Compositional System-Level Design Exploration with Planning of High-Level Synthesis

Hung-Yi Liu, Michele Petracca, and Luca P. Carloni
Department of Computer Science, Columbia University, New York, NY, USA
Email: {hungyi, petracca, luca}@cs.columbia.edu

***Abstract*—The growing complexity of System-on-Chip (SoC) design calls for an increased usage of transaction-level modeling (TLM), high-level synthesis tools, and reuse of pre-designed components. In the framework of a compositional methodology for efficient SoC design exploration we present three main contributions: a concise library format for characterization and reuse of components specified in high-level languages like SystemC; an algorithm to prune alternative implementations of a component given the context of a specific SoC design; and an algorithm that explores compositionally the design space of the SoC and produces a detailed plan to run high-level synthesis on its components for the final implementation. The two algorithms are computationally efficient and enable an effective parallelization of the synthesis runs. Through a case study, we show how our methodology returns the essential properties of the design space at the system level by combining the information from the library of components and by identifying automatically those having the most critical impact on the overall design.**

I. Introduction

The complexity of SoC design continues to grow with the emergence of heterogeneous multi-core architectures that feature a large mix of programmable cores and specialized hardware accelerators. *Design reuse*, i.e. building systems from intellectual property (IP) components, allows savings in both design and verification efforts. To achieve a 10× design productivity gain by 2020 requires that complex SoCs consist of 90% reused components [1]. Further, it is increasingly critical to leverage *soft* IP components, which are designed once using high-level languages and implemented into various instances to meet the changing design requirements [2]. Their implementation is automated by synthesis tools, among which *high-level synthesis (HLS)*, e.g. from SystemC to RTL, has evolved to be of practical utility for early-design space exploration [3], especially at the micro-architectural level (Fig. 1). From an industrial viewpoint [4], however, this *component-based design paradigm* requires major progress on two levels: (i) at the component-level, in the design and cost/performance modeling of an IP component to increase its reusability across potential SoC designs and enable architecture exploration; (ii) at the system-level, in the automatic integration of IP components to find an optimal implementation of a given SoC.

Contributions. We propose a HLS-driven design methodology for compositional system-level design exploration that consists of two main steps. The first step is the development of libraries of synthesizable components which are both designed in high-level languages and characterized through HLS to maximize their reusability. In the second step, from the specification of a given system we construct a *system-level Pareto front* where each point represents a distinct implementation composed of optimal component instances, each of which is implemented in RTL by synthesizing its high-level specification. Our methodology aims at minimizing the total number of component synthesis to achieve a rich set of target system implementations. In particular, we propose:

978-3-9810801-8-6/DATE12/©2012 EDAA

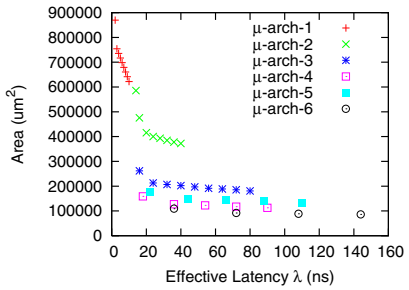


Fig. 1. HLS-driven implementation for a Discrete Cosine Transform.

- a concise component-library format, which preserves the freedom of system designers to select a wide range of component implementation alternatives to fit the specific design context of a system;
- an algorithm that, for a given system design context, prunes the design space of a component to minimize the set of its implementation candidates for integration into the system.
- an algorithm for system-level design space exploration which performs *HLS-planning* before invoking actual synthesis on the components. Thus, all the planned synthesis tasks can be run *in parallel* to minimize the total exploration time.

II. Background and Problem Formulation

Model of Computation. We adopt timed marked graph (TMG), a subclass of Petri nets (PN) [5], as the main model of computation. PN-based models allow compositional system-level performance analysis [6]. The simplicity of TMG allows us to perform rapid static analysis of interesting classes of systems without the need for simulation. Still, we also discuss extensions of TMG to model more general systems.

Definition 1. A PN is a bipartite graph defined as a 5-tuple (P, T, F, w, M_0), where P is a set of m places, T is a set of n transitions, $F : (P \times T) \cup (T \times P)$ is a set of arcs, $w : F \to \mathbb{N}^+$ is an arc weighting function, and $M_0 \in \mathbb{N}^m$ is the initial marking (i.e., the number of tokens at each $p \in P$) of the net.

Let $\bullet t = \{p|(p,t) \in F\}$ denote the set of input places of a transition t and $t\bullet = \{p|(t,p) \in F\}$ the set of output places. Let $\bullet p = \{t|(t,p) \in F\}$ denote the set of input transitions of a place p and $p\bullet = \{t|(p,t) \in F\}$ the set of output transitions. When a transition t fires, $w(p,t)$ tokens are removed from each $p \in \bullet t$ and $w(t,p)$ tokens are added to each $p \in t\bullet$. As a *firing rule* we assume that t fires as soon as it is enabled, i.e. each $p \in \bullet t$ has at least $w(p,t)$ tokens. A PN is *live* iff no transition can be disabled after any marking reachable from M_0, and *safe* iff the number of tokens at each place does not exceed one for any marking reachable from M_0.

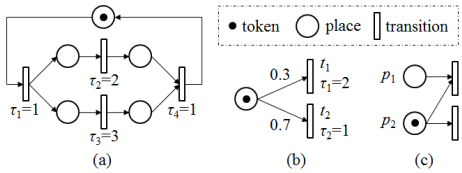


Fig. 2. (a) A timed marked graph; (b) A stochastic timed free-choice net where the token goes to t_1 (t_2) with a 30% (70%) probability; (c) A net that is not free-choice because $\bullet(p_2\bullet) = \{p_1, p_2\} \neq \{p_2\}$.

Definition 2. A TMG is a PN with $w : F \to 1$ and $\forall p \in P$, $|\bullet p| = |p \bullet| = 1$, extended with a transition firing-delay vector $\tau \in \mathbb{R}^n$, where τ_i denotes the duration time which the i-th transition takes for a firing.

Fig. 2(a) shows a simple TMG which models two concurrent data flows. The *minimum cycle time* [7] of a strongly connected TMG is:

$$c = \max_{k \in K} \{D_k / N_k\} \tag{1}$$

where K is the set of cycles in the TMG, D_k is the sum of transition firing-delays in cycle $k \in K$, and N_k is the number of tokens in the cycle. For example, the minimum cycle time of the TMG in Fig. 2(a) is $\max\{4/1, 5/1\} = 5$.

Since every place of a TMG has exactly one input and one output transition, TMG cannot model decisions/conflicts in a dynamic system. We then introduce another subclass of PN, free-choice net (FC) [5], to model decisions by non-deterministic token movement from a place.

Definition 3. An FC is a PN with $w : F \to 1$ and $\forall p \in P$, $|p \bullet| > 1 \implies \bullet(p\bullet) = \{p\}$.

If $|p \bullet| > 1$, the tokens at p can go to any $t \in p\bullet$. The flexibility of FC enables modeling of data dependencies, which commonly exist in stream-computing systems [8].

Definition 4. A stochastic timed FC is an FC extended with: *(i)* a routing rate associated with each arc $(p, t) \in F$ if $|p\bullet| > 1$ such that, for each such p, the sum of the routing rates equals 1, and *(ii)* a transition firing-delay vector $\tau \in \mathbb{R}^n$.

Fig. 2(b) shows an example of stochastic timed FC. A live and safe stochastic timed FC can be transformed to a behaviorally-equivalent TMG, whose minimum cycle time corresponds to the *average* cycle time of the original FC [9]. Therefore, although for simplicity in this paper we focus on TMGs, *our algorithmic findings are applicable to live and safe stochastic timed FC after performing a proper graph transformation.*

Effective Latency and Throughput. The *effective latency* λ of a component is the product of its clock cycle count and clock period. Given a system consisting of a set of components, we use a TMG to model the data flow of a system by representing each component with a transition t_i whose firing delay τ_i is equal to its effective latency. The maximum sustainable *effective throughput* θ of the system is defined as the reciprocal of the minimum cycle time of its TMG if this is strongly connected and as the minimum θ among its strongly-connected components, otherwise. In the sequel, we refer to "maximum sustainable effective throughput" simply as "effective throughput". The TMG of Fig. 2(a) models a system of 4 components with a θ of $1/5 = 0.2$.

The proposed methodology focuses on the design exploration of an SoC (or an SoC subsystem) that is realized by combining components, often in a pipeline fashion, which are specified with a high-level language. This fits well, for instance, a graphics or multimedia subsystem which is specified as a pipeline of SystemC modules communicating through TLM channels. We express the result of the system-level design exploration as a Pareto front which captures the system implementation trade-offs between θ and the *implementation cost* α.

TABLE I
SOME TYPICAL HIGH-LEVEL SYNTHESIS KNOBS AND THEIR SETTINGS

Knob	Setting
Loop Manipulation	Breaking, Unrolling, Pipelining
State Insertion	Number of States
Memory Configuration	Register Array, Embedded Memory
Clock Cycle Time	Nano-seconds

Ratio Distance. To quantify the accuracy of the Pareto front in characterizing the system-level design space we adopt the concept of *ratio distance* ($\mathcal{RD}$) [10]. Let the implementation cost and the effective throughput of a system-design point d on the Pareto front be d_α and d_θ, respectively. Intuitively, the ratio distance captures the greater ratio gap between the two points in terms of the objectives α and θ.

Definition 5. Given Pareto-optimal system-design points d and d' with $d'_\alpha \geq d_\alpha$ and $d'_\theta \geq d_\theta$, $\mathcal{RD}(d, d') = \max\{d'_\alpha/d_\alpha - 1, d'_\theta/d_\theta - 1\}$.

Finally, we can give a formal definition of the *Compositional System-Level Design Exploration Problem*, where we refer to synthesis tools as *oracles* and to "running the tools" as *"querying the oracles"*.

Problem 1. Given a HLS oracle, a target granularity $\delta > 0$, and a TMG model of a system with its components designed in high-level languages, construct a system-level Pareto front w.r.t. α versus θ, such that the maximum $\mathcal{RD}$ of two consecutive points on the front $\leq \delta$, by querying the HLS oracle on individual components as few times as possible.

III. DESIGN METHODOLOGY

The main difficulties of solving Problem 1 are: *(i)* at the *component* level, to identify the HLS knobs and their settings to query the oracle for optimal component instances, and *(ii)* at the *system* level, to identify critical components to query and to reuse the queries whenever possible for efficient Pareto front construction.

Knob-Settings. Table I lists a few typical HLS knobs and their settings. A knob may be applied multiple times with different settings. Besides, some settings may involve multiple parameters. A *knob-setting (KS)* is a particular setting of *all* the knob applications in a design. Hence a KS corresponds to the core content of a HLS script.

Example. To illustrate the application of different KSs on a component, we designed a DCT as a synthesizable SystemC component of an MPEG2 encoder. The DCT consists of nested loops of integer multiplications. Fig. 1 shows a characterization of the DCT implementation in terms of area vs. effective-latency trade-offs which is obtained by running HLS with the application of loop manipulations to the outermost loop (all inner loops are unrolled). Specifically, μ-arch-1 is the result of completely unrolling the loop to perform DCT in a single clock cycle; μ-arch-2 and μ-arch-3 are the results of partially unrolling the loop to perform DCT in 2 and 4 clock cycles, respectively; and μ-arch-4 to μ-arch-6 are the results of pipelining the loop with different initiation intervals and pipeline stages. For each μ-architecture, we sample its clock period to find the two extreme effective latencies, λ_{min} and λ_{max}, between which the DCT can be synthesized.

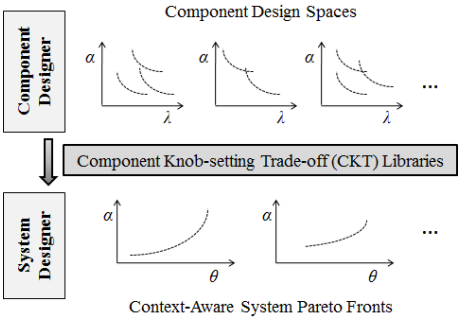


Fig. 3. The proposed design methodology.

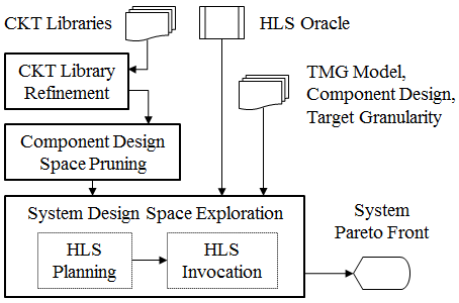


Fig. 4. Our algorithm flowchart in the context of system design.

In this example, we see *intra*-micro-architecture trade-offs explored by adjusting the clock period. There are also *inter*-micro-architecture trade-offs. For example, μ-arch-1 can achieve shorter effective latencies by parallelizing all the multiplications at the cost of greater area occupation due to fully-duplicated multipliers. To distinguish the latter trade-offs, we denote two KSs as *distinct* when they differ by at least one setting among all the knobs, except the technology-dependent knobs such as the clock period. For instance, there are 6 distinct KSs in Fig. 1.

The Proposed Methodology. Fig. 3 illustrates the two main steps of our approach to solve Problem 1.

First, feasible KSs of the component are determined based on the *component designer*'s knowledge. In fact, infeasible KSs can yield component instances that violate design specifications, e.g., arbitrary state insertion may result in instances that do not satisfy clock-cycle-count constraints. After determining the feasible KSs, the component designer should also find the extreme-λ: this step requires only a one-time effort and its results can be reused by different system designers.

Then, at the time of component integration, the *system designer* has the freedom to pre-select a subset of the feasible KSs for each component based on the specific context of the given system design. Hence, different pre-selection schemes will result in *context-aware system Pareto fronts* as shown in Fig. 3 [1]. For example, in addition to typical costs like area and power, which have been characterized in the component libraries, if system designers want to avoid the higher mask/testing cost and/or limited yield due to the deployment of embedded memories [11], then they can set a "don't use" attribute to the knob-settings that involve embedded memories before running HLS. Since the KSs involving register arrays generally yield component instances with greater area than those involving embedded memories, a *reusable* component

[1]In Fig. 3, each θ of the system is actually calculated from the λ's of the components (one of each) as discussed in Section II.

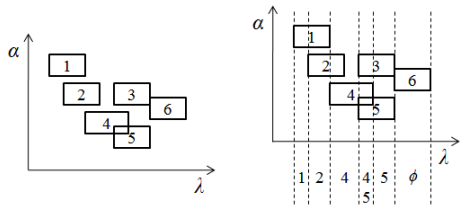


Fig. 5. (a) An instance of Problem 2. (b) Dashed lines identify six non-overlapping λ intervals; their optimal knob-setting candidates for deriving Pareto-optimal instances are shown below the λ axis.

library should not be limited to expose only area-optimal KSs. Consequently, to keep all the feasible KSs for any possible given system-design context while maintaining a concise library format, we propose the notion of *Component Knob-setting Trade-off (CKT) Library*.

A CKT library, which is provided by component designers, stores all the *distinct* feasible KSs of a component: each KS indicates its fixed clock cycle count, its extreme implementation costs, α_{max} and α_{min}, and its extreme effective latencies, λ_{min} and λ_{max}. The average or the worst-case clock cycle count can be used when the clock cycle count is not deterministic. The fixed clock cycle count allows us to explore the intra-knob-setting α vs. λ trade-off by adjusting the clock period to synthesize the component. In addition, the extreme values of α and λ outline the sub-design space of a KS.

To assist the system designer, we propose the algorithms charted in Fig. 4 to explore the design space given the CKT libraries. In the context of a given system design, for each component the CKT library can optionally be refined by pre-selecting KSs as desired and/or re-characterizing the extreme α and λ if a different technology is considered (*CKT library refinement* in Fig. 4). For each refined CKT library, we then apply *component design space pruning* to find out KS candidates for deriving optimal component instances (Section IV). Next, for *system design space exploration*, we propose a HLS-planning algorithm to identify critical oracle queries before actually invoking them (Section V).

IV. Component Design Space Pruning

For each refined CKT library with s distinct KSs, we consider the following problem, where a KS *dominates* another KS iff the former can achieve a lower implementation cost α w.r.t. a fixed effective latency λ, or the former can achieve both a lower α and a shorter λ.

Problem 2. Given s distinct knob-settings, each of which yields a component sub-design space bounded by $(\lambda^i_{min}, \alpha^i_{max})$ and $(\lambda^i_{max}, \alpha^i_{min})$, $i \in \{1, 2, \ldots, s\}$, find the ordered list of non-overlapping λ intervals, each of which indicates its non-dominated knob-settings.

Fig. 5(a) shows an instance of Problem 2 with 6 distinct KSs, each of which yields a design-space *rectangle* bounded by upper-left and lower-right extreme points. In a certain λ interval, KS 3 is dominated by both KS 4 and KS 5 because they can always yield lower-cost instances than KS 3. A cross-λ-interval example is that KS 6 is dominated by KS 5. Obviously, only non-dominated KSs should be kept as candidates for deriving Pareto-optimal component instances. Our goal is to find the ordered list of non-overlapping λ intervals with their optimal KS candidates, as shown in Fig. 5(b).

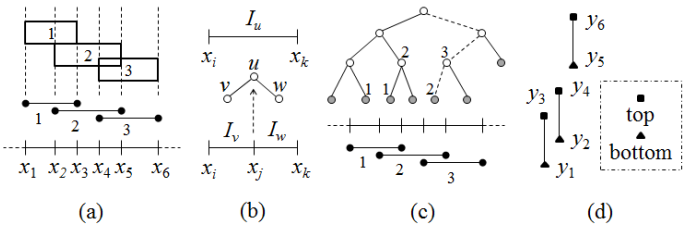


Fig. 6. Illustration of Algorithm 1; see Section IV for explanations.

Theorem 1. *Problem 2 requires $\Omega(s \log s)$ time to compute exact solutions*[2].

For Problem 2, we propose an algorithm based on *segment trees* [12]. Given s KSs with s pairs of extreme λ's, $(\lambda^i_{min}, \lambda^i_{max})$, $i \in \{1, 2, \ldots, s\}$, by regarding these s pairs as s *horizontal* line segments with $2s$ end points, whose ordered x-coordinates are $x_1 < x_2 < \cdots < x_{2s}$, we have the following $2s + 1$ *atomic intervals*: $[-\infty, x_1], [x_1, x_2], \ldots, [x_{2s-1}, x_{2s}], [x_{2s}, \infty]$.

Fig. 6(a) shows an example of 3 KSs and 7 atomic intervals: the middle horizontal line segments are projected from the upper rectangles (i.e. the sub-design spaces of the 3 KSs), and the bottom atomic intervals are derived from the middle line segments. Next, we describe the segment-tree data structure, which is used to store the input KSs as line segments.

A *segment tree* is a balanced binary tree, such that: *(i)* each leaf node represents an atomic interval, *(ii)* each non-leaf node u, with children v and w, represents an interval $I_u = I_v \cup I_w$, i.e., u represents a coarser interval partition of its children's (see Fig. 6(b), where the common x-coordinate x_j of I_v and I_w is stored in node u), and *(iii)* an input line segment can be split into several parts, each of which is stored in a tree node as close to the root as possible. Fig. 6(c) shows an example segment tree, where the atomic intervals and the input line segments are copied from Fig. 6(a) and shown below the tree; the number right above a tree node indicates the line segments stored in that node.

To retrieve the line segments which enclose an x-coordinate x, we can start from the root, by binary search (comparing x with the x-coordinate stored in the non-leaf nodes), down to the leaf node which encloses x. For instance, in Fig. 6(a), to retrieve the line segments that enclose the atomic interval $[x_4, x_5]$, we follow the dashed trace of Fig. 6(c) and obtain Segments 2 and 3. Since a stored line segment corresponds to a KS, the tree search procedure allows to retrieve all the KSs whose $[\lambda_{min}, \lambda_{max}]$ encloses any given atomic interval.

For each atomic interval which is enclosed by r KSs, we present the following procedure to identify its non-dominated KSs. First, these r KSs have r pairs of extreme α's. Similarly, we regard these r pairs as r *vertical* line segments with $2r$ end points, whose ordered y-coordinates are $y_1 \le y_2 \le \cdots \le y_{2r}$. We assign two end-point types to each line segment, whose end-point y-coordinates are $y_i < y_j$: *(i)* "bottom" for the y_i-end and *(ii)* "top" for the y_j-end (Fig. 6(d) shows an example of three line segments). Iterating through y_i for $i = 2, 3, \ldots, r$—trivially, the y_1-end is a bottom end, and the first/lowest line segment is non-dominated—we check the end-point type of y_i: if y_i is a bottom end, the y_i-corresponding line segment overlaps previous ones, and therefore is non-dominated; otherwise, we stop iterating. For example, in Fig. 6(d), the procedure stops after knowing that y_3 is a top end, and thus identifies $[y_1, y_3]$ and $[y_2, y_4]$ as non-dominated line segments.

[2]Due to the space limitation, we skip all the proofs in this paper.

Algorithm 1 Component Design Space Pruning

Input: s distinct knob-settings with extreme λ's and α's
Output: atomic λ intervals w/ non-dominated knob-settings
1: $I \equiv [-\infty, x_1], \ldots, [x_{2s}, \infty] \leftarrow$ sort knob-settings by λ
2: $\Upsilon \leftarrow$ build a segment tree for I
3: $y_{min} \leftarrow \infty$
4: **for** $i = 1 \rightarrow 2s - 1$ **do**
5: $\quad S \leftarrow$ search Υ by $x = (x_i + x_{i+1})/2$
6: $\quad$ prune S by y_{min}
7: $\quad y_1 \le \cdots \le y_{2r} \leftarrow$ sort S by α
8: $\quad$ **for** $j = 1 \rightarrow r$ **do**
9: $\quad\quad$ **if** the y_j-end is a bottom end **then**
10: $\quad\quad\quad$ output the y_j-corresponding knob-setting
11: $\quad\quad$ **else**
12: $\quad\quad\quad$ stop this loop
13: $\quad\quad j \leftarrow j + 1$
14: $\quad$ update y_{min}
15: $\quad i \leftarrow i + 1$

Algorithm 1 gives the pseudocode of our complete procedure for Problem 2. Note that lines 3, 6, and 14 prune the KSs which are dominated *across* atomic intervals.

Theorem 2. *The running time of Algorithm 1 is $O(s \log s + sr \log r)$, where s is the number of distinct knob-settings, and r is the max number of enclosing knob-settings of an atomic interval.*

Since $r \le s$, in the worst case the running time is $O(s^2 \log s)$, whereas in the case of $r \log r = O(\log s)$ the algorithm runs in $O(s \log s)$ time, which achieves the theoretical lower bound indicated by Theorem 1.

V. System Design Space Exploration

Given a strongly connected TMG, to compute its minimum cycle time (Equation (1)), Maggot proposed the linear program [13]:

$$\begin{aligned} \min \quad & c \\ \text{s.t.} \quad & A\sigma + M_0 c \ge \tau^- \\ & c \ge 0, \end{aligned} \tag{2}$$

where c is the minimum cycle time, $\sigma \in \mathbb{R}^n$ is the transition-firing initiation-time vector, $M_0 \in \mathbb{N}^m$ is the initial marking, $\tau^- \in \mathbb{R}^m$ is the input-transition firing-delay vector (i.e., τ^-_i equals the firing-delay τ_k of the $t_k \in \bullet p_i$), and $A \in \{-1, 0, 1\}^{m \times n}$ is the incidence matrix such that

$$A[i,j] = \begin{cases} 1 & \text{if } t_j \text{ is the output transition of } p_i, \\ -1 & \text{if } t_j \text{ is the input transition of } p_i, \\ 0 & \text{otherwise.} \end{cases}$$

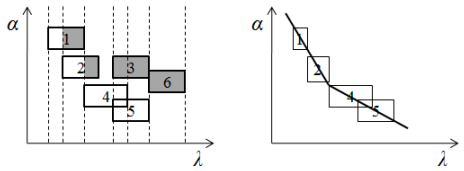
In this linear program, c and σ are decision variables. A running example of the linear program can be found in [14].

Based on Equation (2), we propose the following effective-throughput-θ-constrained cost-minimization linear program:

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^{n} f_i(\tau_i) \\ \text{s.t.} \quad & A\sigma + M_0/\theta \ge \tau^- \\ & \tau^-_{min} \le \tau^- \le \tau^-_{max}, \end{aligned} \tag{3}$$

where the function f_i yields the ith component's implementation cost given the firing-delay τ_i of the transition t_i (i.e., the λ of the t_i-corresponding component equals τ_i), and τ^-_{min} and τ^-_{max} are given according to the extreme λ's of the components. The decision variable is τ, i.e. the λ requirements of the components.

In general, the cost functions f_i's in Equation (3) are unknown a-priori. However, based on our experimental results and recent publications [15], [16], we observe that these cost functions generally exhibit *convexity* [17]. Therefore, for each



(a) (b)

Fig. 7. (a) Pruned component design space: shaded space is sub-optimal. (b) Pruned-space approximation by a convex piecewise-linear function.

component, given its pruned design space (discussed in Section IV), we propose to approximate its cost function f_i with *convex piecewise-linear* functions. Fig. 7 shows an example approximation using a convex piecewise-linear function[3]

$$\bar{f}(\lambda) = \max_{j=1,2}(a_j\lambda + b_j),$$

where a_j and b_j are coefficients. To fit a given set of data points using a convex piecewise-linear function while minimizing its total squared-fitting errors can be solved *optimally* in polynomial time with quadratic programming [17].

Note that, due to the design space pruning procedure presented in Section IV, the fitting function $\bar{f}$ is not only convex piecewise-linear but also *monotonically decreasing*. If we replace the f_i's in Equation (3) with the fitting functions $\bar{f}_i$'s, the objective function $\sum_{i=1}^{n} \bar{f}_i$ will also be convex piecewise-linear and monotonically decreasing. A minimization linear program with its objective function being convex piecewise-linear can be transformed to a standard linear program [17] and therefore is polynomial-time solvable. Besides, the monotonically-decreasing objective function guarantees that all the components not in the performance-critical cycle of the TMG will be run at their maximum possible λ's to minimize their implementation costs. As a result, our linear program is not only computationally tractable, but can also minimize the number of distinct non-critical component instances across different system throughput θ requirements.

Finally, Algorithm 2 gives the pseudocode of our complete procedure for Problem 1. Note that in line 10, to fill the gap where the maximum $\mathcal{RD} > \delta$ is due to consecutive α's instead of θ's, we solve the following α-constrained θ-maximization linear program:

$$\begin{array}{ll} \max & \theta \\ \text{s.t.} & A\sigma + M_0/\theta \geq \tau^- \\ & \sum_{i=1}^{n} \bar{f}_i(\tau_i) \leq \alpha \\ & \tau^-_{min} \leq \tau^- \leq \tau^-_{max}, \end{array} \tag{4}$$

where θ and τ are decision variables of a total size $\leq n+1$.

We remark that in Algorithm 2: *(i)* we do not invoke oracle queries until we have identified all the non-redundant λ requirements, thereby minimizing the number of actual queries, whose runtime will dominate the whole exploration procedure, and *(ii)* we can parallelize all the linear-program solving (lines 8 and 10, by pre-computing the geometric progressions of θ and α), and most importantly, the time-consuming oracle queries (line 11).

VI. Experimental Results

We adopted an MPEG2 encoder designed in SystemC as our benchmark, which is synthesizable by a commercial HLS tool with an industrial $45nm$ technology library. The MPEG2 encoder contains 22 components, whose data flow can be modeled by a live and safe stochastic timed FC. In particular,

[3]Generally, the more number of linear functions are used, the more accurate the approximation can be.

Algorithm 2 System Design Space Exploration

Input: a HLS oracle, a target granularity δ, and a TMG model of a system with a set of components
Output: α vs. θ trade-off
1: **for all** components **do**
2: $\Sigma_i \leftarrow$ run Algorithm 1 for the current component
3: $\bar{f}_i \leftarrow$ run convex piecewise-linear fitting for Σ_i
4: $\Pi \leftarrow$ an empty hash table for optimal λ requirements
5: $\theta \leftarrow$ min throughput w/ all components at max λ's
6: $\hat{\theta} \leftarrow$ max throughput w/ all components at min λ's
7: **while** $\theta \leq \hat{\theta}$ **do**
8: $\Pi \leftarrow$ solve Equation (3) with $\bar{f}_i$
9: $\theta \leftarrow \theta \times (1+\delta)$
10: $\Pi \leftarrow$ solve Equation (4) whenever necessary
11: query the oracle according to Π and Σ

after graph transformation [9], 14 out of the 22 components form a strongly-connected TMG. For these 14 components, we characterized, on average, 10.9 distinct KSs per component to build CKT libraries. Note that these CKT libraries can thereafter be reused in system designs other than the MPEG2 encoder. In our MPEG2-encoder (system) design context, we simply kept all the distinct KSs as implementation alternatives. After applying Algorithm 1 to these CKT libraries, we got, on average, 2.8 non-dominated KSs per atomic interval. This result shows that in order to derive an optimal component instance given its λ requirement, instead of synthesizing at all the available KSs, we can prune on average 74% $((10.9-2.8)/10.9)$ of HLS invocations.

We then applied Algorithm 2 with a target granularity $\delta = 0.2$ to explore the area vs. throughput trade-off of the MPEG2 encoder. For each pruned component design space, we used 3 linear functions to compose a convex piecewise-linear approximation. The MPEG2-encoder exploration result is shown in Fig. 8(a), where the "approximation" points result from solving Equations (3) and (4), while the "oracle result" points are derived by invoking actual HLS. To quantify the mismatch between an "approximation" point d_1 and an "oracle result" point d_2 w.r.t. a specific throughput, we apply the following area-mismatch ($\mathcal{AM}$) metric:

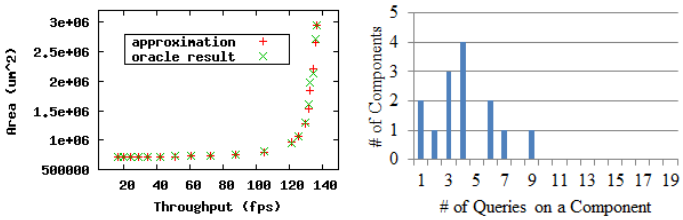
$$\mathcal{AM}(d_1, d_2) = |d_{1\alpha} - d_{2\alpha}|/d_{2\alpha},$$

where $d_{1\alpha}$ ($d_{2\alpha}$) is the area of d_1 (d_2). The exploration in Fig. 8(a) yields 19 system-design points, with an average $\mathcal{AM}$ of 1% and a max $\mathcal{AM}$ of 7%. In particular, 17 out of the 19 points have an $\mathcal{AM} \leq 3\%$. This result shows that component design approximation based on piecewise-linear functions (Fig. 7(b)) delivers a good first-order approximation of the system Pareto front.

Fig. 8(b) shows the oracle query distribution to achieve the result of Fig. 8(a).[4] The maximum number of queries on a component is 19, i.e. to query each component once for each system throughput requirement. However, our exploration algorithm can identify critical queries by solving Equations (3) and (4), catch the effective-latency requirements, and reuse them. Hence, as shown in Fig. 8(b), there are 10 components each getting queried less than 5 times, and all the 14 components each get queried no more than 9 times. In total, 57 component queries were made. In comparison with exhaustive queries (19×14 times), we achieve a 4.7× query efficiency.

In terms of the HLS runtime, instead, we achieve a 3.5× speed-up because the exploration in Fig. 8(a) requires 35-hour *sequential* HLS-tool invocations, whereas sequential ex-

[4]Note that an oracle query may invoke the HLS tool more than once, depending on the number of non-dominated knob-settings. We use the terms "oracle query" and "synthesis invocation" separately.



(a) (b)

Fig. 8. (a) Result of design-space exploration for MPEG2-encoder in terms of area vs. throughput trade-off (frame size is 352×240); target granularity is $\delta = 0.2$. (b) Query distribution for the exploration in (a).

haustive synthesis requires 124 hours. In fact, compared with the other steps in Algorithm 2 that run in seconds, the HLS tool invocation dominates the whole exploration runtime. Considering that the maximum runtime of a single component synthesis in our case is 42 minutes, the power of being able to parallelize all the synthesis tasks is significant. This ability is another advantage of our HLS-planning algorithm.

We summarized the afore-discussed results for $\delta = 0.2$ in Table II, where the results for $\delta \in \{0.15, 0.1\}$ are also listed. As δ decreases from 0.2 to 0.1, the number of explored system-design points increases from 19 to 35. In terms of the approximation accuracy, the average $\mathcal{AM}$ remains at 1% as δ varies. While the maximum $\mathcal{AM}$ increases slightly as δ decreases, there are constantly $\sim$90% points with $\mathcal{AM} \leq 3\%$, showing the general approximation robustness even though we improve the granularity δ. Further, the total number of oracle queries grows from 57 to 77, in order to explore more system-design points. Meanwhile, as δ decreases, the query efficiency compared with exhaustive queries improves from 4.7× to 6.4×, thus implying that critical queries are reused even more effectively among system-design points. The same trend can also be found w.r.t. the sequential HLS tool runtime which increases from 35 hours to 46 hours as needed; meanwhile, the speed-up over exhaustive synthesis goes up from 3.5× to 5.0×. Both trends suggest that our exploration algorithm scale well with δ.

VII. Related Work

General design space exploration algorithms include local-search heuristics like Simulated Annealing [18] and Genetic Algorithms [19]. Techniques based on clustering dependent parameters were proposed to prune the search space before exploration [20], [21]. Techniques based on predicting design qualities were also proposed to reduce the number of simulation/synthesis during exploration [22], [23].

For component-based design, various approaches which exploit design hierarchies to compose system-level Pareto fronts have been proposed, e.g. [10], [24], [25]. Among these, our work is closer to the top-down approach of the "supervised exploration framework", which adaptively selects RTL components for logic synthesis and thus gradually improves the accuracy of the system Pareto front [10]. However, the important differences are that we operate at a higher level of abstraction, can handle components specified in SystemC, and can parallelize HLS runs to construct system Pareto fronts.

VIII. Conclusions

We have presented a compositional methodology for HLS-driven design-space exploration of SoCs. Our approach is based on two new algorithms that combined allow us to derive a set of alternative Pareto-optimal implementations for a given SoC specification by selecting and combining the best implementations of its components from a pre-designed soft-IP library.

TABLE II
MPEG2 encoder area vs. throughput exploration.

δ	0.2	0.15	0.1
# System-Design Points	19	24	35
Average $\mathcal{AM}$	1%	1%	1%
Maximum $\mathcal{AM}$	7%	10%	11%
# Points w/ $\mathcal{AM} \leq 3\%$	17	22	31
Total # Queries	57	63	77
Query Efficiency	4.7×	5.3×	6.4×
Sequential HLS Tool Runtime	35hr	38hr	46hr
Runtime Speed-up	3.5×	4.1×	5.0×

Acknowledgments. This work is partially supported by an ONR Young Investigator Award, an Alfred P. Sloan Foundation fellowship, and the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

References

[1] "2009 ITRS," *Available at http://public.itrs.net.*
[2] W. Cesário *et al.*, "Component-based design approach for multicore SoCs," in *Proc. of DAC*, 2002, pp. 789–794.
[3] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 18–25, Aug. 2009.
[4] W. Kruijtzer *et al.*, "Industrial IP integration flows based on IP-XACT™ standards," in *Proc. of DATE*, 2008, pp. 32–37.
[5] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[6] A. S. Tranberg-Hansen and J. Madsen, "A compositional modelling framework for exploring MPSoC systems," in *Proc. of CODES+ISSS*, 2009, pp. 1–10.
[7] C. V. Ramamoorthy and G. S. Ho, "Performance evaluation of asynchronous concurrent systems using Petri nets," *IEEE Trans. on Software Engineering*, vol. 6, pp. 440–449, 1980.
[8] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, "Synthesis of embedded software using free-choice Petri nets," in *Proc. of DAC*, 1999, pp. 805–810.
[9] J. Campos and J. M. Colom, "A reachable throughput upper bound for live and safe free choice nets," in *Proc. of the Intl. Conf. on Application and Theory of Petri Nets*, Gjern, Denmark, Jun. 1991, pp. 237–256.
[10] H.-Y. Liu, I. Diakonikolas, M. Petracca, and L. P. Carloni, "Supervised design space exploration by compositional approximation of Pareto sets," in *Proc. of DAC*, Jun. 2011, pp. 399–404.
[11] E. J. Marinissen, B. Prince, D. Keltel-Schulz, and Y. Zorian, "Challenges in embedded memory design and test," in *Proc. of DATE*, Mar. 2005, pp. 722–727.
[12] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.
[13] J. Maggot, "Performance evaluation of concurrent systems using Petri nets," *Information Processing Letters*, pp. 7–13, 1984.
[14] T. Yamada and S. Kataoka, "On some LP problems for performance evaluation of timed marked graphs," *IEEE Trans. on Automatic Control*, vol. 39, no. 3, pp. 696–698, Mar. 1994.
[15] H. Javaid, X. He, A. Ignjatovic, and S. Parameswaran, "Optimal synthesis of latency and throughput constrained pipelined MPSoCs targeting streaming applications," in *Proc. of CODES+ISSS*, 2010, pp. 75–84.
[16] A. Kondratyev, L. Lavagno, M. Meyer, and Y. Watanabe, "Realistic performance-constrained pipelining in high-level synthesis," in *Proc. of DATE*, Mar. 2011, pp. 1–6.
[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[18] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *Proc. of VLSI-DAT*, Apr. 2009, pp. 106–109.
[19] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. on Computers*, vol. 55, pp. 99–112, Feb. 2006.
[20] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Trans. on VLSI Systems*, vol. 10, no. 4, pp. 416–422, Aug. 2002.
[21] B. C. Schafer and K. Wakabayashi, "Design space exploration acceleration through operation clustering," *IEEE TCAD*, vol. 29, no. 1, pp. 153–157, Jan. 2010.
[22] G. Beltrame, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *IEEE TCAD*, vol. 29, no. 7, pp. 1083–1095, Jul. 2010.
[23] G. Mariani *et al.*, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proc. of DAC*, 2010, pp. 120–125.
[24] C. Haubelt and J. Teich, "Accelerating design space exploration using Pareto-front arithmetics," in *Proc. of ASPDAC*, Jan. 2003, pp. 525–531.
[25] M. Geilen and T. Basten, "A calculator for Pareto points," in *Proc. of DATE*, 2007, pp. 285–290.