# CIRCUIT OPTIMIZATION USING EFFICIENT PARALLEL PATTERN SEARCH

A Thesis

by

SRINATH SUDHARSHAN NARASIMHAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2010

Major Subject: Electrical Engineering

# CIRCUIT OPTIMIZATION USING EFFICIENT PARALLEL PATTERN SEARCH

A Thesis

by

SRINATH SUDHARSHAN NARASIMHAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

| | |
|---|---|
| Chair of Committee, | Peng Li |
| Committee Members, | Sivakumar Natarajan |
| | Sebastian Hoyos |
| | Seong Choi |
| Head of Department, | Costas Georghiades |

May 2010

Major Subject: Electrical Engineering

# ABSTRACT

Circuit Optimization Using Efficient Parallel Pattern Search.

(May 2010)

Srinath Sudharshan Narasimhan, B.E., College of Engineering Guindy, Anna University

Chair of Advisory Committee: Dr. Peng Li

Circuit optimization is extremely important in order to design today's high performance integrated circuits. As systems become more and more complex, traditional optimization techniques are no longer viable due to the complex and simulation intensive nature of the optimization problem. Two examples of such problems include clock mesh skew reduction and optimization of large analog systems, for example Phase locked loops. Mesh-based clock distribution has been employed in many high-performance microprocessor designs due to its favorable properties such as low clock skew and robustness. However, such clock distributions can become quite complex and may consist of hundreds of nonlinear drivers strongly coupled via a large passive network. While the simulation of clock meshes is already very time consuming, tuning such networks under tight performance constraints is an even daunting task. Same is the case with the phase locked loop. Being composed of multiple individual analog blocks, it is an extremely challenging task to optimize the entire system considering all block level trade-offs.

In this work, we address these two challenging optimization problems i.e.; clock mesh skew optimization and PLL locking time reduction. The expensive objective function evaluations and difficulty in getting explicit sensitivity information make these problems intractable to standard optimization methods. We propose to explore the recently developed asynchronous parallel pattern search (APPS) method for efficient driver size tuning. While being a search-based method, APPS not only provides the desirable derivative-free optimization capability, but also is amenable to parallelization and possesses appealing theoretically rigorous convergence properties.

In this work it is shown how such a method can lead to powerful parallel optimization of these complex problems with significant runtime and quality advantages over the traditional sequential quadratic programming (SQP) method. It is also shown how design-specific properties and speeding-up techniques can be exploited to make the optimization even more efficient while maintaining the convergence of APPS in a practical sense. In addition, the optimization technique is further enhanced by introducing the feature to handle non-linear constraints through the use of penalty functions. The enhanced method is used for optimizing phase locked loops at the system level.

# DEDICATION

This thesis is dedicated to my parents and grandparents, especially to my late beloved grandfather Mr. N.S. Srinivasan.

# ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Peng Li, and my committee members, Dr. Hoyos, Dr. Choi, and Dr. Sivakumar, for their guidance and support throughout the course of this research. I am especially thankful to Dr. Peng Li for providing me with such an excellent opportunity to be able to do something novel during my master's program and also for providing me with financial support during the course of my stay here. I am also very thankful to Dr. Sivakumar for helping me in settling down in College Station in addition to his guidance for my research.

Thanks also go to my colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. I am also thankful to Dr. Tamara G. Kolda of Sandia National Laboratories for her guidance. I am also extremely thankful to my room-mates at College Station and my friends for making my stay an unforgettable experience.

Finally, thanks to my parents and grandparents for their encouragement, support and patience.

# NOMENCLATURE

| | |
|---|---|
| APPS | Asynchronous Parallel Pattern Search |
| CAD | Computer Aided Design |
| CPPLL | Chare Pump Based PLL |
| CS&E | Computer Science and Engineering |
| EDA | Electronic Design Automation |
| PLL | Phase Locked Loop |
| PRIMA | Passive Reduced Order Interconnect Macro Modeling Algorithm |
| SQP | Sequential Quadratic Programming |
| SVM | Support Vector Machine |
| VLSI | Very Large Scale Integration |

.

# TABLE OF CONTENTS

CHAPTER Page

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER I

# INTRODUCTION

There are many kinds of circuit optimization problems. Some of them can be solved by traditional optimization techniques like SQP, Linear programming, etc since there exist closed form expressions for the objective function of these problems. But there are some circuit optimization problems which are so complex that it is impossible to obtain a closed form expression for the objective function. The objective function can only be evaluated through complex and time consuming simulations for those circuits. In such cases, it is necessary to look at non-traditional optimization techniques which do not require any closed form expression for the objective function as input. In most cases, such techniques are heuristics which may or may not converge to an optimum solution. Otherwise, it might be a traditional technique but the derivative information is computed internally through multiple time consuming simulations. It thus makes sense to look at optimization techniques which are guaranteed to converge but do not need to waste time on computing the derivative information either. This work mainly looks at such an optimization technique called Asynchronous Parallel Pattern Search (APPS). It has the advantage of being derivative free and having theoretically rigorous convergence properties apart from being inherently parallel. Two complex circuit optimization problems which have been very difficult to solve using traditional techniques are targeted in this work – clock mesh skew optimization and PLL locking time reduction.

---

The journal model is *IEEE Transactions on Automatic Control*.

Mesh based clock distribution networks have been used in high performance microprocessor designs as the global clock distribution strategy [1], [2] because of their low local skew and immunity to on chip variation. A three dimensional view of the mesh based clock network is illustrated in Fig. 1. In Fig. 1, a central chip buffer drives a tree whose leaves are the sector buffers, also known as clock drivers. Each sector buffer drives a lower level tree which is connected to the grid. Therefore, the grid is driven by multiple trees whose roots are sector buffers. The wiring redundancy of the grid not only enhances design robustness but also has the effect of smoothing out delay differences between clock sink nodes, which helps minimize clock skew.

Despite their favorable properties, mesh-based clock distributions present significant CAD challenges. An accurate clock mesh model considering full coupling effects with power/ground network may consist of up to millions of linear elements and up to hundreds of clock drivers. Simulating the circuit model alone could take up to hours of runtime. Tuning/optimizing such networks at a desirable accuracy level requires even longer time since multiple simulations are needed during the optimization. Compared to many other areas of physical design automation, clock mesh optimization has been researched to a much less extent. To alleviate design complexity, in [1], a divide-and-conquer approach is employed to tune the clock distribution network. First, the grid is cut into smaller independent linear networks. Each smaller linear network is then optimized in parallel. To compensate for the loss of accuracy induced by cutting the grid, capacitive loads are smoothed or spread out on the grid.

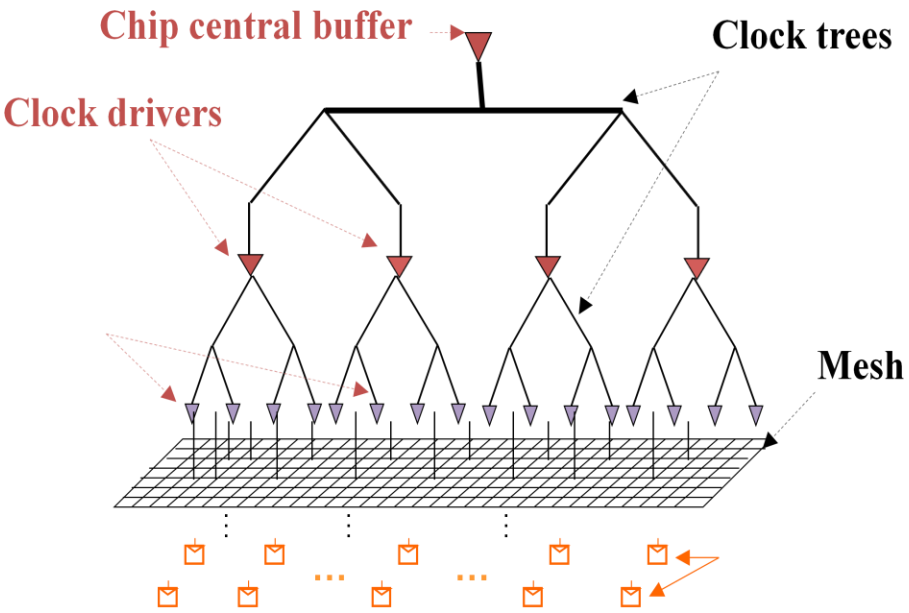


**Figure 1: Mesh Based Clock Distribution Network**

Although the efficiency of the optimization can be improved by this approach, there is no systematic way of controlling the error. In [3], very fast combinatorial techniques are proposed for clock mesh optimizations. These techniques are heuristics in nature. While previous work about the clock mesh optimization focus on the clock driver placement or wire sizing, it appears that sizing the clock drivers is also important since for very non-uniform clock load distribution, if changing the clock driver placement is impossible due to blockage or other constraints, changing the sizes of clock drivers can achieve the same or even better results. Moreover the number of clock mesh drivers is relatively less compared to the size of the mesh. And tuning the tree driving the mesh is a lot easier if only the sizes of the drivers are changed and not the location. However, significant challenges arise from the need to size a potentially large set of nonlinear clock drivers

that are coupled through the large mesh network. To this end, the choice of optimization methods is critical.

Similar problems exist for phase locked loop optimization. Being composed of many smaller subsystems, the performance measures of the PLL are complex functions of the block level parameters. Again it is impossible to obtain a closed form relationship between the block level parameters and the performance measure of the overall system. The performance objective has to be computed through complex time consuming simulations. Most of the time, a hierarchical approach is employed in optimizing such large systems. The block level performance trade-offs or pareto fronts are first obtained before-hand. The pareto fronts represent the best block level performances. The entire system is then optimized across these block level performances. In other words, the block level performance measures are now the variables in the optimization. Once the best block level performances measures are determined from the system optimization, they are later translated into circuit level parameters. Still, it is an extremely challenging problem which cannot be solved by traditional optimization techniques.

In many fields of science and engineering, there are a lot of optimization problems similar to the clock mesh optimization problem and PLL system level optimization characterized by objective function evaluations through expensive computer simulations and lack of explicit derivative information. Standard continuous optimization methods such as sequential quadratic programming method have many disadvantages in solving

this kind of optimization problems. Due to the lack of explicit derivative information, continuous optimization methods compute the derivative internally by using inefficient numerical differentiation. Furthermore, these methods usually have small incremental step sizes which slow down the progress. On the other hand, simulated annealing converges to good final solution given sufficiently long time. And it has been parallelized for CAD problems before [4]. However, the runtime required by simulated annealing to reach a good final solution is often considered to be extreme long, and is thus impractical.

This work proposes to use the recent asynchronous parallel pattern search (APPS) method [5], [6] for the clock driver sizing problem. To the best of our knowledge, this is the first attempt to use APPS for circuit optimization. Based on a manager worker paradigm, the APPS method spawns off a set of trial points from the predefined search directions. These trial points are sent by the manager processor to available worker processors for objective function value evaluation via direct simulation. Then, in an asynchronous fashion, evaluated trial points (not necessarily all trial points) are collected and checked for objective function value. Depending upon completed objective function evaluations, new trial points are generated or the search step length is altered. The process repeats till convergence. The APPS method has many advantages over the aforementioned optimization methods in solving the two specific optimization problems – clock mesh skew reduction and PLL system level optimization. First, no derivative information is needed in APPS. Furthermore, the pattern search based approach is fully

parallelizable. In our experiments, we observed that running the APPS method in parallel mode gives close to linear speedup over the serial mode. It is noteworthy that as a search-based method, APPS has an appealing theoretical convergence property. Under certain mild conditions, APPS is guaranteed to converge to a local optimum [5], [6] and hence it is well suited for tuning of clock driver sizes.

Although the original APPS method is significantly more efficient compared to other alternative optimization methods, two domain-specific enhancements are proposed to further extend its applicability to the challenging clock mesh optimization task. By exploring specific clock mesh circuit topologies, efficient mesh modeling and simulation techniques are developed to provide quick evaluation of the objective function. The quick estimation is employed to pre-screen and preorder trial points before committing to much more expensive full simulation based evaluations. Once the estimated function values are obtained for all new trial points, they are ranked by their estimated function values. Trial points with smallest estimated function values are sent to available processors for full simulations first. In this way, the modified APPS method can find a successful trial point much faster every iteration thus speeding up the entire optimization procedure. The second enhancement is adding additional search directions. By exploring the theoretical convergence properties of APPS, it is found that one or more additional promising search directions can be introduced that can potentially accelerate the optimization process. These additional search directions are estimated by the proposed efficient modeling and simulation techniques. These directions are subjected to mild

convergence requirements such that they can be aggressively estimated without interfering with the theoretical APPS convergence. Experimental results show that for the clock driver sizing problem, the proposed method significantly outperform the traditional sequential quadratic programming (SQP) based method [7]. It achieves much better final solution in less time compared with the SQP method. Furthermore, the application-specific enhancements can achieve more than two times speedup over the original APPS method for a set of clock meshes.

In the case of the PLL system level optimization, there are non-linear constraints enforced by the block-level performance trade-offs. The default APPS package does not have the feature to handle non-linear constraints. In order to solve the PLL optimization problem, we can use some techniques to make APPS handle these constraints. The use of penalty functions is one such technique which may be employed to make APPS handle non-linear constraints. It transfers the non-linear constraint into the objective function through a penalty parameter transforming the original non-linearly constrained probably into a linearly constrained one. A series of such linearly constrained problems are sequentially solved with progressively increasing penalty parameter values. The method progresses in this fashion until the constraints do not violate.

The rest of the thesis is organized as follows. In Chapter II, the basic algorithm of APPS is introduced. In Chapters III and IV, the two proposed improvements to the APPS algorithm are explained. The results of applying APPS to clock mesh optimization are

presented in Chapter V. In Chapters VI and VII, techniques for handling non-linear constraints are introduced followed by the enhanced APPS optimization algorithm with the non-linear constraint handling feature. In Chapter VIII, the phase locked loop and its optimization technique is explained. In Chapter IX, the PLL optimization results are presented. The thesis is concluded in Chapter X.

# CHAPTER II

# ASYNCHRONOUS PARALLEL PATTERN SEARCH

APPS is a derivative free search based optimization method which is best suited for solving problems whose objective functions are evaluated by complex simulations and also lack explicit derivative information [5], [6]. APPS solves unconstrained, bound or linearly constrained nonlinear optimization problems. The bound constrained problem is given by

$$\min_{x \in R^n} f(x) \tag{1}$$

$$subject\ to\ l \leq x \leq u$$

Here $f: R^n \to R$, $x \in R^n$, $l$ is a size n vector with entries in $R \cup -\infty$ and $u$ is a size n vector with entries in $R \cup \{+\infty\}$. APPS can also handle linear constraints.

## *A. Algorithm flow*

The complete algorithm is described in Algorithm 1. Notations used in Algorithm 1 are explained as follows: $D_k = \left\{D_k^{(1)}, D_k^{(2)}, \ldots, D_k^{(p_k)}\right\}$ is the set of search directions at iteration $k$, superscripts denote the direction index, which ranges from 1 to $p_k$ at iteration $k$. $\Delta_k^{(i)}$ the step length along the $ith$ direction. $A_k$ contains the indices of search directions that have an associated trial point in the evaluation queue at the start of iteration $k$, it may be reset or modified in Step 3 or 4. $A_k$ is also called the "active" set.

$q_{max}$ is the max size of the evaluation queue.

**Algorithm 1 Asynchronous parallel pattern search algorithm**

**Initialization**:
Choose initial solution $x_0$
Choose initial step length $\Delta_0$ and step length tolerance $\Delta_{\text{tol}}$ .
Choose initial search directions: $\{\pm e_1, \pm e_2, \dots, \pm e_n\}$.
**Iteration**: For $k = 0,1, \dots$
1: Generate new trial points:
$X_k = \left\{x_k + \Delta_k^{(i)} d_k^{(i)} : 1 \le i \le p_k, i \notin A_k, \text{and } \Delta_k^{(i)} > \Delta_{tol}\right\}$.
Sent all trial points in $X_k$ to the evaluation queue.
Set $A_{k+1} = \left\{i: \Delta_k^{(i)} < \Delta_{\text{tol}}\right\}$.
2: Collect a nonempty set of evaluated points $Y_k$. If $\exists y_k \in Y_k$ such that $y_k$ satisfies the sufficient decrease condition, then go to Step 3; else go to Step 4.
3: The iteration is successful.
Set $x_{k+1} = y_k$.
Choose new search directions $D_{k+1}$.
Set $\Delta_{k+1}^{(i)} = \widehat{\Delta}\ for\ i = 1, \dots, p_{k+1}$ where $\widehat{\Delta}$ is the step length that produced $y_k$.
Reset $A_{k+1} = \emptyset$.
Prune the evaluation queue to $(qmax - p_{k+1})$or fewer entries.
Go to Step 1.
4: The iteration is unsuccessful.
Set $x_{k+1} = x_k$.
Set $D_{k+1} = D_k$.
Let $I_k = \{\text{direction}(y): y \in Y_k \text{ and parent}(y) = x_k\}$ i. e, directions of evaluated trial points whose parent is $x_k$.
Update $A_{k+1} \leftarrow A_{k+1} \backslash I_k$, where $A_{k+1}$ is defined in Step 1.
For $i = 1, \dots, p_{k+1}: if\ i \in I_k$, set $\Delta_{k+1}^{(i)} = 0.5\Delta_k^{(i)}$; else if $i \notin I_k$, set $\Delta_{k+1}^{(i)} = \Delta_k^{(i)}$
If $\Delta_{k+1}^{(i)} < \Delta_{\text{tol}}$ for $i = 1, \dots, p_{k+1}$, terminate. Else, go to Step 1.

During the initialization phase, the user provides the initial trial point $x_0$, step length $\Delta_0$ and search directions $D_0$. The algorithm generates a set of trial points $X_k$ along the set of search directions to begin with. The search directions should satisfy a few conditions in order to guarantee the convergence of the algorithm. Firstly, they should positively span $R_n$, where $n$ is the number of variables and their cosine measure should be uniformly

bounded. Secondly, they should be uniformly bounded. In the original implementation of APPS [5], the search directions are only along axial directions. This choice of search directions satisfies the above two conditions. APPS has a manager-worker paradigm and uses MPI to manage the parallel tasks. There is a single manager processor controlling the optimization flow while worker processors are doing objective function evaluations. At the beginning of every iteration, once the trial points are generated they are sent by the manager processor to the evaluation queue to be evaluated. Trial points are evaluated in parallel by worker processors. In the synchronous PPS method, the algorithm waits for all the trial points to be evaluated. On the other hand, in APPS, the algorithm waits for only a subset of trial points $Y_k$ to be evaluated. If there is a successful trial point among this subset, there is no need to wait for the rest of the trial points to be evaluated thus saving time. The asynchronous behavior of the APPS method makes it more efficient than the synchronous pattern search because no synchronization is needed at the end of each iteration which avoids waste of computing resources in the case of uneven computing power or task load distribution among processors.

A successful trial point is judged based on either the simple decrease condition or sufficient decrease condition. For the simple decrease condition, a successful trial point only needs to give smaller objective function value than the current best point. For the sufficient decrease condition, a successful trial point should be lower than the current best point by a certain margin set by a forcing function $\rho$ [5].

The sufficient decrease condition relaxes the convergence conditions on search directions. Among the subset of evaluated trial points, if there exists one trial point which minimizes the objective function value by satisfying the sufficient decrease condition, the current iteration is successful and this successful trial point is chosen as the starting point for the next iteration. The successful trial point will be used to generate new trial points around it in the next iteration with the same step length with which it was generated. Any or all trial points still in the evaluation queue waiting to be simulated are pruned so that the number of trial points in the evaluation queue is no more than the maximum queue limit. If no evaluated trial point satisfies the sufficient decrease condition, the current iteration is unsuccessful and starting point for the next iteration is unchanged. If the parent of an evaluated unsuccessful trial point is from an earlier iteration, such trial point is discarded.

The step size is reduced by half along directions corresponding to other evaluated and unsuccessful trial points. New trial points along these directions with the reduced step length are generated in the next iteration. No new trial points will be generated along directions for which a trial point is still in the evaluation queue.

The algorithm proceeds in this fashion until when all directions have step length smaller than the step length tolerance $\Delta_{tol}$. Fig. 2 is an illustrative example of APPS for a 2 dimensional case. The number of worker processors is assumed to be three. In the first iteration, we begin with an initial point and generate four trial points along the four axial

directions. Only two of those four points get evaluated in iteration one. Since there is a trial point which provides sufficient decrease of the objective value, it becomes the starting point of iteration 2. In the second iteration, four more points are generated. Unlike the previous iteration, we find that no evaluated trial point decreases the objective function value. Hence, the unsuccessful direction from the current iteration is step reduced and re-evaluated in iteration 3.

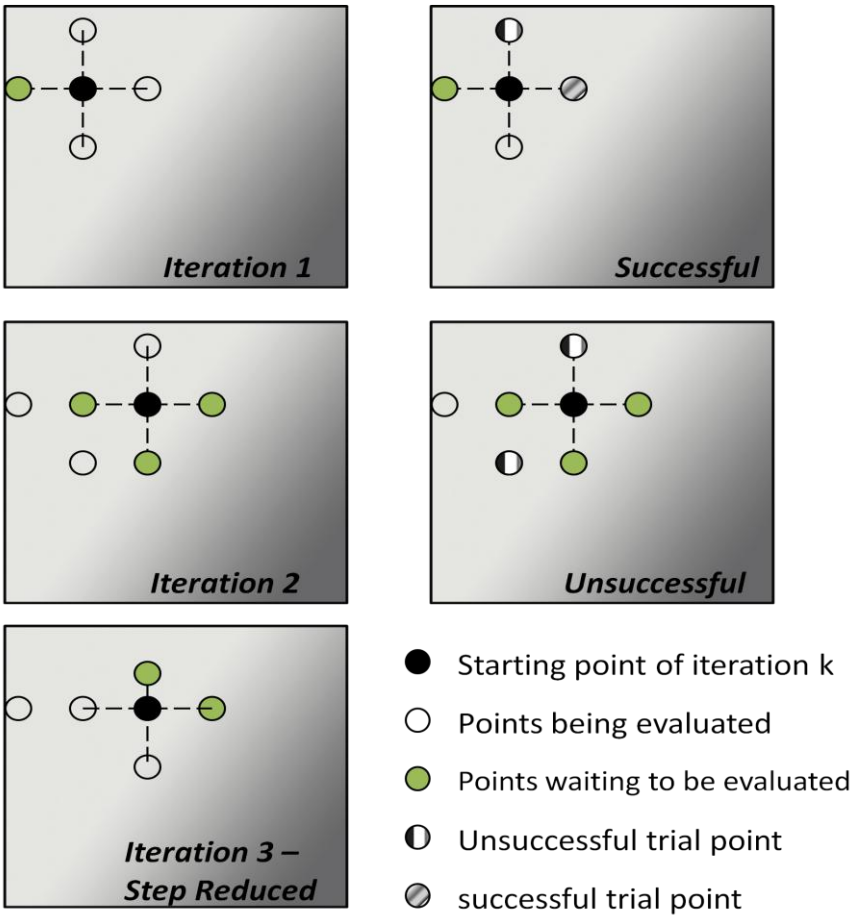


**Figure 2: APPS Illustration**

# CHAPTER III

# QUICK ESTIMATION

This chapter and the next chapter focus mainly on techniques to speed up APPS. While the techniques basically can be applied to any kind of problem for which APPS may be employed, they also make use of some domain specific knowledge. The following two chapters explain the two proposed enhancements to APPS in the context of clock mesh optimization.

For the clock driver sizing problem, since the objective is to minimize clock skew, we define $f(x)$ as a performance metric for clock skew:

$$f(x) = \sum_{j \in S} \left(\mathrm{T_j} - \mu\right)^2 \qquad (2)$$

where $x$ is the vector containing the sizes of all clock drivers, $T_i$ is the clock arrival time at sink node $j$, $S$ is the set contains all sink nodes, $\mu = \left(\sum_{j \in S} T_j\right) / |S|$ is the average of all $Ts$. The purpose of the optimization is to find an optimal set of clock driver sizes to minimize $f(x)$. Since there are only axial search directions in the original APPS method, this means each direction either sizes up or down only one clock driver. Apart from providing the initial clock driver sizes, we also provide an initial step length $\Delta_0$. A large initial step length will result in large change in driver sizes. For the purpose of fine local tuning, it is better to have well controlled initial step size.

*A. Purpose of quick estimation*

In the original asynchronous parallel pattern search (APPS) method [5], once a set of trial points are generated at the beginning of an iteration they are sent out by the master processor to available worker processors for the cost function value evaluation. The sequence of the trial points being sent out to worker processors is random. So the master processor does not control which trial point will be evaluated first and which trial point will be evaluated later. In the clock driver sizing problem, in order to evaluate the objective function $f(x)$ for a trial point $x'$, we have to do an accurate transient simulation for the entire clock mesh using driver sizes in the vector $x'$. The transient simulation of the clock mesh is the most time consuming part in the entire optimization flow. There are some disadvantages with the above process. First of all some trial points or most of the trial points in an iteration cannot satisfy the sufficient decrease condition. Therefore, it is worthless to spend processor resources and time on these "bad" points. Second, for trial points which satisfy the sufficient decrease condition, some of them are better than others. In other words, some trial points may give larger objective function value decrease than others. But the original optimization flow cannot identify those better points since the order in which the trial points are evaluated is random. However, if we can identify a smaller set of good trial points before we commit processor resources to do costly evaluation for all trial points, we can find a successful trial point much faster in an iteration, thereby making the entire optimization flow faster. Identifying the smaller set of good trial points should be done quickly, otherwise this extra step may slow down the entire optimization flow. We propose to use a quick

estimation method for this step. Before we run the accurate simulation, all trial points are going through a quick estimation step. This quick estimation step is like a "virtual evaluation" step in which we estimate the objective function value for all trial points quickly. After the estimated objective function value for all trial points are obtained, we sort them. Trial points with smaller estimated objective function values will be placed before trial points with larger estimated objective function values in the evaluation queue. So, trial points will be sent to available worker processors in the ascending order of the estimated objective function value. In this way, we make sure we always evaluate potentially successful points are given preference over the other trial points, and we always evaluate potentially "better" trial points first. Since the original APPS method only has axial search directions, we need to evaluate large number of trial points quickly and still capture the effect of individual gate change. Since we rank trial points after quick estimation, capturing the relative difference in the objective function value between trial points is important. And after the quick estimation step, we pick the top 10 or 20 with the smallest estimated objective function values instead of only 1 trial points for the accurate evaluation. In this way, despite the fact that the estimated objective function values for those top 10 or 20 points have some error, the chance that the best trial point is among them is very high.

*B. Procedure of quick estimation*

As explained in the beginning of this chapter, the quick estimation is a general technique which can be applied to any kind of optimization problem which can solved using APPS. But what is important is the fact that we need to make use of critical domain specific knowledge to realize it. This section mainly explains the quick estimation techniques which are used for the clock driver sizing problem.

The quick estimation method is similar to the driver merging method and harmonic-weighted model order reduction method proposed in [8]. For fast clock mesh simulation, we want to use model order reduction to reduce the size of the linear mesh. A multi-input multi-output (MIMO) passive interconnect network can be described using the following circuit equations

$$\frac{C\,dx}{dt} + Gx = Bu,\ y = L^T x \tag{3}$$

where $G, C \in R^{nxn}$ describe the resistive and energy storage elements in the circuit, $u \in R^m$ is the input vector, $x \in R^n$ is the vector of unknown voltages and currents, and $B, L \in R^{nxm}$ are the input and output matrices, respectively.

*1. Driver merging*

The widely used passive model reduction algorithm PRIMA [9] generates a reduced order model of (3) by computing an ortho-normal basis $V$ of the Krylov subspace

spanned by $colspan\{R, AR, A^2R, \dots\}$, where $A \equiv -G^{-1}C$ and $R \equiv G^{-1}B$, and $A^iR$ is the $ith$ order block transfer function *moment*.

The reduced order model is given by a set of system matrices of a smaller dimension

$$\tilde{G} = V^T GV, \qquad \tilde{C} = V^T CV, \qquad \tilde{B} = V^T B, \qquad \tilde{L} = V^T L$$

where the order of the reduced order model is determined by the column dimension of $V$ denoted as $q$. The bottleneck in the standard model order reduction is the large number of ports of the linear part. Assuming a clock mesh has 50 clock drivers and 20 moments are matched for each driver port, then a reduced order model with size $q = 1000$ will be computed. The factorization cost of such dense model is $O(1000^3)$. The generation and simulation of such a dense reduced order model can be even more time-consuming than simulation of the original clock mesh. This is why we need to aggressively reduce the number of ports of the linear part of the clock mesh by using the driver merging method. After the number of drivers is drastically reduced, we can then apply the harmonic weighted model order reduction [8] to simulate the simplified clock mesh. As a result, two orders of magnitude of speedup and certain level of accuracy are achieved by the quick estimation routine. The driver merging is done by exploiting the locality in the clock mesh. In the driver merging step, the modified driver is retained as is so that the effect of its size change is captured. All the other drivers are merged into less number of super drivers according to their geometric locations on the clock mesh. For example, if 5 drivers are close together, we merge them into one super driver whose size is the sum of all 5 drivers. The geometrical location of this super driver is the *weighted* center location

of those 5 drivers. More specifically, if the sizes of all 5 drivers are the same, the super driver will be placed in their geometric center. If their sizes are unequal, the super driver will be placed closer to larger drivers to reflect their relatively larger influence in the original clock mesh. The driver merging scheme is formulated in (5). In (5), S is the size of a driver; L is the location of a driver, which can be represented by its coordinates in the X-Y coordinate system. Driver j through driver k are merged into a new driver with size $S_{new}$ and location $L_{new}$.

$$S_{new} = \sum_{i=j}^{k} S_i \tag{4}$$

$$L_{new} = \sum_{i=j}^{k} S_i / S_{new} \; L_i \tag{5}$$

This driver merging approach is illustrated in Fig 3. Our objectives are met: first, simulating the simplified clock mesh is much faster; second, the effect of individual gate change is kept.

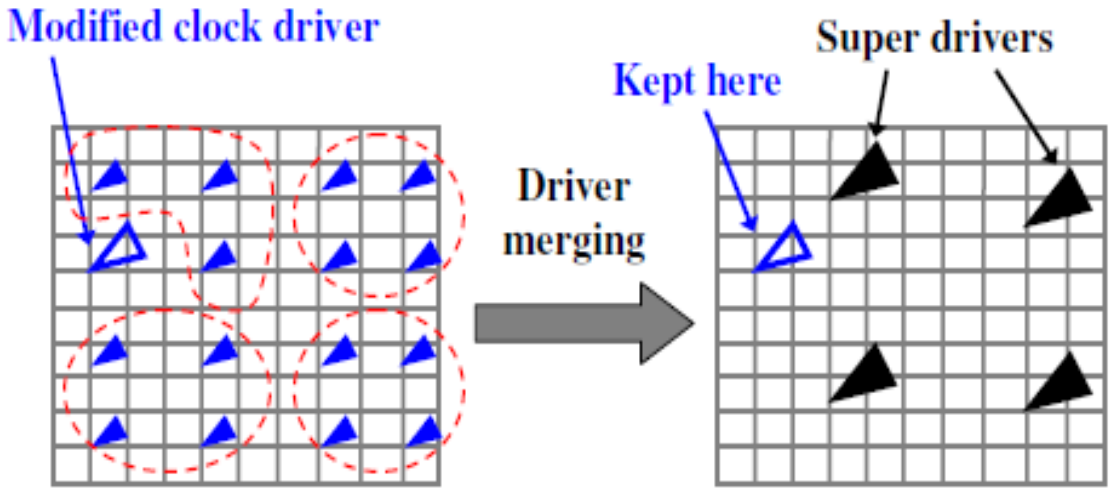


**Figure 3: Driver Merging When Modified Clock Driver Is Retained**

Another more aggressive driver merging approach can also be used. In this approach, there will be only one merging scheme for one clock mesh no matter which driver is modified. This approach is illustrated in Fig. 4. The effect of individual gate change can still be kept. For example, if two adjacent drivers are modified in two trial points respectively, since their sizes are different, the location of super driver into which these two drivers are merged will be different in these two cases. So the relative difference between trial points is captured. In the driver merging, there is a tradeoff between the speedup and accuracy. If there are more super drivers in the resulting simplified clock mesh, the accuracy of the estimated objective function value will be better, but the runtime of simulating the simplified clock mesh will be longer. On the other hand, if lesser number of super drivers are kept in the resulting simplified clock mesh, accuracy will become worse and runtime will become shorter.

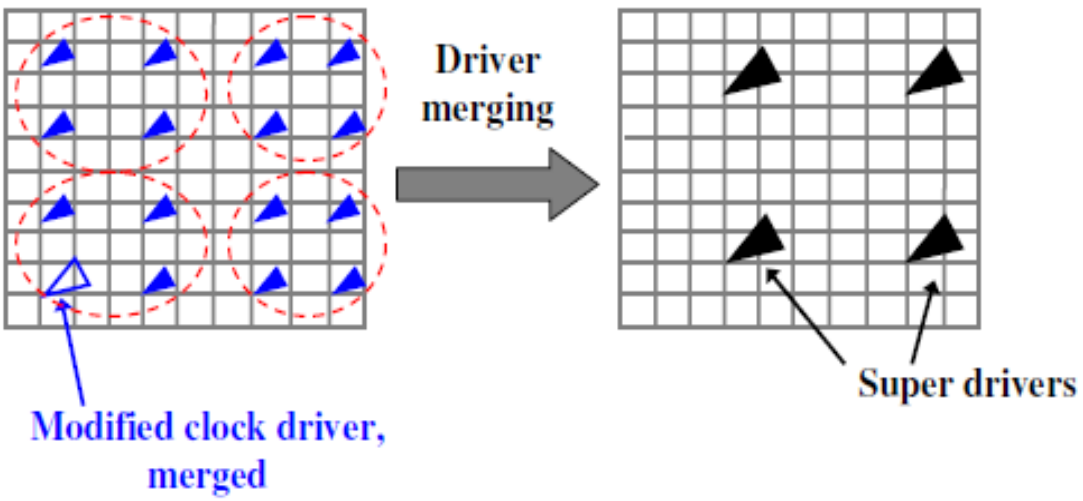


**Figure 4: Driver Merging When Modified Clock Driver Is Not Retained**

*2. Harmonic weighted model order reduction*

Once the number of drivers is drastically reduced by the driver merging method, we can use harmonic weighted model order reduction to simulate the resulting simplified clock mesh. Since the clock signal changes periodically with a known frequency $f_0$ in the clock mesh, a model order reduction technique where the frequency responses at a set of harmonic frequencies are matched would be better than the generic model order reduction technique PRIMA where frequency responses of the network over a continuous frequency range are matched. In the harmonic weighted model order reduction, a multi-point expansion based model order reduction where the transfer functions at each harmonic (corresponding to the expansion point $s = j2\pi kf_0$) are computed and included into the projection matrix V to facilitate projection-based model order reduction. It can be shown that the resulting model will match the system transfer

functions at all these harmonic frequencies considered [10]. Transfer function vectors at these harmonic frequencies can be computed by building SIMO (single input multiple output) based model on a per port basis. Such choice leads to only one LU factorization of the system conductance matrix *G*. Since each harmonic frequency has different impact on the time-domain performance of the clock mesh, we apply weights on transfer function at different frequencies to reflect their relative important. This leads to further reduction of the size of the reduced order model. The entire harmonic weighted model order reduction algorithm is shown in Algorithm 2. The entire quick estimation step is illustrated in Fig. 5. "TFs: port i" in Fig. 5 should be interpreted as contributions from transfer functions at port i instead of the actual transfer functions at port i since there will be weighting and SVD based compression applied on transfer functions. Fig. 6 shows the comparison of waveforms computed by the quick estimation routine and the accurate transient simulation. The starting point of an iteration and a trial point generated from it are evaluated by both quick estimation routine and accurate transient simulation.

The trial point has single clock driver change. We can see that quick estimation routine captures the effect of single driver change very well. More experimental results for the quick estimation method are included in results section.

An important feature of this enhancement is that the quick estimation procedure need not be accurate or physically relevant. For example, the clock mesh drivers are physically not just a single buffer rather each driver is usually realized by a tree of buffers. So, they

are very bulky circuits. In such a case it is physically impossible to merge all these bulky drivers into one super driver. But that is not a cause for concern since for quick estimation the circuit does not have to be physically relevant but should be good enough for a quick approximation.

**Algorithm 2 Harmonic − weighted model order reduction**

**Input**: Full Model: $G, C, B, L$: fundamental frequency $f_0$, Control factor: $\kappa$, Reduced order model size: $S_R$
**Output**: Reduced order model: $\tilde{G}, \tilde{C}, \tilde{B}, \tilde{L}$

1: Compute weight $W_k$ for each harmonic frequency.
2: $V \leftarrow [\,]$.
3: **For** each input $i$ **do**
4: Compute the transfer function at dc: $V_i \leftarrow TF(0, i)$.
5: **For** each harmonic $k, k = 1, \dots, N_h$ **do**
6: Compute the transfer function: $TF(k, i)$.
7: $V_i \leftarrow [V_i, Re\{TF(k, i)\}, Im\{TF(k, i)\}]$.
8: **End for**
9: Normalize each column in $V_i$ and multiply each column using the corresponding weight $W_k$.
10: Perform SVD on the weighted $V_i$ matrix: $V_{i,w} = P_i \Sigma_i Q_i^T$.
11: Keep the first $\kappa$ dominant singular vectors in $P_i$.
V $\leftarrow [V\ [p_i, 1, \dots, p_i, \kappa]]$.
12: **End for**
13: Perform SVD on $V$: $V = P\Sigma Q^T$
14: Keep the first $S_R$ dominant singular vectors $X$ of $P$
$X = \left[p_1, \dots, P_{S_R}\right]$ for model reduction:
$\tilde{G} = X^T G X, \tilde{C} = X^T C X, \tilde{B} = X^T B, \tilde{L} = X^T L$

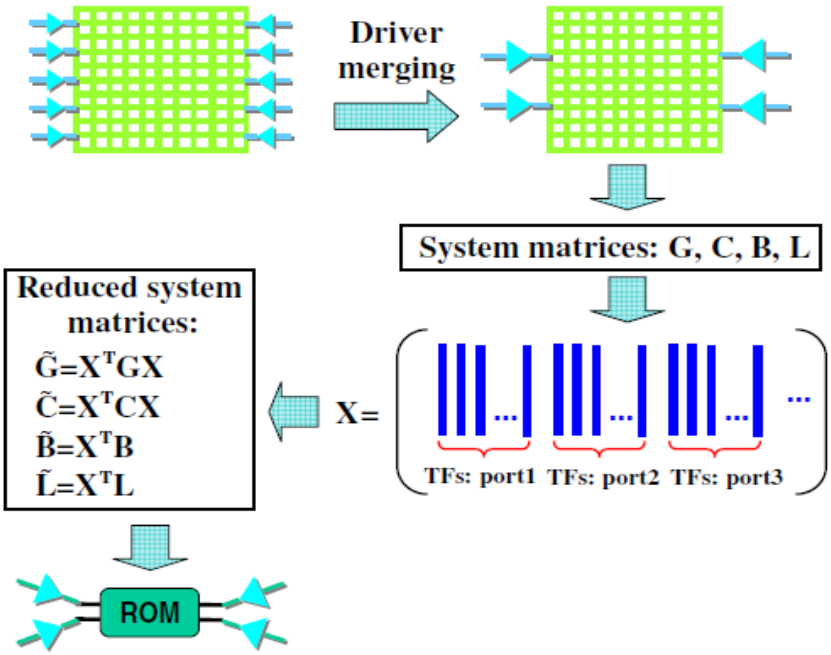


**Figure 5: Quick Estimation Flow**

# CHAPTER IV

# ADDITIONAL SEARCH DIRECTIONS

In the APPS method [6], search directions $D_k$ are the union of two subsets $G_k$ and $H_k$. The subset $G_k$ is the core set of search directions and the subset $H_k$ is a possibly empty set of additional search directions which might accelerate the search. $G_k$ is the key to the convergence analysis and must satisfy Condition 1 for the bound constrained optimization problem defined as

$$\boxed{Condition\ 1. For\ all\ k, G_k = \{\pm e_1, \pm e_2, \dots, \pm e_n\}}$$

The set of additional directions $H_k$ is subject to different convergence conditions under different decrease conditions. If simple decrease condition is used, an additional condition is required to ensure $H_k$ does not interfere with convergence. If sufficient decrease condition is used, the additional condition is not required while Condition 2 is required for the step length of any search directions. Since the implementation of APPS uses sufficient decrease condition, only Condition 2 is required. The additional direction can be a linear combination of any axial directions. And the step length of the additional direction should not exceed $\Delta_k$ at iteration $k$. Condition 2 guarantees that the trial point associated with the additional direction is in the feasible region.

$$
\begin{array}{c}
Condition\ 2. \\
\max\ \tilde{\Delta} \\
subject\ to\ 0 < \check{\Delta} < \Delta_k\ , \\
x_k + \tilde{\Delta} d_k^{(i)} \in \Omega, \\
where\ \Omega\ denotes\ the\ feasible\ region\ defined\ by\ the\ bounds.
\end{array}
$$

Fig. 6 illustrates the benefits of adding additional search directions. The trajectory marked by the solid line only takes axial directions while the trajectory marked by the dashed line takes non-axial directions, the step length is the same for both trajectories. We can see that to reach the same final point, solid line takes 4 steps while dashed lines takes only 3 steps.

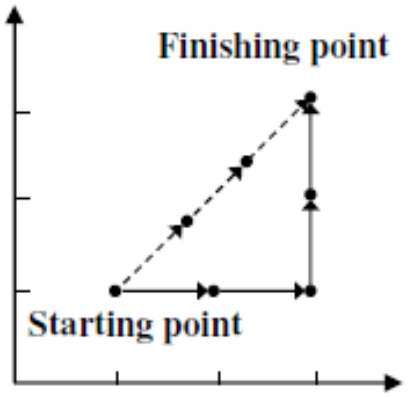


**Figure 6: Advantage of Adding Additional Direction**

*A. Selection of additional directions*

In the modified APPS method, additional search directions are not along axial direction, therefore, their corresponding trial points have multiple drivers change. In the modified APPS method, we select additional directions according to the sensitivity of each driver size with respect to the objective function value. At the beginning of $kth$ iteration, trial points corresponding to $G_k$ (axial directions) are first generated and sent to available worker processors for the quick estimation. In each trial point, there is only one driver size change $\Delta_k$ with respect to the starting point $x_k$ of the current iteration. Since the corresponding objective function value of $x_k$ is available from the last iteration and objective function values for trial points are estimated by the quick estimation routine, the sensitivity of each driver size with respect to the function value can be computed as

$$s_i = \frac{f_{i,estimated} - f(x_k)}{\tilde{\Delta}_k^{(i)} d_k^{(i)}}. \quad (6)$$

In (7), $fi, estimated$ is the objective function value of the $ith$ trial point computed by the quick estimation routine, $f(x_k)$ is the objective function value of the starting point $x_k$ at the $kth$ iteration, $\tilde{\Delta}_k^{(i)} d_k^{(i)}$ is the change in size of the $ith$ driver in the $kth$ iteration. Once the sensitivity for each individual driver is computed, the additional direction is computed as follows: Let $S_{vec} = (\ldots -s_i \ldots, 0, \ldots, -s_j)$ be the size n vector whose entries are either negative of the sensitivity if the driver provides smaller objective function value (either size down or size up), or zero if the driver provides larger objective function value (both size down and size up). The vector of size change associated with the additional direction is

$$h_k^{(l)} = \frac{S_{vec}}{\|S_{vec}\|} \tilde{\Delta} \tag{7}$$

where $\tilde{\Delta}$ is the step length value which satisfies Condition 2. The main benefit of using additional search directions is to reduce the number of iterations and the total runtime.

*B. Flow of modified APPS*

The complete flow of modified APPS method is shown in Fig. 7. Quick estimation is after Step 1 in Algorithm 1. Adding additional direction is after quick estimation since it needs the estimated objective function values to compute sensitivities.

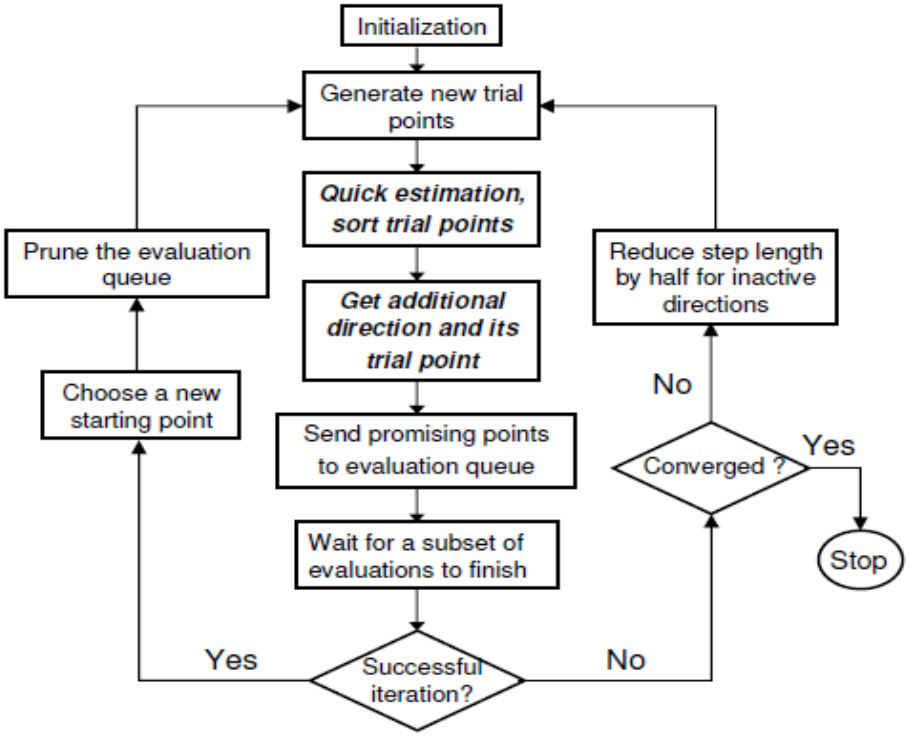


**Figure 7: Modified APPS Algorithm**

# CHAPTER V

# CLOCK MESH OPTIMIZATION RESULTS

In this section, the results of the proposed modified APPS method are shown for the clock driver sizing problem. Since the quick estimation routine is the deciding factor for the speedup of the modified APPS method over the original APPS method, we conduct experiments to verify the accuracy and speedup of the quick estimation routine. The tradeoff between accuracy and speedup is also carefully studied. For the overall optimization, we use a set of clock meshes with different number of clock drivers and circuit elements as test cases. These examples with varying characteristics and sizes allow us to understand how the modified APPS method works for a wide range of problems. We also run the original APPS method [5] and the sequential quadratic programming based optimization method called DONLP2 [7] for these example circuits as comparison reference. The clock skew improvement, number of iterations and the runtime for the modified APPS method are compared against the original APPS method. We also record the final objective function value and clock skew, and runtime for DONLP2. Experimental results show that the modified APPS method has on average about 2 times speedup over the original APPS method while DONLP2 only works for very small clock meshes. The driver merging step in the quick estimation routine is implemented using the Perl scripting language. The model order reduction and transient circuit simulation program is implemented in C++. The software package of the original APPS method is freely available. It is based on MPI. We add the quick estimation and

additional directions modifications to the original APPS implementation. All experiments are conducted on a Linux server with 8GB memory and two 2.33GHz quad-core processors. We use 7 processors for the original and modified APPS methods. 1 processor is the manager and 6 other processors are the workers.

*A. Quick estimation*

As mentioned in Chapter III, the quick estimation routine needs to provide a fairly accurate estimation of the objective function value for a trial point in much shorter time compared with the full evaluation. We achieved this purpose by using the driver merging and model order reduction techniques. The results of verifying the quick estimation routine are included in Table 1. We do both the quick estimation and full evaluation for three clock mesh examples. Their corresponding runtimes, speedup of the quick estimation routine, error of the quick estimation in objective function value are included. We can see that for all three clock mesh examples, quick estimation routine achieves good accuracy in objective function value in much shorter time compared with the full simulation. In this way, it helps the modified APPS method to identify potential successful trial points before the full evaluations and provides estimated sensitivities which are needed to decide the additional direction.

There is tradeoff between the accuracy and runtime in the quick estimation routine. In Table 2, we do the quick estimation for the same three clock mesh examples. But we

keep more drivers after the driver merging step. We can see that the runtime of quick estimation is increased while the accuracy becomes better.

**Table 1: Quick estimation results**

| Mesh ID | #drivers | #drivers (after merging) | #linear elements | Runtime | | Speedup | Error% |
|---|---|---|---|---|---|---|---|
| | | | | Full sim(s) | Quick est(s) | | |
| 1 | 15 | 5 | 2370 | 7.37 | 0.95 | 7.76 | 4.75 |
| 2 | 20 | 5 | 16000 | 160.23 | 2.92 | 54.87 | 4.89 |
| 3 | 25 | 5 | 25000 | 292.56 | 3.11 | 94.07 | 10.68 |

**Table 2: Quick estimation results showing trade-off**

| Mesh ID | #drivers | #drivers (after merging) | #linear elements | Runtime | | Speedup | Error% |
|---|---|---|---|---|---|---|---|
| | | | | Full sim(s) | Quick est(s) | | |
| 1 | 15 | 8 | 2370 | 7.37 | 1.93 | 3.82 | 3.17 |
| 2 | 20 | 10 | 16000 | 160.23 | 8.05 | 19.90 | 0.98 |
| 3 | 25 | 13 | 25000 | 292.56 | 19.95 | 14.66 | 4.52 |

*B. Comparison of optimization methods*

In this subsection, we present the results of applying the original APPS method, our modified APPS method and DONLP2 to the clock driver sizing problem. We have 6 different clock mesh examples with varying complexities and clock load distribution. For every clock mesh example, we start the three optimization methods with the same initial clock driver sizes. Original APPS method and modified APPS method use the

same initial step length and stopping criteria. In Table 3, we include the results of applying DONLP2 for the optimization. We run DONLP2 for much longer time than APPS method for every example. DONLP2 only reduces the objective function value for the smallest clock mesh. For all the other larger ones, it does not effectively reduce the objective function value within the time frame. The reason for the poor performance of DONLP2 is that DONLP2 needs to approximate the Hessian matrix of the Lagrangian internally, which requires multiple full simulations of the clock mesh. For the clock driver sizing problem where n is in the range of 20 to 50 and one simulation takes a few minutes at least, approximating the Hessian matrix could take days.

Table 4 summarizes the runtime and the number of iterations spent by the original APPS method and the modified APPS method to reach the same objective function value. For mesh1 and mesh2, the optimization process is carried to the convergence. For the other larger clock mesh examples, we stop the optimization when it reaches a satisfying objective function value and clock skew. This is due to practical considerations. At the later stages of the optimization, the APPS method needs to spend much more time to find a successful trial point than it does in the earlier stages. If the objective function value is already good enough, it would be better to stop the optimization than carrying out the optimization for one or two more days for a small improvement in objective function value. We can see that the modified APPS method gets 2x speedup over the original APPS method on average. Also the modified APPS method uses less number of iterations. The performance improvement is due to the incorporation of the quick

estimation step and additional directions. From this comparison we can see that for this practical optimization problem which is characterized by expensive objective function value evaluation and lack of explicit derivative information, parallel pattern search based methods are much more effective than sequential quadratic programming based method.

In Figs. 8 to 11, we show the relative clock arrival time distribution for a clock mesh before and after the optimization for different loading conditions. Here the relative clock arrival time at each sink node is defined as $T_j - \mu$, where $T_j$ is the actual clock arrival time at node $j$, $\mu$ is the average clock arrival time among all sink nodes. We can see that after the clock driver size optimization, the clock arrival time at sink nodes across the chip become much closer.

**Table 3: APPS and modified APPS results**

| Mesh ID | Driver count | Linear elements | Clock Skew (ps) | | APPS Runtime | | Speedup | APPS Iterations | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Initial | Final | Original | Modified | | Original | Modified |
| 1 | 15 | 2370 | 12.91 | 2.82 | 6 mins | 3 mins | 2 | 48 | 35 |
| 2 | 20 | 16000 | 91.82 | 7.5 | 9 hrs | 8 hrs | 1.125 | 166 | 119 |
| 3 | 25 | 25000 | 100.98 | 21.7 | 25.7 hrs | 11 hrs | 2.34 | 225 | 76 |
| 4 | 25 | 27000 | 159.74 | 59.8 | 10.5 hrs | 5.5 hrs | 1.91 | 84 | 34 |
| 5 | 30 | 30000 | 103.88 | 38.6 | 27.5 hrs | 12.5 hrs | 2.2 | 158 | 62 |
| 6 | 50 | 40000 | 114.97 | 44 | 41 hrs | 20 hrs | 2.05 | 164 | 37 |

**Table 4: DONLP2 results**

| Mesh ID | Driver count | Linear elements | Function Value | | Clock Skew (ps) | | DONLP Runtime | APPS Runtime |
|---|---|---|---|---|---|---|---|---|
| | | | Initial | Final | Initial | Final | | |
| 1 | 15 | 2370 | 1.16e1 | 6.04 | 12.91 | 9.83 | 20 hrs | 3 mins |
| 2 | 20 | 16000 | 8.52e1 | 8.49e2 | 91.82 | 90.78 | 47 hrs | 8 hrs |
| 3 | 25 | 25000 | 7.02e2 | 7.01e2 | 100.98 | 100.95 | 48 hrs | 11 hrs |
| 4 | 25 | 27000 | 1.68e3 | 1.68e3 | 159.74 | 159.72 | 48 hrs | 5.5 hrs |
| 5 | 30 | 30000 | 5.07e2 | 5.07e2 | 103.88 | 104.84 | 58 hrs | 12.5 hrs |
| 6 | 50 | 40000 | 1.07e3 | 1.07e3 | 114.97 | 114.96 | 58 hrs | 20 hrs |

*C. Delay surfaces*

The below surface charts show the delay distribution across the mesh nodes before and after optimization for different kinds of loading patterns. The delay surfaces represent the arrival time of the clock signal at different output nodes of the mesh also taking into account the placement information. Two different loading patterns are considered – a smooth loading distribution in which the load varies uniformly from one side of the mesh to the other (figures 8 and 9) and a random loading distribution (figures 10 and 11).

*1. Smooth Load Distribution*

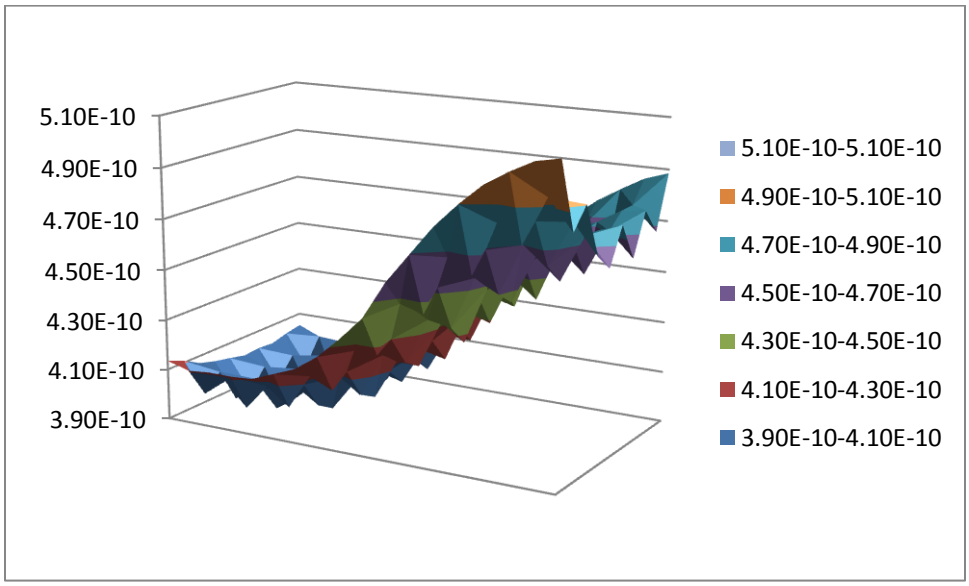


**Figure 8: Clock Arrival Time Distribution before Optimization for Smooth Load Variation**

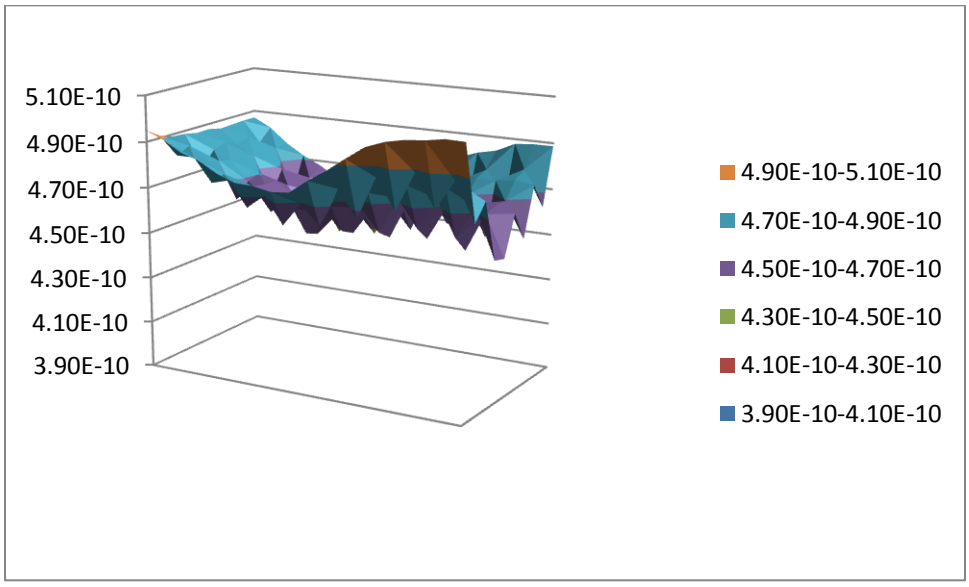


**Figure 9: Clock Arrival Time Distribution after Optimization for Smooth Load Variation**

*2. Non-Uniform Load Distribution*

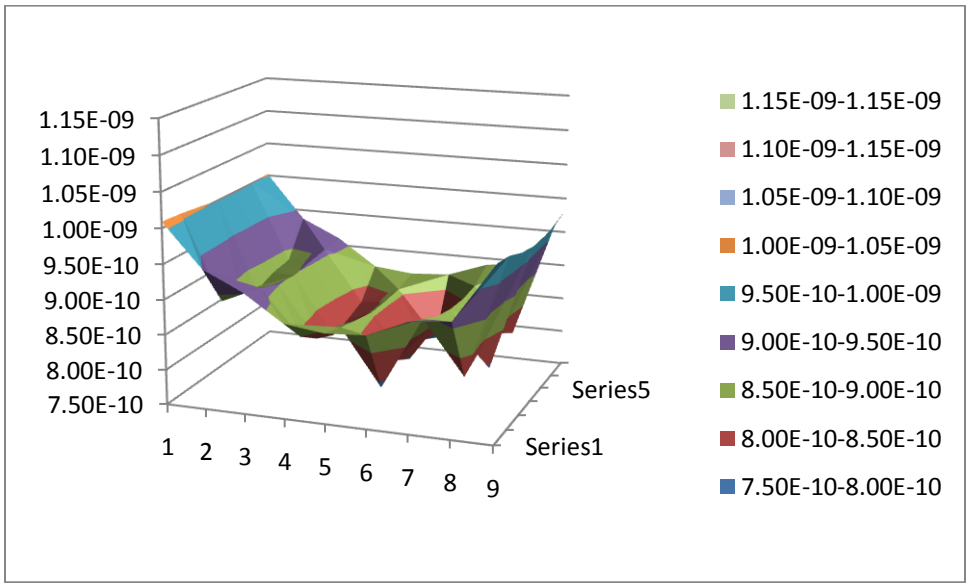


**Figure 10: Clock Arrival Time Distribution before Optimization for Random Load Variation**

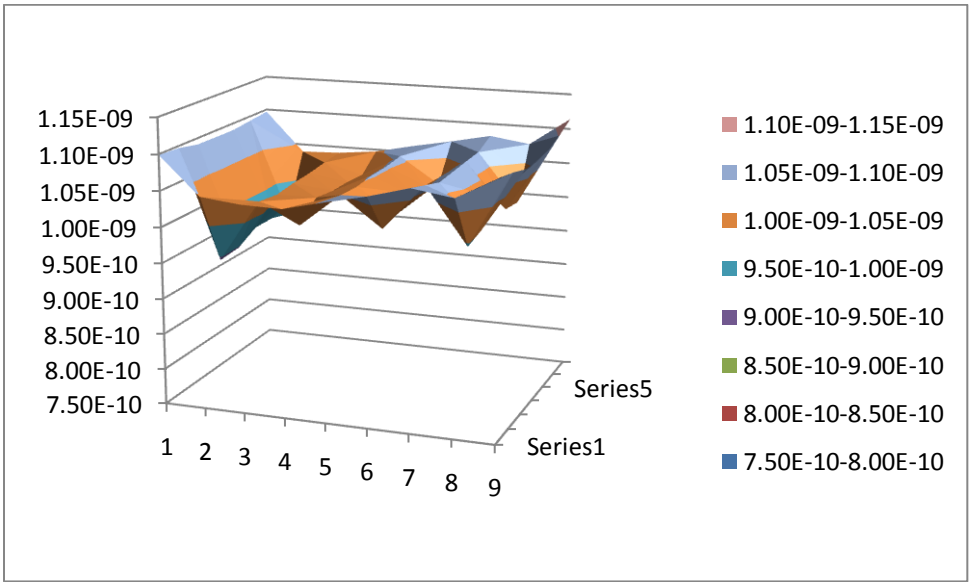


**Figure 11:Clock Arrival Time Distribution after Optimization for Random Load Variation**

# CHAPTER VI

## NON-LINEAR OPTIMIZATION USING APPS

The default APPS algorithm cannot handle non-linear constraints. But many VLSI optimization problems have important trade-offs which appear as non-linear constraints in optimization problems. This chapter and the next few chapters present ways of including non-linear constraints into the APPS method and using it for large VLSI optimization problems specifically PLL system level optimization for locking time reduction.

A few options exist in practice to extend APPS to non-linear optimization. Approaches like the Augmented Lagrangian method or a filter-like method for handling constraints have been proposed based on pattern search and later based on GSS[9]. It works well but is expensive in terms of the number of function evaluations. Augmented Lagrangian methods have many parameters to tune. This work makes use of a penalty based approach that solves a sequence of linearly constrained sub-problems using APPS.

The nonlinear programming problem is defined as follows

$$\min_{x \in R^n} f(x) \tag{8}$$

$$subject\ to\ C_E\ (x) = 0$$

$$C_I(x) \leq 0,$$

$$l\ \leq Ax\ \leq u.$$

Here, $f : R^n \rightarrow R$ is the objective function, , $c : R^n \rightarrow R^m$ includes both the $m_e$ equality and $m_i$ inequality nonlinear constraints with $I \cup E = \{1, \ldots, m = m_e + m_i\}$. The matrix $A \in R^{pxn}$ contains all linear constraints and we require only that $l \leq u$ (permitting equality constraints). Penalty methods transform constrained optimization problems into a sequence of unconstrained (or linearly constrained) sub-problems whose solutions converge to a solution of the original optimization problem. Consequently, (8) is transformed into a linearly constrained problem of the following form:

$$\min_{x \in R^n} \quad f(x) + P(x, \rho_k) \tag{9}$$

$$subject\ to \quad l \leq Ax \leq u.$$

A sequence of such linearly constrained sub-problems is solved with progressively increasing penalty parameter values $\rho$.

The penalty function $P : R^n \rightarrow R$ enforces feasibility in the limit, i.e.,

$$\lim_{\rho \rightarrow \infty} P(x, \rho) = \begin{cases} +\infty & if\ any\ nonlinear\ constraint\ is\ violated \\ 0 & otherwise \end{cases}$$

The parameter $\rho$ is referred to as the penalty parameter and determines the severity of the penalty.

To simplify descriptions of the penalty functions, the following standard transformation to all nonlinear equality constraints is used by defining

$$c_i^+(x) = \begin{cases} c_i(x) & if\ i \in \mathcal{E} \\ \max\{0, c_i(x)\} & if\ i \in \mathcal{I} \end{cases} \tag{10}$$

A commonly used penalty function is based on the squared $l_2$ norm:

$$\mathcal{P}_{l_2^2}(x,\rho) = \rho\|c^+(x)\|_2^2 \tag{11}$$

The $l_2^2$ penalty function has the advantage of being smooth and having "simple" derivatives. More complex penalty functions mean that the relationship between $c(x)$ and the corresponding $\lambda(x)$ would necessarily be nonlinear because the derivatives are no longer "simple"[9].

APPS theoretically requires the existence of derivatives for the convergence theory to apply; however, the specific structure of the derivatives is irrelevant because they are not used explicitly. Still, smoothness is important because non-smooth penalty functions have been shown to cause APPS to converge to a non-differentiable point rather than a KKT point. Unfortunately, a major drawback to the $l_2^2$ penalty function is the uneven way that it penalizes constraints. It places extreme emphasis on constraint violations larger than one and little emphasis on violations less than one. This means that $\rho_k$ has to be very large to enforce asymptotic feasibility. But larger values of $\rho_k$ force APPS to tick-tack down steep constraint valleys using very small steps.

To overcome the above problem, there are other exact penalty functions based on $l_1, l_2,$ and $l_\infty$ for which there exists a finite penalty parameter $\rho$ such that a minimum of (9) coincides with the minimum of (8). A difficulty with exact penalty functions is their inherit non-smoothness. Hence, the APPS may converge to a point of non-differentiability. In order to "fix" the non-smoothness of exact penalty functions, many

authors have proposed smoothed variants based on $l_1, l_2, and\ l_\infty$ norms. The smoothed exact penalty functions are mid-way between the $l_2^2$ penalty function and the exact penalty functions. While the $l_2^2$ penalty function has the advantage of being smooth and simple, exact penalty function converge much faster to an optimal solution but has the disadvantage of being non-smooth. The smoothed exact penalty functions solve the smoothness problem of exact penalty function but converge slower than the exact penalty functions.

# CHAPTER VII

# ALGORITHMIC FRAMEWORK

The basic framework in Algorithm 3 is the same for all kinds of penalty functions. At each iteration, a linearly-constrained sub-problem of the form in (9) is solved. The accuracy requirement of the sub-problem is progressively increases as the iterations progress. Also, the penalty parameter is progressively increased thereby penalizing the constraints more as the iterations progress. The method continues until either the constraint violation is reduced to the desired level and the sub-problem is solved to the desired accuracy, or the budget of function evaluations is exhausted.

The penalty function in Algorithm 3 takes three parameters: $P(x, \rho, \alpha)$. The parameter $\rho$ controls the constraint penalization. The new additional parameter $\alpha$ controls the degree of smoothing for the smoothed exact penalty functions. For penalty functions that do not require it, the parameter can be ignored by initializing $\alpha_0 = 0$. At each iteration, a linearly-constrained sub-problem of the form in (9) is solved using APPS for linearly-constrained problems. As inputs, it takes the solution of the previous solved sub-problem ($x_k$), the penalty-based objective function with $\rho = \rho_k \, and \, \alpha = \alpha_k$, the stopping tolerance ($\delta_k$), and the maximum number of function evaluations allocated for the subproblem ($S_{max}$). The subproblem continues until it converges or exhausts the function evaluations. It returns the best point found $x_{k+1}$; the number of function evaluations used, $S$; and a flag indicating whether or not the sub-problem solver exited successfully,

$state$. The parameters to be used are flexible. But the basic idea behind the algorithm is to initially arrive a reasonable solution for the unconstrained problem and then use that as the starting point to reduce the constraint violation. In other words, the first few iterations are aimed at solely reducing the objective function value while the subsequent iterations are aimed at reducing the constraint violation.

**Algorithm 3 Generic penalty method**

Require: $\rho(\cdot,\cdot,\cdot)$ . Choose penalty function
Require: $x_0$ satisfying $l \leq Ax0 \leq u$ . Initial starting point
Require: $Smax > 0$ . Max evaluations per subproblem
Require: $Tmax \gg Smax$ . Max evaluations overall
Require: $\rho max \gg 1$ . Maximum allowable penalty parameter
Require: $0 < \rho_0 < \rho max$ . Initial value for penalty parameter
Require: $\alpha_0 > 0$ ($\alpha_0 = 0$ if not smoothed) . Initial value for smoothing parameter
Require: $0 < \alpha min < \alpha_0$ . Minimum value for the smoothing parameter
Require: $\delta^* > 0$ . Final subproblem stopping tolerance
Require: $0 < \delta min < \delta^*$ . Minimum subproblem stopping tolerance
Require: $\delta_0 > \delta^*$ . Initial subproblem stopping tolerance
Require: $\eta^* > 0$ . Final constraint tolerance
1: k $\leftarrow$ 0
2: T $\leftarrow$ 0

3: while not converged do
4: $(x_{k+1}, S, \text{state}) \leftarrow$ APPS$(x_k, \mathcal{P}(\cdot, \rho k, \alpha k), \delta k, Smax)$ . Solve subproblem
5: if $\delta$k $< \delta^*$, state is successful, and $\eta(\text{xk}+1) < \eta^*$ then
6: exit (successfully)
7: end if
8: $T \leftarrow T + S$ . Update total number of evaluations
9: if $T > Tmax$ then
10: exit (unsuccessfully)
11: end if
12: if $\eta(xk+1) > max\{\eta^*, \frac{m}{5}\ \alpha k\}$ then
13: $\rho k + 1 \quad min\{2\rho k, \rho max\}$ . Increase penalty parameter
14: end if
15: $\alpha k + 1 \quad max\{\alpha k/2, \alpha min\}$ . Reduce smoothing parameter
16: $\delta k + 1 \quad max\{\delta k/2, \delta min\}$ . Reduce subproblem stopping tolerance
17: $k \leftarrow k + 1$

18: *end while*

An important factor is reducing the overall constraint violation, which is measured in terms of the maximum violation given by

$$\eta(x) = max\{ \mid c_i^+(x) \mid, i = 1, \ldots, m\}. \quad (12)$$

Consequently, $\eta(x)$ plays a role in the convergence of the algorithm. Algorithm 3 is considered to have exited successfully if the following three criteria are satisfied[9]:

1. The sub-problem stopping tolerance is less than the desired final tolerance $\delta^*$. Note that $\delta_k$ is allowed to drop below $\delta^*$ but not below $\delta_{min}$.

2. The sub-problem is solved successfully, meaning that APPS successfully exited with a step length tolerance of $\delta_k \leq \delta^*$

3. The penalty parameter is large enough so that the maximum constraint violation, $\eta(x_{k+1})$, is less than the specified threshold, $\eta^*$.

Note also that the penalty parameter $\rho$ is not increased if $\eta(x_{k+1})$ is sufficiently small.

# CHAPTER VIII

# PHASE LOCKED LOOP OPTIMIZATION

Phase locked loops play an important role in many applications ranging from frequency synthesis to clock recovery in wireless receivers. It is made up of many individual circuit blocks – both analog and digital. As most communication systems and integrated circuits get faster, the performance of the phase locked loop becomes more critical in those applications which makes use of it. But as explained in the introduction, it is not an ordinary task optimizing the performance of phase locked loops. Lack of closed form expression for the objective function means we need to evaluate the same through time consuming simulations. Due to their complexity, the simulation of phase locked loops is extremely time-consuming. To overcome the same, hierarchical techniques have been proposed which make use of behavioral models of the building blocks to quicken the simulation at the system level. The behavioral models are performance based models. The best performance trade-offs of each block are represented using pareto-curves. Hence the goal of the system level optimization is to achieve the best overall system performance along the block level pareto fronts. Mathematically, these pareto fronts can be modeled using non-linear equations. Hence the optimization problem is a non-linear one in both objective function and constraints.

The rest of the chapter is organized as follows. The first sub-section of this chapter explains about the hierarchical optimization framework. Then the basics operation of the

phase locked loop and its modeling is covered followed by the APPS non-linear optimization setup.

*A. Hierarchical optimization*

For large analog circuits with multiple building blocks, hierarchical optimization is a well established approach for optimization [10], [11]. It uses a top-down methodology in which the optimization of a complex analog system is decomposed into that of optimizing several but smaller building blocks. Such an approach alleviates optimization cost and provides significant run time reduction.

One approach to hierarchical analog optimization is to model the best performance trade-offs or pareto fronts of the building blocks beforehand. The pareto fronts represent the best block level performance trade-offs. When there are multiple competing performance measures, one performance measure can only be improved at the cost of the other. Pareto fronts contain those performance/design points such that no single performance can be improved without degrading the other performance parameters. Once we have the pareto-fronts of the block level performance parameters, it is just a matter of doing optimization at the system level within these pareto fronts.

To get the best overall system performance, it is obvious to find the design points which result in best building block performances. Since most building blocks have competing performance objectives, it is impossible to find a design point which gives the best

performance of all objectives. The design task then becomes a multi-objective optimization problem which is to find the best performance trade-offs (pareto fronts). In multi-objective optimization, performance **pa** dominates performance **pb** (supposing smaller value is better) when,

$$pa \prec pb : \forall (pa_i \leq pb_i) \wedge \exists (pa_i < pb_i),\ i = 1, \ldots, n \qquad (13)$$

where $pa_i$ and $pb_i$ are the $i$-th performances of interest, and there are totally $n$ performances. The above relation means that for a design point $pa$ to be dominant to $pb$, all individual performances $pa_i$ should be less than or equal to $pb_i$ and there exists at least one performance measure $pa_i$ which is strictly less than $pb_i$ . A set of performances is considered as pareto-optimal if it is not dominated by any other set of performances. The obtained pareto fronts represent the best performance tradeoffs the circuit blocks can achieve.

The system level optimization is carried out by searching in the space constrained by block-level pareto fronts. There exist two key benefits for this hierarchical optimization. First, since the number of performances in the block level is much smaller than that of the original design space, the search space can be reduced significantly, leading to improved optimization efficiency. Another important benefit is that performance based behavioral models for the building blocks can be used for the system level simulations. Hence the simulation time can be significantly improved thereby reducing the overall optimization cost. Behavioral modeling using hardware description languages (HDL) like verilog-A has been developed for large analog designs. It is then just a matter of

realistically transforming the block level specifications to behavioral level models for the building blocks.

*B. PLL basics and modeling*

The CPPLL architecture is considered as a simple and effective design platform with advantages such as zero phase error and an extended frequency range of operation, and is widely adopted in many PLL systems.

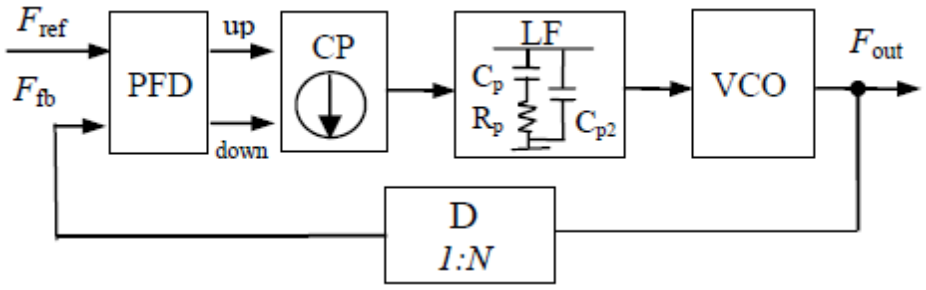


**Figure 12: PLL Block Diagram**

A CPPLL consisting of five building blocks, namely phase frequency detector (PFD), charge pump (CP), loop filter (LF), voltage controlled oscillator (VCO) and divider (D) is shown in Fig. 12. The output frequency can be set to multiples of the reference input frequency by changing the ratio *N* of the divider: *Fout = N·Fref*.

The analog blocks CP, LF and VCO are only considered for the optimization while the digital blocks (PFD&D) are assumed as ideal. The CP shown in figure 13 consists of two current sources: source and sink currents. When the up (down) signal is active, the

source current flows into (out of) the loop filter shown in figure 14, so that the output voltage of the loop filter rises up (drops down), which forces a higher (lower) oscillation frequency. Note that the up and down signals cannot be active at the same time.

The VCO can either be a ring oscillator or an LC tuned oscillator shown in figure 15. In the ring oscillator, input voltage controls the current through the delay elements which determines the delay of each stage and the output oscillation frequency. In an LC oscillator, the input voltage fine tunes the capacitance of a varactor thereby modifying the output resonant frequency of the VCO. An ideal VCO generates a periodic signal whose frequency is a linear function of the controlling voltage. The output frequency $f_{out}$ is given by:

$$f_{out} = f_{min} + K_{VCO} \cdot (Vin - Vmin) \quad (14)$$

$f_{min}$ is the minimum output frequency at the corresponding minimum input voltages *Vmin*. *Vin* is the output controlling voltage of the loop filter.

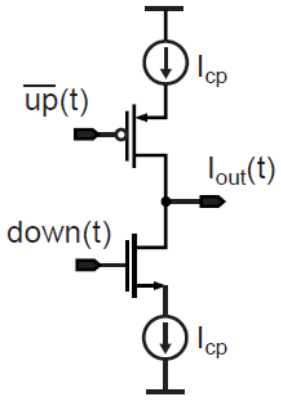


**Figure 13: Charge Pump**

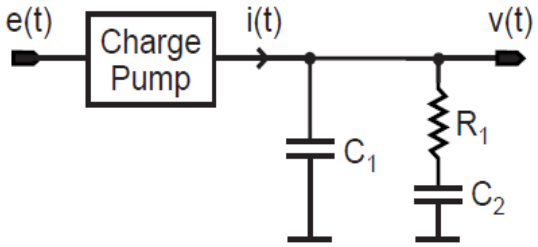


**Figure 14: Loop Filter**

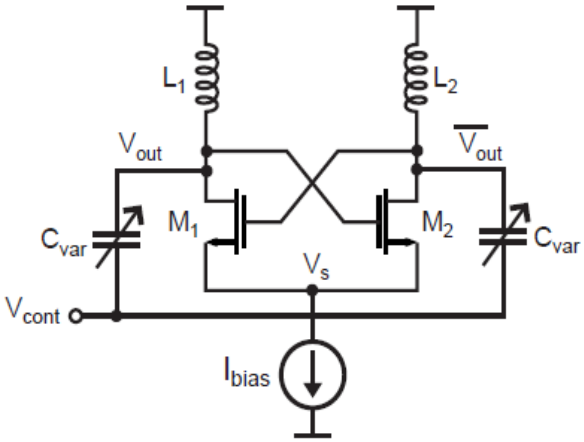


**Figure 15: Voltage Controlled Oscillator**

There are many important performance parameters for the CPPLL system: *locking time*, *jitter*, *power consumption*, *unity gain-bandwidth*, *phase margin* and *output frequency*. In this work we mainly consider the locking time of the PLL system. The locking time is defined as the time taken by the CPPLL to synchronize with or to lock onto a new frequency. In other words, it is the time required for the PLL to go into capture state. The performances shall be considered at the worst case. Therefore the locking time is

defined as the time for the output frequency directly jumping from $f_{min}$ to $f_{max}$ . Jitter or phase noise is the random deviation of the PLL output frequency. Though only the locking time is considered as the objective function, the addition of jitter and power into the objective function is trivial.

At the block level, as mentioned before we use performance based behavioral models to specify the operation. Each building block has its own set of performance parameters which are used to model it at the block level.

For the VCO, three main performance measures are considered: gain, phase noise and power. In addition, the performance trade-offs or the pareto fronts are also modeled before-hand. For the charge pump, the up and down currents are the performance parameters while for the loop filter, they are the filter parameters themselves ($R_p$,$C_{p1}$,$C_{p2}$) .

The pareto fronts represent the optimal performance trade-offs at the block level. For instance as the VCO gain varies inversely with the phase noise, the pareto front captures the best VCO gain for a given phase noise and vice versa. An example of the pareto curves for the PLL building blocks is shown in figure 16. As it can be seen, the pareto surface for the VCO is a 3-D hyper-surface with gain, phase noise and power as the dimensions. . The VCO gain and phase noise are inversely related; and for a given gain, the phase noise decreases as the power increases.

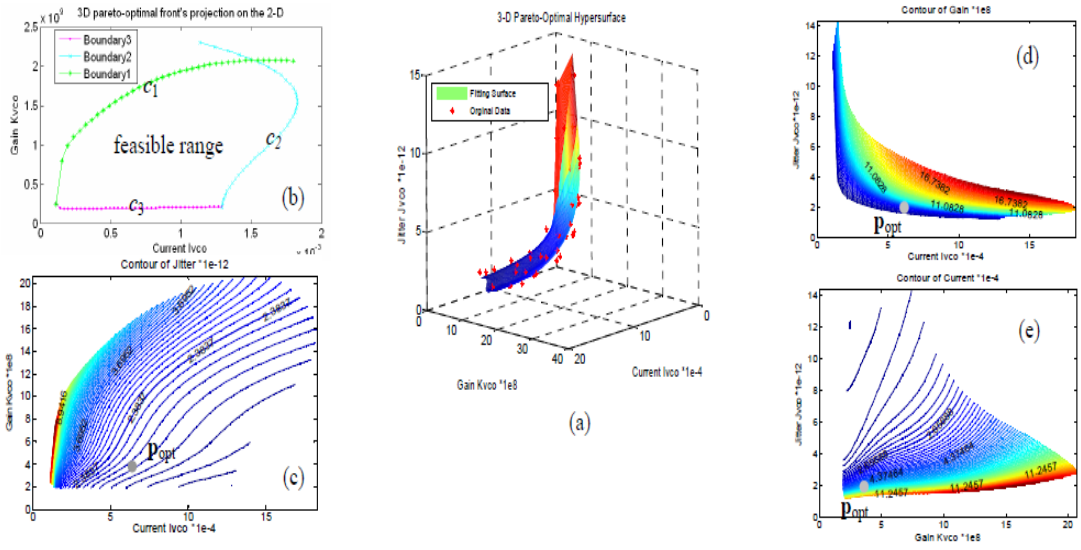


**Figure 16: PLL Block Level Pareto Curves**

*C. PLL optimization setup*

The lock time of the PLL is considered to be the objective function for the system level optimization. The optimization variables are the block level performance measures. The constraints are the block level trade-offs represented as pareto-fronts. For the charge pump, there are only two conflicting performance measures and hence it is modeled as a 2 D curve while for the VCO, the pareto-fronts form a 3 D hyper-surface. The limits of the block level performance measures in the pareto-fronts are the bounds for the optimization variables. The mathematical representation of the optimization problem is given by

$$\min_{x \in R^n} f(x) \tag{15}$$

$$subject\ to\ C_E\ (x) = 0$$

$$l \leq Ax \leq u.$$

Here $f(x)$ represents the PLL locking time as a function of $x$, the block level performance measures. The performance measures considered are VCO gain, power and phase noise, Charge pump up and down currents, and loop filter element values. As explained in the section on hierarchical optimization, the number of optimization variables is significantly reduced if we use performance models for the individual building blocks. The equality constraints $C_E(x)$ represent the non-linear block-level performance trade-offs or pareto-fronts. The main trade-offs considered are those involving the VCO. For example, the VCO gain and phase noise are inversely related; and the relationship also varies with the power. The pareto-fronts are modeled using a regression engine called SVM (support vector machine). The optimization is done using APPS and the non-linear constraints are handled using the modified flow described in Chapter VII.

# CHAPTER IX

# PLL OPTIMIZATION RESULTS

The PLL block level performance measures and the lock time before and after optimization are given in Table 5. The parameters chosen for the non-linear APPS algorithm are given in Table 6.

**Table 5: PLL optimization results**

| | VCO Gain | VCO Noise | VCO Power | CP up(uA) | CP down(uA) | LPF (R,C1,C2) | PLL Locktime (µs) |
|---|---|---|---|---|---|---|---|
| Initial | 1.57e+9 | 1.187e-11 | 7.92e-6 | 1.0985 | 1.1656 | 120k, 3.125p,0.75p | 1.99 |
| Final | 1.9e+9 | 1.069e-11 | 1.01e-5 | 1.0885 | 1.1695 | 188k, 3.125p,0.8p | 0.65 |

The initial scaled constraint violation was 1.27637 for the VCO and the final constraint violation after all the optimization iterations was found to be 0.0092 showing a 99% reduction in constraint violation. In relative terms, the deviation of the VCO gain from the pareto curves reduced from 20.92 % to just 0.14%. Thus the enhanced optimization flow reduces the objective function without violating the constraints. The results also show that the locking time was also reduced from 1.99 micro-seconds to 0.65 micro-seconds, a 67 percent reduction. The result was reached in just three iterations of APPS with increasing penalty parameter values. The final solution also conforms to intuitive reasoning that locking time can be reduced by increasing the loop gain which is achieved by increasing VCO gain.

**Table 6: Non-linear optimization parameters**

| | |
|---|---|
| $\rho(\cdot,\cdot,\cdot)$ penalty function | $l_2^2$ |
| $\rho max$ . Maximum allowable penalty parameter | 1e8 |
| $\rho_0$ . Initial value for penalty parameter | 1 |
| $\delta^* > 0$ . Final subproblem stopping tolerance | 10e-3 |
| $\delta min$ Minimum subproblem stopping tolerance | 1e-6 |
| $\delta_0$ Initial subproblem stopping tolerance | 1e-1 |
| $\eta^*$ Final constraint tolerance | 1e-3 |

The experiments were done using $l_2^2$ penalty functions since they are smooth and guaranteed to converge. Though their penalization is low for small constraint violations, they were preferred over other penalty functions due to their simplicity. Also, it is not clear why we should penalize small constraint violations more. Support vector machine (SVM) was used to model the non-linear constraints representing the pareto fronts. The models were generated prior to simulation and were used to get the difference between the predicted values and optimization variables. It should be noted that for the VCO, SVM is used to model the hyper-surface representing the various performance trade-offs. But if we are modeling two performance parameters as independent variables, there is bound to be a small error since they are not exactly uncorrelated. But this should not cause any problem to the final objective function value as the optimization should bring the variables to within the hyper-surface. One more observation is regarding the speed of the method. Initially it might appear that we need to run the optimization for multiple iterations and run time grows linearly with the number of iterations. But it should be noted that at the end of each APPS iteration the objective function value gets closer and closer to the optimum value. Hence the number of inner loop or the actual APPS iterations keeps progressively reducing as the algorithm proceeds. It is also advisable to

progressively reduce the step size as the penalty parameter is increased. This is to speed up the first few iterations in which we are having a low penalty parameter and the main intention is to get a feasible starting point for the future iterations. But it does not hurt to have the same step length from the beginning with regard to the final function value. This is because, if we have a fine step length in the first iteration itself, the algorithm will spend a long time in reducing the constraint violation right from the start. The run time will increase at the cost of lesser number of APPS iterations.

# CHAPTER X

# CONCLUSION

Thus, a modified asynchronous parallel pattern search for clock mesh skew optimization is presented in this thesis. The proposed method is shown to achieve desirable results in terms of skew reduction and runtime. The method is further extended to be able to incorporate non-linear constraints. The enhanced algorithm is then applied to PLL system level behavioral optimization to reduce the locking time of the system. Desirable results are achieved on that front too.

The future course of work is to incorporate more advanced penalty functions to make the algorithm suitable for any kind of non-linear constraints. Also, generalized speeding up techniques can be developed to make the algorithm more efficient for any kind of VLSI optimization problem.

# REFERENCES

[1] P.J. Restle, T.G. McNamara, D.A. Webber, P.J. Camporese, K.F. Eng, *et al.*, "A clock distribution network for microprocessors," in *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 792–799, May 2001.

[2] R. Heald, K. Aingaran, C. Amir, M. Ang, M. Boland, *et al.*, "Implementation of a 3rd-generation SPARC V9 64b microprocessor," in *Proceeding of the ISSCC Digest of Technical Papers*, San Francisco, California, Feb. 2000, pp. 412-413.

[3] G. Venkataraman, Z. Feng, J. Hu, and P. Li, "Combinatorial algorithms for fast clock mesh optimization," in *Proc.IEEE/ACM Intl. Conf. on CAD*, San Jose, California, November 2006, pp. 563 – 567.

[4] T. G. Kolda, "Revisiting asynchronous parallel pattern search for nonlinear optimization," *SIAM Journal of Optimization.*, vol. 16, no. 2, pp. 563–586, 2005.

[5] G. A. Gray and T. G. Kolda, "Algorithm 856: Appspack 4.0: Asynchronous parallel pattern search for derivative-free optimization," *ACM Trans. Math. Softw.*, vol. 32, no. 3, pp. 485–507, 2006.

[6] X. Ye, P. Li, M. Zhao, R. Panda, and J. Hu, "Analysis of large clock meshes via harmonic-weighted model order reduction and port sliding," in *Proc. IEEE/ACM Intl. Conf. on CAD*, San Jose, California, November 2007, pp. 627-631.

[7] A. Ruhe, "The rational Krylov algorithm for nonsymmetric eigenvalue problems. III: Complex shifts for real matrices," *BIT Numerical Mathematics*, vol. 34, pp. 165–176, 1994.

[8] P. Spellucci, "An SQP method for general nonlinear programs using only equality constrained subproblems," *Mathematical Programming*, vol. 82, pp. 413–448, 1993.

[9] J.D. Griffin and T.G. Kolda, "Nonlinearly-constrained optimization using asynchronous parallel generating set search," *SAND 2007-3257*, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, May 2007.

[10] G. Yu and P. Li, "Yield-aware hierarchical optimization of large analog integrated circuits," in *Proc*. of *IEEE/ACM Int. Conf. on Computer-Aided Design*, San Jose, California, November 2008, pp. 79-84.

[11] J. Zou, D. Mueller, H. Graeb, and U. Schlichtmann, "A CPPLL hierarchical optimization methodology considering jitter, power and locking time," in *Proc. of IEEE/ACM DAC*, San Francisco, California, July 2006, pp. 19–24.

# VITA

Srinath Sudharshan Narasimhan received his Bachelor of Engineering degree in electronics and communication from College of Engineering Guindy, Anna University in 2005. He entered the Master of Science program at Texas A&M University in August 2007. His research interests include Circuit Design and EDA. Mr. Srinath may be reached at D-33 Bay View Apartments, Kalakshetra Colony, Besant Nagar, Chennai 600090, India. His email is srinath1984@gmail.com.