a “Master” processor sends these tasks to the “Worker” processors, and retrieves the results when the tasks are finished. The MW APIs interact with Condor to find computing resources for the tasks, handle communication between Master and Workers, and to re-assign tasks if job migration is necessary. Figure (3) shows the MW framework to maintain a task list, and the relationship between Master and Worker processors. Note that the Master processor is owned by us and can not be interrupted. Together with Condor, MW provides a dynamic and opportunistic framework for management of computational resources for implementation of parallel algorithms.

Implementation of Parallel PaRS using MW: PaRS is an ideal (embarrassingly parallel) optimization algorithm to be implemented using MW. The computation on each subregion can be considered as a task to be done by a Worker processor at each iteration. We have q^k Workers for q^k subregions, and one Master processor that receives the information from the Workers and decides the next promising region. We also have one (or more) Workers that sample the complementary region. The number of workers are fixed and usually small in PaRS.

Implementation of Parallel BB using MW: In parallel BB, the Workers compute the upper and lower bounds for the nodes of the search tree. The Master receives the bound information from the Workers, applies pruning to decide the next set of workers. The Master also applies custom variable ordering that we discussed before assigning tasks to Workers. The main issue in parallel implementation of BB is the exponential increase in the number of nodes in the search tree in two consecutive levels [4]. Here, the Master initially allocates as many Workers as possible and assigns tasks to them. When moving to a new level at the search tree, if the number of Workers are not sufficient, the Master maintains a list of current tasks and their corresponding Workers. It also maintains a list of tasks to be assigned upon availability of resources. If a current task is terminated and its Worker becomes free, or if a new resource becomes available in the network, the Master allocates this as a Worker and assigns a new task from the wait list. Figure (3) illustrates this process. Using MW, we can *dynamically* assign workers in an opportunistic manner.

Bench	#cells	POWER			RUNTIME						AVE #WORKERS	
		$\frac{BB2}{OP}$ %	$\frac{NR}{OP}$ %	$\frac{PaRS}{OP}$ %	OP(s)	BB-2(s)	PaRS(s)	$\frac{OP}{PaRS}$	$\frac{BB-2}{PaRS}$	PaRS(#BT)	BB	BB-2
c432	136	0.00%	7.99%	0.00%	51	22	1	50.7X	21.6X	0	75.4	50.0
c499	163	0.30%	18.37%	2.11%	390	120	4	97.5X	29.9X	1	56.6	81.1
c880	208	0.04%	12.66%	0.04%	63	21	2	31.6X	10.5X	0	42.6	60.1
c1355	205	0.00%	2.66%	0.00%	60	45	4	15.0X	11.4X	1	80.3	60.0
c1908	199	0.15%	27.63%	0.66%	6844	111	6	1140.7X	18.4X	2	142.0	99.8
c2670	472	0.06%	11.30%	0.70%	1258	45	8	157.2X	5.6X	2	57.5	71.4
c3540	479	0.03%	17.17%	1.65%	78240	11380	17	4602.3X	669.4X	1	157.8	125.3
c5315	705	0.37%	3.34%	1.02%	2156	581	21	102.6X	27.7X	5	105.7	53.6
c6288	1256	0.16%	2.13%	0.62%	3992	1239	45	88.7X	27.5X	0	114.9	92.4
c7552	822	0.00%	4.28%	0.09%	1870	682	14	133.6X	48.7X	3	101.9	127.2
s1488	307	0.00%	13.41%	2.33%	125671	88	8	15708.8X	11.0X	1	114.7	76.7
s1494	309	0.00%	8.85%	1.83%	3233	197	5	646.5X	39.3X	2	111.7	85.4
s5378	930	0.07%	7.09%	0.14%	2332	133	19	122.7X	7.0X	2	95.5	78.5
s9234	740	0.14%	50.60%	0.14%	3105	341	9	345.0X	37.9X	1	95.8	98.0
s15850	617	0.06%	10.19%	0.53%	2872	420	7	410.3X	60.0X	1	112.6	101.4
s35932	7260	0.00%	14.79%	0.00%	3174	159	142	26.1X	1.1X	0	108.1	79.0
s38584	6950	0.00%	2.93%	0.08%	3700	576	462	6.9X	1.2X	4	113.8	94.4
b18	47191	0.17%	13.12%	0.84%	188862	41721	1117	169.1X	37.4X	11	189.8	179.8
b20	15699	0.11%	11.40%	1.02%	103384	19528	613	168.7X	31.9X	3	167.2	124.2
b22	24484	0.09%	1.29%	0.41%	138473	23112	681	203.3X	33.9X	6	173.7	128.3
Ave.		0.09%	12.63%	0.70%								

TABLE I

VIII. SIMULATION RESULTS

We used a standard-cell 90nm TSMC library and initially obtained convex cell delay and leakage power models using curve fitting as done or assumed in many previous works such as [9], [6]. Convex delay/power models are necessary in continuous sizing which can be used to bias the search. Our cost was the total power. We implemented the following approaches:

1. LB: Here we assume each cell size i is a continuous quantity within S_i^{min} and S_i^{max} provided by the library. We use the continuous sizing solution as a lower bound (LB) for minimum attainable cost. LB is solved as a convex program using MOSEK (see Section II).

2. NR: In the Nearest Rounding approach, we round the solution of LB to closest size. This solution violates the timing. We then upsize the cells one by one based on their criticality factor and perform static timing analysis every time until delay is satisfied.

3. OP: This is parallel and optimal BB. The discrete cell sizes were given by the library and BB was done over all possible cell sizes. The number of sizes per cell in the library (q) varied between 3 to 10 and on-an-average was 4.48. OP generates the optimal solution.

4. BB-2: We also apply BB assuming each cell has only two sizes (BB-2) which are the ones immediately before and after the solution of LB. Refer to Section (VI) for more details. We verified by simulation that the percentage difference between the sizes generated by OP and BB-2 for each cell is very low (under 5% on average). We skip detailed results due to lack of space.

5. PaRS (our proposed algorithm): We selected two sizes like BB-2. We chose $k=4$ variables and $M=100$ samples.

Parallel implementations of OP, BB-2, and PaRS were done using Condor and MW software on a grid (see Section VII for details). The Master machine had a 2.2GHz processor with 2GB memory running Linux. Our program and the MW APIs were in C++. We used ISCAS85', '89, '99 combinational and sequential benchmarks which were synthesized using Synopsys Design Compiler.

Comparison of Total Cost: Table (1) reports the total cost of each approach. Columns 3, 4, 5 list the percentage different between BB2, NR, and PaRS, with respect to OP. On average NR is **12.6%** away from the optimal while we are within **1%** of the optimal. BB2 also generates near-optimal solutions. All timing constraints were met.

Comparison of Runtime: Table (1) also reports the (wall clock) runtime including the communication overhead. PaRS is done in seconds, while BB-2 and OP are done in hours. The number of backtracking steps in PaRS is given in Column 11.

Comparison of Number of Workers: The last 2 columns of Table (1) show the average number of workers for BB and BB-2. The maximum number of workers in OP, BB-2 and PaRS were 192, 187 and 20, respectively.

IX. CONCLUSIONS AND FUTURE WORK

We presented PaRS, a parallel and randomized algorithm for cell sizing on a grid. It achieves near-optimal solution mostly within seconds for benchmarks of varying sizes and topologies using only 20 computers. We also presented a custom and parallel BB algorithm for cell sizing which uses up to 200 computers (at a time) to find the optimal solution. Overall, by incorporating random sampling, PaRS can tremendously outperform BB in identifying the correct optimization direction without compromising the solution quality using a fraction of resources. It can achieve very fast runtimes, despite the communication overhead on the grid, due to its embarrassingly parallel nature. In future we plan to experiment with other large-scale CAD problems and assume a heterogenous grid of multi-cores as our computing platform.

REFERENCES

- [1] C.-C.P. Chen, C.C.N. Chu, and D.F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *ICCAD*, pp. 617–624, Nov 1998.
- [2] F. Beaufort, P. Kudva, D. Kung, and L. Stok, "Gate size selection for standard cell libraries," *ICCAD*, pp. 545–550, 1998.
- [3] H. Tennakoon, and C. Sechen, "Efficient and accurate gate sizing with piecewise convex delay models," *DAC*, pp. 807–812, Jun 2005.
- [4] J.-P. Goux, S. Kulkarni, J.T. Linderth, and M.E. Yoder, "Master-Worker: an enabling framework for applications on the computational grid," *Cluster Computing*, vol. 4, pp. 63–70, 2001.
- [5] J.P. Fishburn, and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *TCAD*, pp. 326–328, Nov 1985.
- [6] K. Kasamestry, M. Ketkar, and S.S. Sapatnekar, "A new class of convex functions for delay modeling and their application to the transistor sizing problem," *TCAD*, vol. 19, no. 7, pp. 779–788, Jul 2000.
- [7] L. Shi, and S. Ólafsson, "Nested partitions method for global optimization," *Operations Research*, vol. 48, no. 3, 2000.
- [8] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *TVLSI*, vol. 5, no. 4, pp. 465–472, 1997.
- [9] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," *DAC*, pp. 847–852, 2007.
- [10] S.S. Sapatnekar, V.B. Rao, P.M. Vaidya, and S.M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *TCAD*, vol. 12, no. 11, pp. 1621–1634, Nov 1993.
- [11] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – A Distributed Job Scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press, October 2001.