Efficient Design Space Exploration for Component-Based System Design

Yinghai Lu
Analog Mixed Signal Group
Synopsys Inc, USA

Hai Zhou
Electrical Engineering and Computer Science
Northwestern University, USA

***Abstract*—As the technology scaling down continues to go beyond $22nm$, the increasing transistor density on a single die is leading towards more and more complex systems-on-chip. Designers are faced with the challenge of how to efficiently design such a complicated system with tight time-to-market constraints. Component-based system design and design space exploration are two key techniques to overcoming the challenge. In this paper, we model the design space exploration of a system with difference constraints as a bi-criteria convex cost flow problem and develop an efficient solver for it based on parametric simplex method. Furthermore, considering the high cost of synthesizing the underlying soft IP cores, we propose an online algorithm to incrementally refine the system-level Pareto curves as more component-wise sampling points are added. The experimental results demonstrate the efficiency and effectiveness of the proposed algorithms.**

I. Introduction

As the technology scaling-down continues, billions of transistors will be integrated onto one die, creating more complex systems-on-chip (SoC's). The prohibiting size of the system is creating a huge challenge for traditional CAD flow and tools [1]. Moreover, the short time-to-market window of consumer electronics demands highly efficient design process. To cope with the challenge, design abstraction and component-based design methodology are preferred [2]. Instead of handling detailed gates and transistors, the designers build their system upon functional blocks, i.e. soft IP cores. The resulting digital system will consist of a collection of heterogeneous cores.

In order to achieve the best quality out of the component-based design, we need system level optimization. Due to the increasing complexity of the system and the flexibility of the design specification, unlike the traditional single objective optimization that has been widely studied in CAD, the goal of system level optimization is to achieve an intricate trade-off between performance and power efficiency. Instead of obtaining one power-optimized system under a specified timing constraint, it is more preferable to expose the best trade-off between power and performance in order to make a decision among them. This set of optimal design points is called Pareto set or Pareto front [3]. Design space exploration (DSE) is required to generate the Pareto front for a specific design. The major challenge of DSE is its exponential dependency on the number of underlying components, which has been received lots of studies. Some of these studies target on micro-architecture of the circuits and processors [4], [5]. Other work [6], [7], [8], [9] focuses on how to heuristically explore the solution space, which does not have performance guarantee. Another category of work is based on smart sampling [10], [11]. Among this category, recently, Liu et al. proposed a framework to build the system level approximate ϵ-Pareto curve by efficiently sampling the underlying component curves [12] which guarantee a bounded error. Despite the enlightening effective sampling technique, their system composition function is quite simple, assuming no dependence between the components. Moreover, this category of approaches does not explore the structure of the system. As a result, their approaches may waste effort to generate excessive samples to represent the system level Pareto-front.

Given a system composed of a collection of heterogeneous components, the performance of the entire system depends on the computational power of each components as well as the communication between them, which can generally be modeled as a DAG [13], [14]. The dependencies between the components can be represented by a system of difference constraints (SDCs) [15] on the edges of the DAG. Single objective optimization under this DAG model is a well studied problem [13]. However, the idea of how to generate system level Pareto-front under this model has not been explored. As one alternative, the general framework from [12] combined with single objective optimization techniques such as [13] provides a solution for this problem. However, since their work does not explore the DAG structure of the system, their solution is not optimal in terms of both the accuracy of the Pareto-front and the number of samples needed to represent it.

In this work, we study the problem of design space exploration on component-based system design with difference constraints. We formulate the problem as a bi-criteria convex cost flow. Next, we relate it to the parametric convex cost flow to explore its special network structure and propose a parametric simplex based solver. We show that with the component-wise trade-off curves being represented by piecewise-linear convex functions, which rises naturally from the physics of the devices or from convexification of model generation [3], [16], [17], the system level Pareto-front will also be a piecewise linear convex function with bounded number of breakpoints. Furthermore, considering the high cost of synthesizing the soft IP cores, the component-wise curves may not be fully specified at first. In this case, we propose an online sampling algorithm to iteratively refine the system level Pareto curve based on the prediction/selection framework proposed by [12]. In summary, the contribution of the paper includes:

- We for the first time consider the dependencies between the components in a system and model the design space exploration

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright © 2012 ACM 978-1-4503-1573-9/12/11... $15.00

problem of such an inter-dependent system as a bi-criteria convex cost flow.
- We show that the Pareto-front of such a system can be succinctly represented by a piecewise linear function and can be efficiently generated on-the-fly by parametric convex simplex algorithm.
- We propose an online sampling algorithm to iteratively refine the system level Pareto-front.

The rest of the paper is organized as follows. In Section II, we build the model of system and formulate the design space exploration problem as a bi-criteria convex cost flow. In Section III, we study the optimality condition of the formulated problem and introduce the parametric simplex algorithm to get the system level Pareto-front in an on-the-fly fashion. In Section IV, we propose the online sampling algorithm. Experimental results are given in Section V, followed by the conclusions in Section VI.

II. Problem Formulation

A. System-Level Design Space Exploration

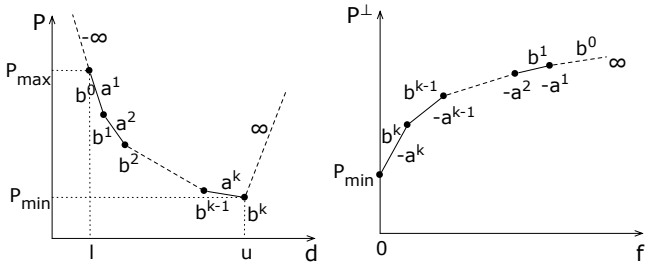
The design of a digital system starts from an application specification, which consists of a set of tasks and the communication among them. Such a specification can usually be modeled as a DAG called Communication Task Graph (CTG) [14], [18]. Figure 1(a) gives an example of a CTG. After mapping, each task is associated with a computational component, e.g. an IP core, and interconnects are implemented between the cores. For each component, there are generally trade-offs between different objectives. Throughout this work, we use the power-latency trade-off as a case study. The goal of system level design space exploration is to generate the power-performance Pareto-front for the whole system with a given the set of trade-off curves of each component. In this paper, we assume that each task is mapped to a different component. Our methods can be extended to handle shared resource by binding tasks to one core and assigning the maximum power among the tasks to that component.

We start by transforming the CTG into another graph $G = (V, E)$, following a similar procedure as in [19], [13]. We split each node in CTG into two nodes which present the input and output of a component. An edge is established from the input node to the output node of the component. Its cost denotes the power-latency trade-off of the component. Inter-component edges are preserved to present the communication. The cost of an intra- or inter-component edge (i, j) can generally be represented by a convex piecewise linear function $P_{ij}(d_{ij})$ with k_{ij} segments [3], [16], [17]. Let the slope of k-th interval (b_{ij}^k, b_{ij}^{k+1}) be $a_{ij}^{k+1} = \frac{P_{ij}(b_{ij}^{k+1}) - P_{ij}(b_{ij}^k)}{b_{ij}^{k+1} - b_{ij}^k}$, while $b_{ij}^0 = l_{ij}$ and $b_{ij}^{k_{ij}} = u_{ij}$ are defined as the lower and upper bounds of the latency on (i, j). The convexity of the function suggests that $a_{ij}^k < a_{ij}^{k+1} < 0$. To extend the range of P_{ij} to $[0, \infty)$, we add two more breakpoints $b_{ij}^{-1} = 0$, $b_{ij}^{k_{ij}+1} = \infty$ and let $a_{ij}^0 = -\infty$, $a_{ij}^{k_{ij}+1} = \infty$. Such an extension enforces that optimal solutions can only be found in $[l_{ij}, u_{ij}]$. Figure 2(a) shows a typical power-latency curve for a component. Given d_{ij}, the power cost on (i, j) can be evaluated as

$$P_{ij}(d_{ij}) = P_{ij}(l_{ij}) + \sum_{k=0}^{q_{ij}-1} a_{ij}^{k+1}(b_{ij}^{k+1} - b_{ij}^k) + a_{ij}^{q_{ij}}(d_{ij} - b_{ij}^{q_{ij}})$$

where q_{ij} is the active segment of P_{ij} such that $b_{ij}^{q_{ij}-1} \le d_{ij} < b_{ij}^{q_{ij}}$.

Next, we add two anchor nodes, s and t, with s connected to all the primary inputs and all the primary outputs connected to t. These I/O edges are artificially added and are of cost zero. Finally, we connect (t, s) to assert parametric latency constraint ϕ of the whole system. Figure 1(b) gives an example of the converted graph of the CTG in Figure 1(a) where dashed edges are auxiliary edges added for problem transformation. The system level design space exploration problem can be formulated as follows:



(a) $P_{ij}(d_{ij})$ (b) $P_{ij}^{\perp}(f_{ij})$

Fig. 2. Examples of $P_{ij}(d_{ij})$ and its conjugate function $P_{ij}^{\perp}(f_{ij})$

Problem 1 (System Level Design Space Exploration):

$$\mathcal{P}(\phi) = \min \sum_{(i,j)\in E} P_{ij}(d_{ij}) \tag{1}$$

subject to

$$t_i + d_{ij} \le t_j, \quad \forall (i,j) \in E/\{(t,s)\} \tag{2}$$

$$d_{ij} \ge 0, \quad \forall (i,j) \in E/\{(t,s)\} \tag{3}$$

$$t_t - \phi \le t_s, \tag{4}$$

$$\phi_{min} \le \phi \le \phi_{max} \tag{5}$$

where t_i is the arrival time on node i.
To generate a continuous system Pareto-front $\mathcal{P}(\phi)$, objective function (1) needs to be optimized for each ϕ in $[\phi_{min}, \phi_{max}]$.

B. Transformation to Bi-Criteria Convex Network Flow

We notice that Problem 1 is a convex dual network problem plus a parametric constraint (5). Applying standard piecewise linear network programming theory [20], we obtain the primal of Problem 1 as:

$$\max \sum_{(i,j)\in E/\{(t,s)\}} P_{ij}^{\perp}(f_{ij}) - \phi f_{ts}, \tag{6}$$

subject to (5) and

$$\sum_{(i,j)\in E} f_{ij} - \sum_{(j,i)\in E} f_{ji} = 0, \qquad \forall i \in V \tag{7}$$

$$f_{ij} \ge 0, \quad \forall (i,j) \in E \tag{8}$$

where f_{ij} is the introduced dual variable which can be also viewed as the flow on edge (i, j), $P_{ij}^{\perp}$ is the conjugate function of P_{ij} [20] which has slope b_{ij}^k on interval $(-a_{ij}^{k+1}, -a_{ij}^k)$. Figure 2(b) shows the conjugate function of the power-latency curve in Figure 2(a). For the sake of clarity, we define

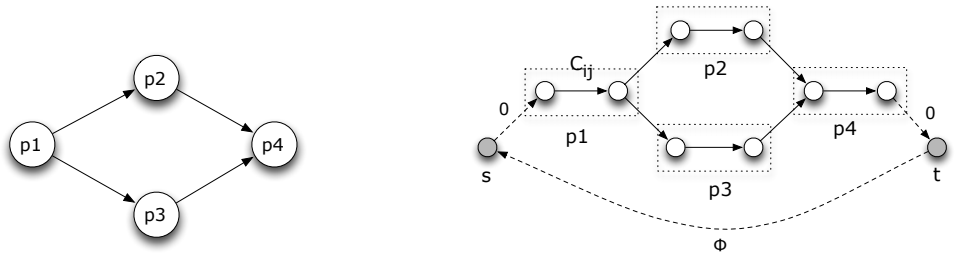
$$\bar{a}_{ij}^k = \begin{cases} 0 & k = 0 \\ -a_{ij}^{k_{ij}-k+1} & k = 1, \ldots, k_{ij} \\ \infty & k = k_{ij}+1 \end{cases}, \tag{9}$$

and

$$\bar{b}_{ij} = b_{ij}^{k_{ij}-k+1}, \quad k = 1, \ldots, k_{ij}+1. \tag{10}$$

Now $P_{ij}^{\perp}(f_{ij})$ can be evaluated as

$$P_{ij}^{\perp}(f_{ij}) = P_{ij}(u_{ij}) + \sum_{k=0}^{q_{ij}-1} \bar{b}_{ij}^{k+1}(\bar{a}_{ij}^{k+1} - \bar{a}_{ij}^k) + \bar{b}_{ij}^{q_{ij}}(f_{ij} - \bar{a}_{ij}^{q_{ij}})$$



(a) CTG (b) Converted Graph G

Fig. 1. Examples of CTG and Converted Graph

where $\bar{a}_{ij}^{q^{ij}-1} \leq f_{ij} < \bar{a}_{ij}^{q^{ij}}$.

Let $C_{ij}(f_{ij}) = -P_{ij}^{\perp}(f_{ij})$. It is easy to see that C_{ij} is a convex piecewise linear function. The system level design space exploration problem can therefore be transformed to a convex flow circulation problem:

Problem 2 (Bi-Criteria Convex Flow Circulation):

$$-\mathcal{P}(\phi) = \min \sum_{(i,j)\in E/\{(t,s)\}} C_{ij}(f_{ij}) + \phi f_{ts}, \tag{11}$$

subject to (5), (7) and (8).

Note that ϕ on (t,s) gives the latency cost of the system while C_{ij} on other edges gives the power cost of the system. Therefore, Problem 2 can be regarded as a bi-criteria optimization [21]. On the other hand, it can be proved that with ϕ fixed, formulation of Problem 2 is equivalent to that of timing-constrained single-objective optimization problem [13], [22] and they share the same optimal solutions.

C. Incremental Design Space Exploration with Online Refinement

In Problem 1 and 2, we assume that the convex piecewise linear Pareto-curve P_{ij} for each component are characterized offline using the method proposed in [12]. But as is pointed out by [12], generating the P_{ij}'s involves heavy calls to synthesizer, which could take days to finish. Although parallel computing may help to save time, it is still desirable to build the system level Pareto-front incrementally by online refinement of the component-wise piecewise linear curves. For each component (i,j), a synthesis on (i,j) will yield a refined P_{ij} curve and will in turn improve the system level Pareto-front $\mathcal{P}$. With K synthesis opportunities, we state the online system level design space exploration problem as follows.

Problem 3 (Online Design Space Exploration): Given K opportunities to synthesize any of the components in the system, distributing the K opportunities among the components so that the system level Pareto-front $\mathcal{P}$ is mostly improved.

III. Parametric Convex Network Simplex Algorithm

In this section, we leverage the parametric network simplex algorithm [23], [24] to obtain system Pareto-front $\mathcal{P}(\phi)$ in an incremental fashion by sliding ϕ on (t,s). We also show that the proposed algorithm capture the piecewise linear nature of system Pareto-front and generates the most succinct representation of $\mathcal{P}(\phi)$.

A. Optimality Condition

Before actually solving Problem 2, we introduce its twin problem by removing (t,s) and injecting flows from s to t, which can be formally described as follows:

Problem 4 (Parameterized Convex Flow Transshipment):

$$\mathcal{P}'(\lambda) = \min \sum_{(i,j)\in E'} C_{ij}(f_{ij}) \tag{12}$$

subject to (8) and

$$\sum_{(i,j)\in E'} f_{ij} - \sum_{(j,i)\in E'} f_{ji} = \begin{cases} -\lambda, & i = s \\ \lambda, & i = t \\ 0, & \text{Otherwise} \end{cases} \tag{13}$$

where graph $G' = (V, E')$ is the same as G except for (t,s), i.e. $E' = E/\{t,s\}$ and λ is the amount of injected flow.

Due to the similar structure of Problem 2 and 4, we show that the parametric convex simplex algorithm can be employed to solve Problem 2, using Problem 4 as a driver.

Now let us look at the optimality condition for both problems. The network simplex method maintains a solution structure for a given graph $G^* = (V^*, E^*)$. It partitions the edges of G^* into basis $\mathcal{T}$ and nonbasis $\mathcal{U} = E^*/\mathcal{T}$, where $\mathcal{T}$ forms a spanning tree of G^* with the property that the flow on any of the edge $(i,j) \in \mathcal{T}$ resides within a certain segment q on $C_{ij}(f_{ij})$, i.e.

$$\bar{a}_{ij}^{q-1} \leq f_{ij} \leq \bar{a}_{ij}^{q}, \quad \forall (i,j) \in \mathcal{T}^q. \tag{14}$$

On the other hand, an edge $(i,j) \in \mathcal{U}$ has the property that the flow is saturated at some breakpoint of $C_{ij}(f_{ij})$, i.e.

$$f_{ij} = \bar{a}_{ij}^{q}, \quad \forall (i,j) \in \mathcal{U}^q. \tag{15}$$

Note that the basis and nonbasis can thus be written as

$$\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \dots \quad \text{and} \quad \mathcal{U} = \mathcal{U}^0 \cup \mathcal{U}^1 \cup \dots$$

We define an optimal solution structure as a pair $(\mathcal{T},\mathcal{U})$ which yields the optimal solution for Problem 2 or 4. The optimal structure of Problem 4 should satisfy the following conditions [24]:

$$\pi_j - \pi_i = \bar{b}_{ij}^{q}, \quad \forall (i,j) \in \mathcal{T}^q, \quad q = 1,2,\dots \tag{16}$$

$$\bar{b}_{ij}^{q} \leq \pi_j - \pi_i \leq \bar{b}_{ij}^{q+1}, \quad \forall (i,j) \in \mathcal{U}^q, \quad q = 0,1,\dots \tag{17}$$

where $\pi_i = -t_i$ is the potential on node $i \in V$. Similarly, the optimal structure of Problem 2 should satisfy not only (16) and (17), but also

$$\pi_s - \pi_t = \phi \tag{18}$$

The above condition ensures that $(t,s) \in \mathcal{T}$ for G because (t,s) has a linear cost function defined over $[0,\infty)$ and can never be saturated at any breakpoint. Intuitively, $\pi_s - \pi_t < \phi$ indicates that there is still timing slack on some components which can be harvested to reduce power.

Another key observation is that although the optimal solution changes as λ or ϕ changes, it is possible that the underlying optimal structure remains the same. We define an interval of injected flow $(\underline{\lambda}, \bar{\lambda})$ during which $\mathcal{T}'$ remains unchanged as a characteristic interval of $(\mathcal{T}',\mathcal{U}')$ for Problem 4. Similarly, we define an interval of the system latency $(\underline{\phi}, \bar{\phi})$ during which $\mathcal{T}$ remains unchanged as a characteristic interval of $(\mathcal{T},\mathcal{U})$ for Problem 2.

B. Algorithm Description

The parametric convex simplex algorithm solves Problem 4 by starting from an initial optimal solution at $\lambda = 0$. Then, it finds the next characteristic interval $(\underline{\lambda}, \bar{\lambda})$ of $\mathcal{T}'$ and increases the injection $\bar{\lambda}$. Next, a dual simplex pivoting is performed to update the optimal basis. At the same, this process actually generates the characteristic interval $(\underline{\phi}, \bar{\phi})$ of $\mathcal{T}$ for Problem 2, which we are interested in. We repeat the above processes until $-\mathcal{P}(\phi)$ is obtained within $[\phi_{min}, \phi_{max}]$.

1) Initialization: We obtain the optimal solution at $\lambda = 0$ by initializing $f_{ij} = 0$ for $(i,j) \in E$. Since all the flow rests in the first segment of C_{ij}, we use its slope $\bar{b}^1_{ij}$ as the cost on edge, set $\pi_s = 0$ and call shortest path routine to generate the shortest path tree as well as the potential for each node on G'. Since G' is acyclic, shortest path on G' is well defined. The resulting shorting path tree constitutes the initial optimal basis $\mathcal{T}'$ for Problem 4 [24]. On the other hand, we notice that $C_{ij}(0) = P_{ij}(u_{ij})$, which means that all the components are powered at minimum voltage with longest latency. Therefore, we obtain the rightmost Pareto point $\mathcal{P}(\phi)$ of Problem 2 at $\phi = \phi_{max} = -\pi_t$.

2) Finding Characteristic Interval: With given optimal structure $(\mathcal{T}', \mathcal{U}')$ and the associated optimal flow f_{ij} at $\lambda = \underline{\lambda}$, we fix all π_i for $i \in V$ and try to find its characteristic interval $(\underline{\lambda}, \bar{\lambda})$. This can be done by augmenting flows along the edges on $\mathcal{T}'$ until at least one edge is saturated at the next breakpoint of C_{ij}, which violates the optimality condition (16).

Since $\mathcal{T}'$ forms a spanning tree of G', there must be a unique path p from s to t. We define p as the basic path and let $\underline{p}$ and $\bar{p}$ be the sets of backward and forward edges on p. The residue capacity of the edges before saturating on p can be computed as

$$w_{ij} = \begin{cases} \bar{b}^q_{ij} - f_{ij}, & (i,j) \in \bar{p} \cap \mathcal{T}^q \\ f_{ij} - \bar{b}^{q-1}_{ij}, & (i,j) \in \underline{p} \cap \mathcal{T}^q \end{cases}, \tag{19}$$

where q is the index of active segment where f_{ij} locates on C_{ij}. The next breakpoint of the current characteristic interval for $\mathcal{T}'$ can thus be obtained as

$$\bar{\lambda} = \underline{\lambda} + \min_{(i,j) \in p} w_{ij}.$$

3) Dual Simplex Pivoting: With π_i fixed, we can keep injecting flow into s and $\mathcal{T}'$ remains optimal until $\lambda = \bar{\lambda}$. Further increase of flow will violate (16). In this case, we can identify the edge (α, β) on p which blocks the flow, i.e. $w_{\alpha\beta} = \min_{(i,j) \in p} w_{ij}$, and remove it from $\mathcal{T}'$. Next, we choose another edge to enter $\mathcal{T}'$ to maintain its optimality. This process is called a dual simplex pivoting. Notice that (α, β) can reenter $\mathcal{T}'$ as a basis edge in $\mathcal{T}'^{q+1}$ or $\mathcal{T}'^{q-1}$.

The removal of (α, β) parts $\mathcal{T}'$ into two subtrees $\mathcal{T}'_s$ and $\mathcal{T}'_t$. The edges joining $\mathcal{T}'_s$ and $\mathcal{T}'_t$ constitute an $s-t$ cut Q. Let $\underline{Q}$ and $\bar{Q}$ be the sets of backward and forward edges in Q. Furthermore, we define μ as

$$\mu_{ij} = \begin{cases} \pi_i - \pi_j + \bar{b}^{q+1}_{ij}, & (i,j) \in \bar{Q} \cap \mathcal{U}^q \\ \pi_j - \pi_i - \bar{b}^q_{ij}, & (i,j) \in \underline{Q} \cap \mathcal{U}^q \end{cases} \tag{20}$$

It follows the optimality condition (17) that $\mu_{ij} \geq 0$ should be satisfied for all $(i,j) \in Q$. Therefore, we update the node potential as

$$\pi_i = \begin{cases} \pi_i, & i \in \mathcal{T}'_s \\ \pi_i + \min_{(i,j) \in Q} \mu_{ij}, & i \in \mathcal{T}'_t \end{cases} \tag{21}$$

and select edge (l,m) with $\mu_{lm} = \min_{(i,j) \in Q} \mu_{ij}$ into $\mathcal{T}'$. In this way, optimality conditions (16) and (17) are preserved. And we obtain a new optimal structure $\mathcal{T}'$.

4) Obtaining $\mathcal{P}(\phi)$: Now let us turn to Problem 2. During the first stage of dual simplex pivoting, we remove a saturated edge (α, β) from $\mathcal{T}'$ to $\mathcal{U}'$. Now let $\mathcal{T} = \mathcal{T}'$ and $\mathcal{U} = \mathcal{U}'$. It is easy to verify that optimality conditions (16)-(18) are all satisfied and $\mathcal{T} \cup \{(t,s)\}$ constitutes an optimal tree structure for Problem 2. Furthermore, this tree structure remains optimal during the interval $[\underline{\phi}, \bar{\phi} = \underline{\phi} + \min_{(i,j) \in Q} \mu_{ij}]$. In other words, the dual simplex pivoting process for Problem 4 actually computes the characteristic interval $[\underline{\phi}, \bar{\phi}]$ for Problem 2.

To obtain the Pareto-front $\mathcal{P}(\phi)$ during $[\underline{\phi}, \bar{\phi}]$, we observe the objective function in (12). During dual simplex pivoting, all the flows are fixed and $f_{ts} = \bar{\lambda}$. As π_t increases from $\underline{\phi}$ to $\bar{\phi}$, the system latency decreases from $-\bar{\phi}$ to $-\underline{\phi}$ and the Pareto-front changes from $\mathcal{P}(-\bar{\phi})$ to $\mathcal{P}(-\underline{\phi})$ with a slope of $-f_{ts} = -\bar{\lambda}$.

5) Termination Condition: The process of characteristic interval search and dual simplex pivoting is carried out repeatedly, during which $\mathcal{P}(\phi)$ is computed along the way starting from the rightmost point where $\phi = \phi_{max} = -\pi_t|_{\lambda=0}$. To determine when to terminate the algorithm, we observe from (8) that the maximum flow which can be injected to G' is unbounded. If during the computation of characteristic interval we get $\min_{(i,j) \in p} w_{ij} = \infty$, it indicates that all the edges on basic path p are forward ones and the latencies of the components on this path are reduced to their minimum with highest power consumption. In this case, we know that the previous dual simplex operation already yielded the leftmost point for $\mathcal{P}(\phi)$ with $\phi = \phi_{min}$. Therefore, the parametric simplex algorithm can be terminated.

C. Summary

We list the system level design space exploration algorithm based on parametric convex network simplex in Algorithm 1. At each iteration, Algorithm 1 generates a segment of $\mathcal{P}(\phi)$ with slope $-\lambda$ in a left to right fashion. Since λ increases with the number of iteration, it reveals an important property of the system level Pareto-front stated as follows.

Algorithm 1 System Level Design Space Exploration

Input: Mapped CTG, component-wise Pareto curves $P_{ij}(d_{ij})$
Output: System level Pareto-curve $\mathcal{P}(\phi)$

```
Generate G from CTG and C_ij from P_ij
Initialize f_ij = 0, λ = 0, π_s = 0
Initialize T' by shortest path with b̄^1_ij as distance
Initialize φ_max = −π_t, φ = φ_max
loop
    Augment flows from s to t along basic path p
    Set w_min = min_{(i,j)∈p} w_ij
    if w_min = ∞ then
        φ_min = φ
        break
    else
        λ = λ + w_min
    end if
    Perform dual simplex pivoting
    Compute μ_min = min_{(i,j)∈Q} μ_ij
    Generate P(φ) on [φ − μ_min, φ] with slope −λ
    Update node potential on T_t, φ = −π_t
end loop
```

Theorem 1: The system level Pareto-curve of Problem 1 is obtained as $\mathcal{P}(\phi)$, which is a piecewise linear convex function with increasing negative slope.

Our algorithm captures the piecewise linear nature of the system level Pareto-front and uses the exact breakpoints to give a most succinct representation of the curve.

The above algorithm is guaranteed to terminate. Denote λ^* as the flow injected when the algorithm terminates which is bounded by $|\sum a_{ij}^1|$. The following theorem follows the conclusions in [23], [24].

Theorem 2: If all b_{ij}^k and a_{ij}^k are integers, the complexity of Algorithm 1 is $O(|V||E||\sum a_{ij}^1|)$.

Despite the pseudo-polynomial complexity bound, our algorithm runs very efficiently in practice, as does other simplex based algorithms [20]. Moreover, instead of optimizing from scratch, it generates the Pareto-front incrementally from the rightmost corner to the leftmost corner.

IV. Incremental System Level Pareto-Front Update with Component-wise Online Sampling

In this section, we design an iterative online algorithm to solve Problem 3. At each iteration, a new breakpoint is added to the piecewise linear curve of the chosen component. The choices are made based on the proposed estimation technique.

A. Piecewise Refinement of Component-wise Curves

Given the opportunity to synthesize a specified component for once, we have to decide under which latency constraint we want to synthesize this component and predict what impact the refined component-wise curve will have on the whole system level Pareto-front. Since the authors in [12] already addressed how to do component-wise sampling, we briefly review their method and focus on the latter.

Let E_1 denotes the set of intra-component edges in G. For a component which corresponds to an edge $(i,j) \in E_1$, its Pareto-curve is P_{ij}. The sampling method in [12] finds the segment with the largest ratio distance and generates a new point in the middle of this segment. For k-th segment (b_{ij}^k, b_{ij}^{k+1}), its ratio distance is defined as [12]:

$$RD(k) = \max\left\{\frac{P_{ij}(b_{ij}^k)}{P_{ij}(b_{ij}^{k+1})}, \frac{b_{ij}^{k+1}}{b_{ij}^k}\right\}. \tag{22}$$

Suppose l-th segment has the largest ratio distance, i.e. $RD(l) = \max_{k=0}^{k_{ij}} RD(k)$, the new sampling point will be synthesized at $b_{ij}^{l'} = (b_{ij}^l + b_{ij}^{l+1})/2$.

After synthesis, a new breakpoint will be generated on the piecewise linear convex curve of P_{ij}. We denote the new curve as $\hat{P}_{ij}$. In $\hat{P}_{ij}$, the old segment (b_{ij}^l, b_{ij}^{l+1}) is replace by two new segments $(b_{ij}^l, b_{ij}^{l'})$ and $(b_{ij}^{l'}, b_{ij}^{l+1})$ with new slopes $a_{ij}^{l'}$ and $a_{ij}^{(l+1)'}$, as is illustrated in Figure 3. The increase of number of segments of P_{ij} translates naturally to the increase of number of segments of $P_{ij}^{\perp}$. Therefore, for Problem 2, the cost function is changed to $\hat{C}_{ij} = -\hat{P}_{ij}^{\perp}$ and the resulting system level Pareto-front $\mathcal{P}(\phi)$ will also change. A refinement of C_{ij} on (i,j) on $\mathcal{P}(\phi)$ will change not only the number of breakpoints of $\mathcal{P}(\phi)$, but also the slope of some intervals of $\mathcal{P}(\phi)$. We need to refresh the Pareto-front $\hat{\mathcal{P}}(\phi)$ by calling Algorithm 1 again. However, since $\hat{P}_{ij}(b_{ij}^{l'}) \leq P_{ij}(b_{ij}^{l'})$, it is guaranteed that with new breakpoint added to any of the component-wise curves, the system level Pareto curve will either remain unchanged or come down, i.e.

$$\hat{\mathcal{P}}(\phi) \leq \mathcal{P}(\phi), \quad \phi \in [\phi_{min}, \phi_{max}].$$

This effect will be demonstrated later in Section V.

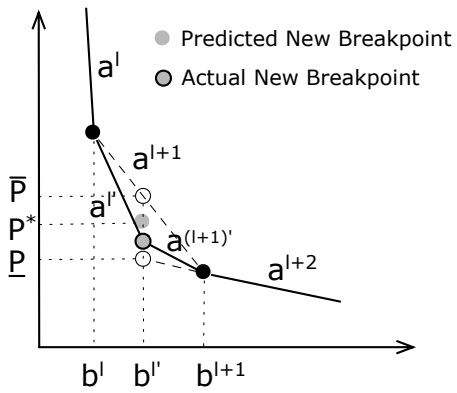


Fig. 3. Refinement of Piecewise Linear Curve P_{ij}

B. Sensitivity Estimation for Online Sampling

Suppose we are given K opportunities to synthesis any of the components of the system, we are facing the problem of which components to refine such that the resulting system level Pareto-front is mostly improved. There are two major difficulties. First, the exact power value of $P_{ij}(d)$ is not known until synthesizer is finished on given latency d. Secondly, even we have the full knowledge of the shape of P_{ij}, it can still be proved that it is NP-hard to optimally choose the K components to synthesize when K is smaller than the number of component by reduction from *Set Cover*.

In this work, we propose a greedy online algorithm with online sampling based on the prediction/selection framework proposed by [12]. The algorithm works iteratively. At each iteration, it generates a new sampling point on each of the components. With the new sampling point $b_{ij}^{l'}$, it predicts the value of $P_{ij}(b_{ij}^{l'})$ using the following criteria. Suppose we choose segment (b_{ij}^l, b_{ij}^{l+1}) for refinement. It follows from the convexity of P_{ij} that the slopes $a_{ij}^{l'}$ and $a_{ij}^{(l+1)'}$ of the newly generated two segments $(b_{ij}^l, b_{ij}^{l'})$ and $(b_{ij}^{l'}, b_{ij}^{l+1})$ satisfy the that:

$$a_{ij}^{l'} \geq a_{ij}^{l+1} \quad \text{and} \quad a_{ij}^{(l+1)'} \leq a_{ij}^{l+2}.$$

It is thus easy to derive that the true value of $P_{ij}(b_{ij}^{l'})$ should be bounded between

$$\bar{P}_{ij}(b_{ij}^{l'}) = P_{ij}(b_{ij}^l) + a_{ij}^{l+1}(b_{ij}^{l'} - b_{ij}^l) \tag{23}$$

and

$$\underline{P}_{ij}(b_{ij}^{l'}) = \max\left\{\begin{array}{l} P_{ij}(b_{ij}^{l+1}) - a_{ij}^{l+2}(b_{ij}^{l+1} - b_{ij}^{l'}), \\ P_{ij}(b_{ij}^l) + a_{ij}^l(b_{ij}^{l'} - b_{ij}^l) \end{array}\right\}, \tag{24}$$

as is demonstrated in Figure 3. We use their average as the prediction of $P_{ij}(b_{ij}^{l'})$:

$$P_{ij}^*(b_{ij}^{l'}) = \frac{\bar{P}_{ij}(b_{ij}^{l'}) + \underline{P}_{ij}(b_{ij}^{l'})}{2} \tag{25}$$

With the predicted value on $P_{ij}(b_{ij}^{l'})$, we update the curve P_{ij} and C_{ij} of component $(i,j) \in E_1$ and call Algorithm 1 to generate a Pareto-front $\mathcal{P}_{ij}(\phi)$, which predicts the impact of refining component (i,j) on the change of the Pareto-front of the whole system. Next, we choose the component (i^*, j^*) which pushes down $\mathcal{P}(\phi)$ the most to synthesize. After actual synthesis on component (i^*, j^*), we commit the change to $P_{i^*j^*}$ and $C_{i^*j^*}$ and update $\mathcal{P}(\phi)$.

The above process is repeated until maximum sampling limit K is reached or the difference between $\mathcal{P}(\phi)$ and $\mathcal{P}_{ij}(\phi)$ drops below

a given threshold ϵ. The refined system level Pareto-front is yielded at the end of the algorithm. The whole flow is listed in Algorithm 2. Since system level Pareto-front needs to be regenerated repeatedly, it

Algorithm 2 Incremental DSE with Online Sampling

Input: maximum sampling number K or accuracy tolerance ϵ

1: Generate the initial set using Algorithm 1
2: **repeat**
3: **for all** $(i, j) \in E_1$ **do**
4: Sample P_{ij} and predict its value using (25)
5: Call Algorithm 1 and obtain $-\mathcal{P}_{ij}(\phi)$
6: **end for**
7: Choose component (i^*, j^*) with largest improvement
8: Sample (i^*, j^*) with actual synthesizer and update P_{ij}
9: Call Algorithm 1 and update $-\mathcal{P}(\phi)$
10: $K = K - 1$
11: **until** $K = 0$ or $RD(\mathcal{P}(\phi)), \mathcal{P}_{ij}(\phi)) < \epsilon$

is important that we have an efficient Algorithm 1 to do this job.

V. Experimental Results

We implemented the proposed algorithms in C++ programming language and all the experiments are carried out on a Linux machine with 2GHz Intel quad-core CPU and 2GB memory. We use the well-received Embedded system Synthesis Benchmark (E3S) [25] as test cases, which includes applications from office automation, consumer, networking, auto industry and telecommunication. The number of tasks in CTGs from E3S benchmarks ranges from 5 to 30 and they are pre-mapped to different processor cores provided by E3S [14]. The power-latency trade-off for each component is characterized using the scaling functions provided by authors from [13] with supply voltage $0.8v$, $1.0v$, $1.2v$, $1.4v$ and $1.6v$. This yields a piecewise function P_{ij} of 4 segments for each component.

First, we compare the efficiency of Algorithm 1 on building the system level Pareto-front. Using the setup described above, we generate the system Pareto-front on the benchmarks by Algorithm 1. For comparison, we implemented the approach in [12] to generate the ϵ-Pareto curve with accuracy $\epsilon = 1\%$ and $\epsilon = 0.01\%$. For the latter case, we are in effect doing exhaustive search for the Pareto-front. Note that since the method proposed in [12] does not explore the structure of Problem 2, we have to call optimizer to generate the optimal power on each of the sampling point on ϕ. The comparison of results is listed in Table I. Since the resulting Pareto-front are piecewise linear connected by breakpoints. We count the number of segments of the curve to compare the representation power of different methods, which is denoted as "#Seg" in Table I. The exact number of segments generated by the proposed algorithm is in linear order to the total number of segments of the curves for underlying components. On average, the proposed method reduced number of segments by 62.2% compared to the ϵ-Pareto method in [12]. When compared to the exhaustive search method, it reduces the number of segments by 99.6%. The proposed algorithm is very efficient, finishing all the cases within 1 second. On top of this, it is important to notice that the proposed algorithm generates exact Pareto-front on system level while ϵ-Pareto generates the curve within an error of ϵ, where it is set as 1% in this experiment. Figure 4 gives the system Pareto-fronts of "Telecomm", generated by the proposed method and ϵ-Pareto method with $\epsilon = 1\%$. Since the curve generated by our algorithm is exact, we omit the curve of exhaustive search method to make the graph clear. We can observe that the general

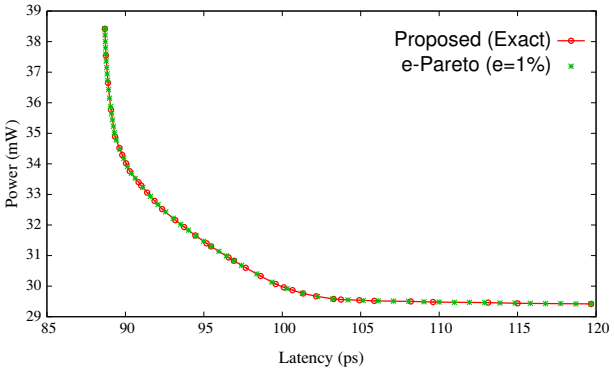


Fig. 4. System Pareto-fronts of Telecomm.

ϵ-Pareto method does not explore the piecewise-linear nature of $\mathcal{P}$ and will generate excessive breakpoints on linear segments, e.g. the first three and the last segments in Figure 4. On the other hand, being an approximation method, it cannot match the exact breakpoints generated by the proposed algorithm, causing errors on the segments between them.

Next, we test the proposed incremental Pareto-front update algorithm with online sampling. We set a linear function with one segment as the initial curve for each component, including only the leftmost and rightmost breakpoints. We then generate an initial system level Pareto-front based on this setup. For comparison, we generate smooth convex curves for each component, which we dubbed as "full" curves. The resulting system Pareto-front is obtained by exhaustive searching and is used as the exact curve. Then we use Algorithm 2 to incrementally generate new breakpoints by sampling from the "full" curves. We set the stop condition as $\epsilon = 0.1\%$. Figure 5 shows the number of iterations that Algorithm 2 takes before reaching stop condition. The resulting system Pareto-fronts, which is called the

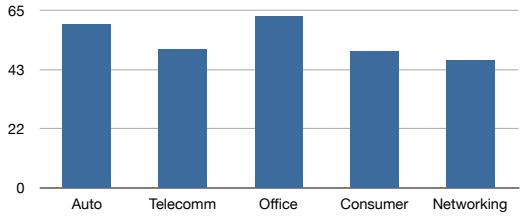


Fig. 5. Number of Refinement Iterations of Algorithm 2 on E3S Bench

refined Pareto-front, for each of the test case are all within the error of 1%, when compared to the exact ones. Figure 6 gives the resulting curves of Algorithm 2 for case Auto Industry. As has been mentioned in Section IV, with refined component-wise curves, our algorithm progressive pushes the system level Pareto-front down while adding necessary breakpoints.

Finally, we use Figure 7 to demonstrate how Algorithm 2 adaptively samples the components based on the proposed prediction technique. Figure 7 shows the distribution of number of segments of the component-wise curves on Auto Industry after refinement. In order to improve the system Pareto-front in a most effective way, the proposed algorithm focused on Components 0, 8 and 13 while on the other hand, components like 1-7 and 14-17 were only refined once.

TABLE I
COMPARISON OF SYSTEM LEVEL PARETO-FRONT GENERATION

Benchmark	#Components	Exhaustive Search ($\epsilon = 0.01\%$)	[12] ($\epsilon = 1\%$)	Proposed Algorithm
		#Seg	#Seg	#Seg
Auto Industry	18	5848	55	57
Telecommunication	15	5998	59	37
Office Automation	5	12844	137	29
Consumer	11	12211	120	37
Networking	6	11728	118	29
Average		9725	98	37

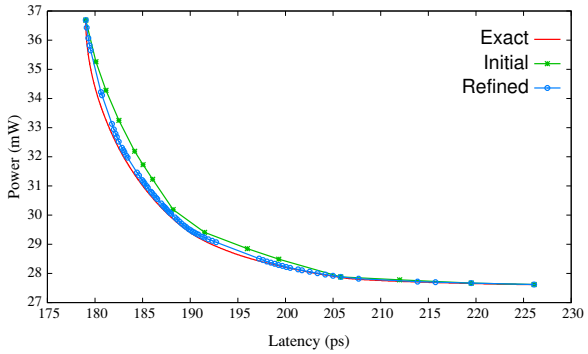


Fig. 6. Refinement of System Pareto-fronts of Auto

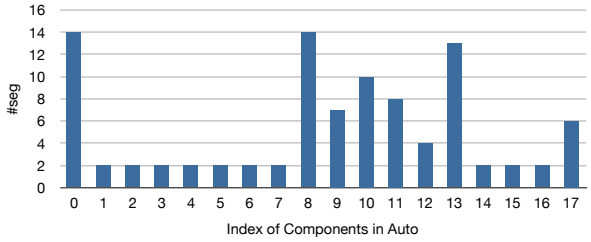


Fig. 7. Statistics of #Segments for Components on Auto after Refinement

VI. CONCLUSIONS

Component-based system design and design space exploration are the two keys to overcoming the challenge of complex SoC design. In this paper, we model the design space exploration of system with difference constraints as a bi-criteria convex cost flow and design an efficient solver for it based on parametric simplex method. In addition, we propose an online algorithm to incrementally refine the system-level Pareto-front with more component-wise sampling points added. The experimental results demonstrate the efficiency and effectiveness of the proposed algorithms.

REFERENCES

[1] ITRS, *The International Technology Roadmap for Semiconductors*, http://public.itrs.net.

[2] S. Borkar, "Design perspectives on 22nm cmos and beyond," in *Proc. of the Design Automation Conf.*, 2009, pp. 93–94.

[3] I. Diakonikolas and M. Yannakakis, "Succinct approximate convex pareto curves," in *ACM-SIAM symposium on Discrete algorithms*, 2008, pp. 74–83.

[4] G. Beltrame, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *IEEE Transactions on Computer Aided Design*, vol. 29, no. 7, pp. 1083–1095, 2010.

[5] U. D. Bordolo, H. P. Huynh, S. Chakraborty, and T. Mitra, "Evaluating design trade-offs in customizable processors," in *Proc. of the Design Automation Conf.*, 2009, pp. 244 –249.

[6] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Transactions on Very Large-Scale Integrated Systems*, vol. 10, no. 4, pp. 416–422, 2002.

[7] B. Schafer and K. Wakabayashi, "Design space exploration acceleration through operation clustering," *IEEE Transactions on Computer Aided Design*, vol. 29, no. 1, pp. 153–157, 2010.

[8] T. Eeckelaert, T. McConaghy, and G. Gielen, "Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces," in *Proc. DATE: Design Automation and Test in Europe*, 2005, pp. 1070–1075.

[9] S. Tiwary, P. K. Tiwary, and R. A. Rutenbar, "Generation of yield-aware pareto surfaces for hierarchical circuit design space exploration," in *Proc. of the Design Automation Conf.*, 2006, pp. 31–36.

[10] G. Palermo, C. Silvano, and V. Zaccaria, "Respir: A response surface-based pareto iterative refinement for application-specific design space exploration," *IEEE Transactions on Computer Aided Design*, vol. 28, no. 12, pp. 1816 –1829, 2009.

[11] A. Singhee and P. Castalino, "Pareto sampling: Choosing the right weights by derivative pursuit," in *Proc. of the Design Automation Conf.*, 2010, pp. 913–916.

[12] H.-Y. Liu, I. Diakonikolas, M. Petracca, and L. Carloni, "Supervised design space exploration by compositional approximation of pareto sets," in *Proc. of the Design Automation Conf.*, 2011, pp. 399–404.

[13] Q. Ma and E. Young, "Network flow-based power optimization under timing constraints in msv-driven floorplanning," in *Proc. Intl. Conf. on Computer-Aided Design*, 2008, pp. 1–8.

[14] Y. Liu, Y. Yang, and J. Hu, "Clustering-based simultaneous task and voltage scheduling for noc systems," in *Proc. Intl. Conf. on Computer-Aided Design*, 2010.

[15] J. Cong and Z. Zhang, "An efficient and versatile scheduling algorithm based on sdc formulation," in *Proc. of the Design Automation Conf.*, 2006, pp. 433–438.

[16] S. Roy and W. Chen, "Convexfit: an optimal minimum-error convex fitting and smoothing algorithm with application to gate-sizing," in *Proc. Intl. Conf. on Computer-Aided Design*, 2005, pp. 196–203.

[17] S. Roy and C.-P. Chen, "Convexsmooth: a simultaneous convex fitting and smoothing algorithm for convex optimization problems," in *Proc. International Symposium Quality Electronic Design*, 2006.

[18] Y. Chen and H. Zhou, "Buffer minimization in pipelined sdf scheduling on multi-core platforms," in *Proc. Asian and South Pacific Design Automation Conference*, 2012, pp. 127–132.

[19] C. Lin, A. Xie, and H. Zhou, "Design closure driven delay relaxation based on convex cost network flow," in *Proc. DATE: Design Automation and Test in Europe*, Nice, France, Apr. 2007.

[20] F. A. Marins, E. L. Senne, K. Darby-Dowman, A. F. Machado, and C. Perin, "Algorithms for network piecewise-linear programs: A comparative study," *European Journal of Operational Research*, vol. 97, no. 1, pp. 183–199, 1997.

[21] H. W. Hamacher, C. R. Pedersen, and S. Ruzika, "Multiple objective minimum cost flow problems: A review," *European Journal of Operational Research*, vol. 176, no. 3, pp. 1404 – 1422, 2007.

[22] L. Li, J. Sun, Y. Lu, H. Zhou, and X. Zeng, "Low power discrete voltage assignment under clock skew scheduling," in *Proc. Asian and South Pacific Design Automation Conference*, 2011, pp. 515–520.

[23] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[24] R. Ahuja, J. Batra, and S. Gupta, "A parametric algorithm for convex cost network flow and related problems," *European Journal of Operational Research*, vol. 16, no. 2, pp. 222–235, 1984.

[25] R. Dick, *Embedded system synthesis benchmarks suites(E3S)*, http://ziyang.eecs.umich.edu/dickrp/e3s/.