Supervised Design Space Exploration by Compositional Approximation of Pareto Sets

Hung-Yi Liu[1], Ilias Diakonikolas[2], Michele Petracca[1], and Luca Carloni[1]
Dept. of Computer Science, Columbia Univ.[1] Dept. of EECS, UC Berkeley[2]
{hungyi, petracca, luca}@cs.columbia.edu[1], ilias@eecs.berkeley.edu[2]

ABSTRACT

Technology scaling allows the integration of billions of transistors on the same die but CAD tools struggle in keeping up with the increasing design complexity. Design productivity for multi-core SoCs increasingly depends on creating and maintaining reusable components and hierarchically combining them to form larger composite cores. Characterizing such composite cores with respect to their power/performance tradeoffs is critical for design reuse across various products and relies heavily on synthesis tools. We present CAPS*, an online adaptive algorithm that efficiently explores the design space of any given core and returns an accurate characterization of its implementation tradeoffs in terms of an approximate Pareto set. It does so by supervising the order of the time-consuming logic-synthesis runs on the core's components. Our algorithm can provably achieve the desired precision on the approximation in the shortest possible time, without having any a-priori information on any component. We also show that, in practice,* CAPS *works even better than what is guaranteed by the theory.*

Categories and Subject Descriptors
B.6.3 [**Design Aids**]: Automatic synthesis
General Terms
Algorithms, Design, Performance
Keywords
System-Level Design, System-on-Chip, Design Reuse.

1. INTRODUCTION

Future multi-core systems-on-chip (SoC) will host billions of transistors to implement an ever growing, increasingly-heterogeneous collection of functional blocks. Growth in available transistors has outpaced the CAD tools' ability to design them (*design productivity gap*), a gloomy trend for the entire semiconductor investment cycle [15]. As explained in [6], it is necessary to go beyond current design methodologies that are "built around hard design blocks for a specific technology" and develop instead new ones that rely on "soft design blocks or macros, described at a higher level of abstraction, such as RTL description in Verilog" [6]. These functional blocks, aka *soft Intellectual Property (IP) cores*, are designed, validated, and properly characterized only once and in a way that simplifies migration across technology nodes and design reuse. Furthermore, the bulk of the design effort will consist in the system-level optimization of the specific SoC under design through the proper selection and composition of the parametrized soft IP cores [6]. For most chip designs, the goal of system-level optimization will be neither low power nor high performance, but instead an *elusive combination of these two objectives*: i.e. achieving power efficiency without affecting the system's performance [14].

In this *component-based design* approach, high-level and logic synthesis tools will play a central role. Only a small number of function blocks will be designed from scratch for a new product.

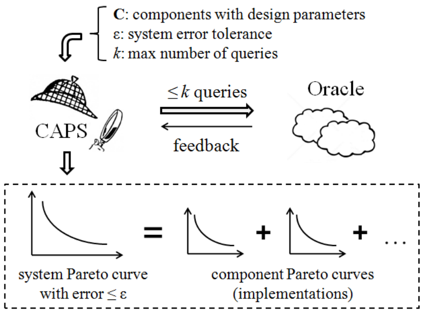

C: components with design parameters
ε: system error tolerance
k: max number of queries
CAPS
≤ k queries
feedback
Oracle
=
+
+
...
system Pareto curve with error ≤ ε
component Pareto curves (implementations)

Figure 1: Compositional Approximation of Pareto Sets.

Most of the soft IP cores will be selected for reuse from in-house libraries of legacy designs or acquired from third-party providers. Indeed, according to the ITRS, in order to achieve a 10× design-productivity improvement by 2020, an SoC for embedded applications (either consumer mobile or consumer stationary) will need to consist of 90% of reused design [15]. This requires to create and maintain reusable cores, a task that is estimated to be 2–5× more difficult than their creation for one-time use [15]. Hence, designers will increasingly rely on synthesis tools for the automatic generation of the low-level circuit implementation of a given soft IP core after having explored various configurations of its high-level parameters to achieve a system-level optimization objective. Synthesis-driven design space exploration, however, faces major challenges. The effectiveness of *synthesis tools* is limited by the long execution times that they suffer even when applied to a medium-size IP core. Not only to synthesize an entire complex SoC in a single run goes beyond the capabilities of today's synthesis tools, but to fully characterize a medium-size IP core with respect to its operating points in terms of power and performance may easily require tens of logic synthesis runs for each of its components.

Contributions. We present a novel and general approach to effectively coordinate the execution of synthesis runs needed to explore the design space of a system that consists of a set C of interacting components. We model this computational task as a multi-objective optimization problem, in a rigorous mathematical framework, which we then solve with CAPS, a new algorithm that can derive an approximation of the *Pareto set* of the system by iteratively invoking a commercial tool (*the oracle*) to synthesize various instances of the components and, in doing so, implicitly build approximations for their Pareto sets (Fig. 1). CAPS is optimal in our framework, and experimental results show that it works extremely well in practice. In particular, our algorithm:

- is the (provably) optimal online algorithm, within a natural framework of bi-objective optimization problems, in terms of *(i)* the accuracy of the returned approximation and *(ii)* the CPU time needed to achieve such an approximation;
- is inherently scalable; it works only on the single components, and never runs a synthesis of the whole system, which would be impractical and/or infeasible for large systems;
- copes with the inherent *limited-information* setting in our context: i.e. the facts that no information about any component is known a-priori and at any given step the synthesis

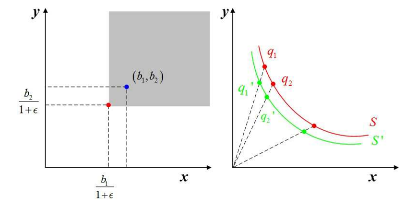


Figure 2: Point b ϵ-covers all points in the shaded region (left); Computing the ratio distance between curves S and S' (right).

- tool can only sample *one* point of the design space of *one* component;
- works on a bi-objective optimization problem coping with the typical asymmetry of synthesis tools, which allow controllability only on the clock period, but leave just observability on the power and area of the returned implementation.

In Sections 2 and 3 we provide a rigorous presentation of our algorithm and its theoretical properties, while in Section 4 we demonstrate the practical advantages of our approach with a rich set of experimental results. Related work is discussed in Section 5.

2. MODEL AND BASIC DEFINITIONS

A multiobjective optimization problem Π has a set $\mathcal{I}$ of *valid instances*, every instance $I \in \mathcal{I}$ has a set of feasible solutions $\mathcal{S}(I)$. There are d objective functions, $f_1, \ldots, f_d$, each mapping an instance I and a solution $s \in \mathcal{S}(I)$ to a value $f_j(I, s)$. The problem specifies for each objective whether it is to be maximized or minimized. We assume that the objective functions have positive values. In our context, a "feasible solution" at the system/component level is an implementation of the system/component.

Dominance Relation, Pareto Set. We say that a d-vector u *dominates* another d-vector v if it is at least as good in all the objectives, i.e. $u_j \geq v_j$ if f_j is to be maximized ($u_j \leq v_j$ if f_j is to be minimized). Similarly, we define domination between any solutions according to the d-vectors of their objective values. The *Pareto set* $P(I)$ of an instance I is the set of undominated d-vectors of values of the solutions in $\mathcal{S}(I)$. For any instance, the Pareto set is unique.

Approximate Pareto Set. We say that a d-vector u *ϵ-covers* another d-vector v ($\epsilon \geq 0$) if u is at least as good as v up to a factor of $1 + \epsilon$ in all the objectives, i.e. $u_j \geq v_j/(1+\epsilon)$ if f_j is to be maximized ($u_j \leq (1+\epsilon)v_j$ if f_j is to be minimized). Given an instance I and $\epsilon > 0$, an *ϵ-Pareto set* $P_\epsilon(I)$ is a set of d-vectors of values of solutions that $(1+\epsilon)$-cover all vectors in $P(I)$.

Throughout this paper, we will be concerned with bi-objective problems, i.e. optimization problems with two objective functions (or criteria), which we denote with symbols x and y, respectively. In the experiments of Section 4, x is the clock period of a circuit and y is either its power dissipation or area occupation. Hence, in our setting, both objectives are to be minimized. Consider the plane whose coordinates correspond to the two objectives. Every feasible solution (design) s is mapped to a point q on this plane. We use $x(q), y(q)$ to denote the coordinates of q, i.e. $q = \big(x(q), y(q)\big)$.

We define the *ratio distance* from p to q as $\mathcal{RD}(p, q) = \max\{x(q)/x(p)-1, y(q)/y(p)-1, 0\}$. By definition, the value $\mathcal{RD}(p, q)$ is the minimum value of $\epsilon \geq 0$ such that q ϵ-covers p. Intuitively this measure captures "how much worse is point q from point p". Note that this notion is asymmetric in p and q, which is why the ratio distance is not symmetric (see example below). If $S, S' \subseteq \mathbb{R}^2_+$, we define the *ratio distance between sets of points* as $\mathcal{RD}(S, S') = \max_{q \in S} \min_{q' \in S'} \mathcal{RD}(q, q')$. In words, for a given point $q \in S$ we find its closest point $q' \in S'$ (according to the ratio distance) and then take the maximum over all points in S.

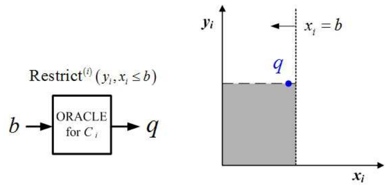


Figure 3: The oracle solves the Restricted Problem while guaranteeing that there are no solution points in the shaded region.

As a corollary, set $S \subseteq A$ is an ϵ-Pareto set for A if and only if $\mathcal{RD}(A, S) \leq \epsilon$. These definitions are illustrated in Fig. 2.

Example: Consider the points $q_1 = (1, 10)$ and $q_2 = (5, 6)$. Intuitively, we would like to say that q_2 is 4 times worse than q_1 (because of the ratio in the first coordinate), while q_1 is only $2/3$ times worse than q_2 (because of the ratio in the second coordinate). This is captured by the ratio distance and approximate dominance relation. According to our definitions, $\mathcal{RD}(q_1, q_2) = 4$ and q_2 4-covers q_1. On the other hand, $\mathcal{RD}(q_2, q_1) = 2/3$. Hence, q_1 2/3-covers q_2. □

Restricted Problem and Oracle Access. We stress that the objective space of our bi-objective problem is not given explicitly, but rather implicitly through the instance. In particular, we access the objective space $\mathcal{I}$ of Π via an appropriate oracle. We now describe our way of accessing the Pareto set *for each individual component.* Fix a component C_i. We assume we can efficiently minimize one objective (the y-coordinate, aka power) subject to a constraint on the other (the x-coordinate, aka clock period). Our *oracle* is an efficient program that solves the following optimization problem.

Restricted Problem (for the y-objective): For a given component C_i (e.g. with a set of possible feasible designs $\mathcal{S}(C_i)$ discoverable by the commercial tool) and a bound b, either return a feasible solution point (i.e. a design for this component) q_i satisfying $x(q_i) \leq b$ and $y(q_i) \leq \min\{y$ over all designs $q \in \mathcal{S}(C_i)$ with $x(q) \leq b\}$ or report that there does not exist any solution q such that $x(q) \leq b$.

For simplicity, we will drop the instance from the notation and use Restrict$^{(i)}(y, x \leq b)$ to denote the solution returned by the corresponding oracle. (The superscript "(i)" means that we are dealing with the i-th component.)

We call an oracle that satisfies the above *"ideal"*. In Sections 2 and 3 we assume an ideal oracle. In Section 4 we discuss the performance of our approach when the oracle presents a noisy behavior. In practice, if the oracle does not return a solution, it returns its best achievable value for x, i.e. its lowest bound (Fig. 3.)

Combining Components. Let $n \geq 1$ be the number of components $\{C_i\}_{i=1}^n$. Let $q_i \in \mathcal{S}(C_i)$, $i \in [n]$ be a feasible design for the i-th component and q be the design for the system obtained as the union of the q_i's. In particular, q_i is a 2-dimensional vector – point in the xy plane – whose first coordinate is the x-value (the clock period of the corresponding design) and whose second coordinate is its y-value (its power) We assume there are two *combining functions* $f_x, f_y : \mathbb{R}^n \to \mathbb{R}$ that specify how q is related to the q_i's. In particular, $x(q) = f_x(x(q_1), \ldots, x(q_n))$ and $y(q) = f_y(y(q_1), \ldots, y(q_n))$. Both functions are assumed to be monotone increasing in each coordinate and efficiently computable. In our concrete setting, $f_x(x_1, \ldots, x_n) = \max_i x_i$ and $f_y(y_1, \ldots, y_n) = \sum_i y_i$, i.e. the clock period is the maximum of the periods and the total power is the sum of the component powers. Note that the algorithmic results of Section 3 are quite general, as far as the choice of the combining functions f_x, f_y is concerned. In fact, they apply as long as these functions are monotone, efficiently computable, and relatively smooth (e.g. if $|f(x) - f(y)| \leq c \cdot \|x - y\|$ for an appropriate constant $c > 1$), which is practically often the case.

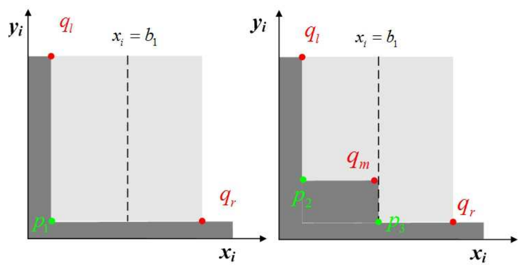


Figure 4: Geometric interpretation of CAPS algorithm (for one component): the ideal oracle guarantees that no solution points exist in the dark-shaded region.

3. THE ALGORITHM

Let us start with some basic notation. Recall that n denotes the number of components. Fix component C_i, $i = 1, 2, \dots, n$. We use $x^{(i)}_{min}$, $y^{(i)}_{min}$, $x^{(i)}_{max}$, $y^{(i)}_{max}$ to denote the minimum and maximum values of the coordinates for component C_i. We define the maximum ratio between the objective values to be $\alpha_i = x^{(i)}_{max}/x^{(i)}_{min}$. Intuitively, α_i represents the "range" of the x-values in the objective space for C_i. Let $\alpha = \max_{i=1,\dots,n} \alpha_i$ be the maximum ratio over all components. It is clear that $\alpha > 1$; in practice, it is bounded by a small constant. To avoid clutter in the relevant expressions, we will henceforth assume that $\alpha = O(1)$.

Our first result is that there exists an algorithm that can compute an ϵ-Pareto set for a system of n components, whose number of queries is *independent* of the size of the exact Pareto set (which could be arbitrarily large). [1] In particular, the number of queries depends only on the number of components n, their range α and the desired accuracy ϵ.

THEOREM 3.1. *There is an algorithm to construct an ϵ-Pareto set for a system of n components with $O((n/\epsilon) \cdot \log(\alpha))$ oracle queries.*

We stress that the above theorem is non-trivial in the sense that it provides an upper bound that does not depend on the size of the exact Pareto set. Of course, it is not enough; the underlying algorithm is non-adaptive, and our underlying goal was to compute an ϵ-Pareto set *as efficiently as possible* (i.e. with as few queries as possible). Formally, we consider the following problems:

Primal Problem. Given an error tolerance ϵ, compute an ϵ-Pareto set for the system using as few queries to the oracle as possible. We denote by k^* the optimal number of queries.

Dual Problem. Given a budget k on the number of queries, make k queries to the oracle and compute an ϵ-Pareto set for the system for the minimum possible value of ϵ. We denote by ϵ^* the optimal error attainable with k queries.

We remark that these two problems have been considered in the context of bi-objective (combinatorial) optimization problems [8, 9, 10, 22]. The main difference between these works and our setting lies in the compositional aspect of the current paper (roughly, the aforementioned results correspond to the case of one component.) Moreover, even for a single-component setting, our oracle is different (in fact, weaker) than the corresponding oracles used in these works. (This is dictated by our underlying application, since we used a commercial tool to implement the oracle.) In particular, in addition to our oracle, those related works use a symmetric oracle (which allows also to minimize x subject to a bound on y). This makes a big difference for the performance guarantees attainable in the two settings.

[1]The theorem proofs are removed due to lack of space and can be found in [16].

Algorithm 1 CAPS

```
Input: Oracle for the components' Restricted Problem
Output: System Pareto set approximation
1: $(b_{min}, b_{max}) \leftarrow$ the (min, max) values of the $x$–objective
2: for $i = 1, 2, \dots, n$ do
3:     $q_r^{(i)} \leftarrow$ Restrict$^{(i)}(y, x \le b_{max})$
4:     $q_l^{(i)} \leftarrow$ Restrict$^{(i)}(y, x \le b_{min})$
5: $Q = (q_1, \dots, q_m) \leftarrow$ combine $q_r$'s and $q_l$'s by $f_x, f_y$
6: sort $Q$ by increasing x–coordinates, i.e. from left to right
7: while Error $\ge \epsilon$ and queries $< k$ do
8:     for all interval $(q_i, q_{i+1})$ do
9:         compute $\mathrm{Error}_i$
10:    $I \leftarrow (q_j, q_{j+1})$ s.t. $j = \mathrm{argmax}_{i=1,\dots,m}\mathrm{Error}_i$
11:    $C_l \leftarrow$ component w/ max potential error reduction in $I$
12:    $(q_k^{(l)}, q_{k+1}^{(l)}) \leftarrow$ the relevant interval of $C_l$
13:    $b_m \leftarrow (x(q_k^{(l)}) + x(q_{k+1}^{(l)}))/2$
14:    $q_m^{(l)} \leftarrow$ Restrict$^{(l)}(y_l, x_l \le b_m)$
15:    $q_{sys} \leftarrow$ derive a new system point using $f_x, f_y$, considering $q_m^{(l)}$
16:    add $q_{sys}$ into $Q$, sorted by increasing x–coordinates
```

Next, we describe CAPS, an *adaptive "online" algorithm* to solve both the primal and the dual problems. Our algorithm is "online" in the following sense: it discovers the objective space by querying the oracle appropriately and uses the returned solutions (points) to compute an ϵ-Pareto set. Our algorithm is iterative. In every step, it maintains an *upper approximation* and a *lower approximation* to the Pareto set. As the number of iterations increases, these two approximations become finer and finer, hence the error decreases.

For the sake of intuition, we first describe the algorithm CAPS for the (very special) case that the system comprises of one component and we have oracle access to solve the Restricted Problem for the component. In this case, we proceed as follows: We start by computing the extreme points $q_l = (x(q_l), y(q_l))$ (leftmost), $q_r = (x(q_r), y(q_r))$ (rightmost) of the Pareto set (see Fig. 4). Each such point can be computed using appropriate queries to the oracle. Consider the point $p_1 = (x(q_l), y(q_r))$. CAPS *knows* that the exact Pareto set lies entirely to the right of q_l and above q_r (see Fig. 4). Consider the triangle $\triangle(q_l p_1 q_r)$. By definition, the error of the initial approximation $\{q_l, q_r\}$ is at most $\mathcal{RD}(p_1, \{q_l, q_r\})$. That is, $\{q_l, q_r\}$ is the current *upper* approximation to the Pareto set, while p_1 is the current *lower* approximation. If $\mathcal{RD}(p_1, \{q_l, q_r\}) \le \epsilon$, then CAPS terminates and returns $\{q_l, q_r\}$. Otherwise, we call the oracle for b_1 being the midpoint of q_l, q_r, i.e. $b_1 = (x(q_l) + x(q_r))/2$. Let q_m be the obtained point. The right part of Fig. 4 shows the information we obtain about the space after q_m is returned. In particular, this implies that the error in the interval $\{q_l, q_m\}$ is at most $\mathcal{RD}(p_2, \{q_l, q_m\})$, while in the interval $\{q_m, q_r\}$ is at most $\mathcal{RD}(p_3, \{q_m, q_r\})$. The question that now arises is which interval to update next. CAPS uses a greedy criterion: it selects to update the interval in which the error is maximum. Similarly to the first stage, it divides the chosen interval in half and queries the oracle appropriately. As shown in Theorem 3.2 below, CAPS finds an ϵ-Pareto set with at most $O(\log(1/\epsilon)) \cdot k^*$ queries to the Restricted routine, which is optimal.

Now consider a system comprising of many components. If we had access to the oracle to solve the Restricted Problem *for the system*, then it would be possible to implement the above described algorithm and obtain the desired approximation. The main difficulty arises from the fact that we only have access to the oracle to solve the Restricted Problem *for the individual components.* However, we can overcome this difficulty as follows: at every step we identify the interval *at the system-level* where the approximation error is maximum. To achieve this, we first combine all component-level points to derive system-level Pareto points by f_x and f_y, and then

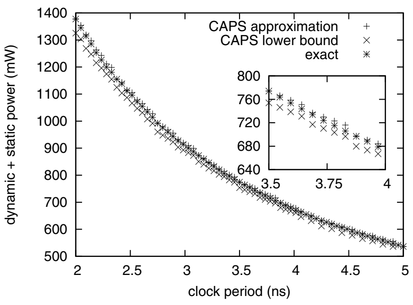


Figure 5: Power/performance trade-off of StereoDepth at $90nm$.

compute for each system-level interval its corresponding approximation error by ratio distances. At this point, another difficulty arises. Since we can afford only one query at the given step, we need to select which component to update for the selected system-level interval. In fact, there are several ways to update an interval at the system-level (potentially as many as the number of components). We choose to update the component that can potentially reduce the system error as much as possible. This requires the application of f_x, f_y, and ratio distance for each component to estimate a potential system-level error reduction due to that component. Once we determine the component to update and the relevant interval in that component, we update that interval by dividing into halves as above. This step sends a query to the oracle and updates the selected component's points. Finally, we add (still by f_x and f_y) a new system-level point according to the component update, and then proceed to the next system-level interval for error reduction. The iteration continues until a termination condition is satisfied. The pseudocode for CAPS, given as Algorithm 1, is basically the same for both problems. The only difference is the termination condition: for the Primal Problem, it terminates when the error falls below ϵ; for the Dual Problem, it terminates after making k queries.

The algorithm satisfies the following properties:

THEOREM 3.2. *(performance guarantee for primal problem) Algorithm* CAPS *finds an ϵ-Pareto set for a system of n components using at most $O(k^* \cdot n \cdot \log(1/\epsilon))$ queries to the oracle. Further, the dependence of the performance ratio on both n and ϵ is necessary.*

THEOREM 3.3. *(performance guarantee for dual problem) Algorithm* CAPS *finds an ϵ^*-Pareto set for a system of n components using at most $O(k \cdot n \cdot \log(1/\epsilon^*))$ queries to the oracle. Further, the dependence of the performance ratio on both n and ϵ^* is necessary.*

4. EXPERIMENTAL RESULTS

In this section we experimentally verify: *(i)* the accuracy of the approximated Pareto set produced by CAPS , and *(ii)* the convergence time of CAPS (or, dually, the best approximation within the time budget). We show that *besides having theoretical guarantees on its worst-case performance, in practice* CAPS *can return results that are more accurate in a shorter time, even in case of a non ideal (i.e., noisy) oracle*.

Experimental Set-Up. We implemented the CAPS algorithm and applied it to three synthesizable RTL designs: (a) *StereoDepth*, an SoC for video-rate stereo depth measurement that consists of 4 blocks [7]; (b) an *MPEG2 Encoder* consisting of 8 blocks; and (c) *MinSoC*, which consists of 8 blocks and was designed by starting from an *OR1200* processor-based SoC [1] and adding two more cores compatible with the Wishbone standard (a *DVI/VGA Video Controller* and an *SD Card Controller*).

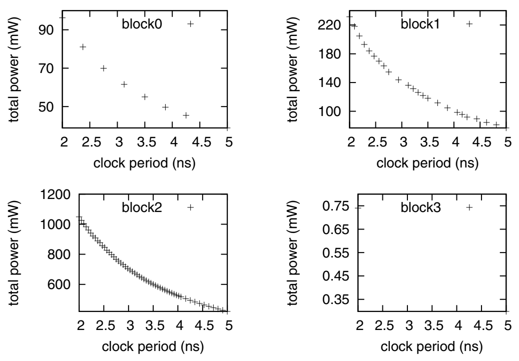


Figure 6: Component sampling of StereoDepth at $90nm$ to build the system approximation of Fig 5.

To play the role of the *oracle* of Fig. 1 we chose a widely-adopted commercial tool for logic synthesis equipped with two industrial standard-cell libraries developed for two technology processes ($90nm$ and $45nm$). We define performance as the maximum clock frequency at which the design can run. This is the inverse of the *target clock period* T_{clk} that is given to the oracle as the controllable constraint.

Power/Performance Tradeoffs. Fig. 5 shows the final approximation obtained by running CAPS to solve the Primal Problem (with error tolerance $\epsilon = 3\%$) to characterize the power/performance trade-offs of StereoDepth with the $90nm$ standard-cell library. As the clock period varies between $2ns$ and $5ns$ the power dissipation varies between $550mW$ and $1400mW$. The points of these curves are computed iteratively through a sequence of 65 oracle queries which required an aggregate 63 hours until the target $\epsilon = 3\%$ is matched. This is just 25% of the number of queries (and about 50% of the CPU time) necessary to compute the exact Pareto set, whose curve lays between the approximation and lower bound curves as we expected (see Fig. 5). Now, the exact Pareto set can generally be obtained by executing a *long* sequence of synthesis runs (corresponding to a series of equally-spaced values of T_{clk}) *for each component*. Instead, CAPS dynamically adapts the distribution of the queries to the portions of the design space that actually need to be explored to improve the approximation accuracy. For instance, as shown in Fig. 6, CAPS repeatedly invokes the oracle on *block2* and, to lesser extents, on *block0* and *block1*, while avoiding *block3* after the initial queries necessary to establish its extreme points.

Fig. 7 shows the evolution of the *guaranteed error* ϵ (i.e. guaranteed by Theorem 3.2) for each new query that we obtained by running CAPS to solve the Primal Problem (with error tolerance $\epsilon = 3\%$) to characterize the power/performance trade-offs of the MPEG2 Encoder with the $45nm$ library. As expected, the error decreases monotonically over time, until it reaches the target approximation. For these experiments we also computed the exact Pareto set. Hence, we can evaluate the *actual approximation error* ϵ_a, i.e. the ratio distance between the exact and the approximated Pareto sets. In practice, as shown in Fig. 7, after each query CAPS returns a value of ϵ_a that is much lower than the guaranteed error ϵ.

Table 1 shows the convergence results for each design. Columns "Exhaustive Search" combines pre-computed component curves to derive an exact system curve. Given ϵ, "CAPS Primal" finds an ϵ-Pareto curve for the system in k queries. Given k, "CAPS Dual" finds a system Pareto curve with guaranteed error $\leq \epsilon$. For all designs, CAPS Primal requires only 7–19% queries and 8–39% CPU time of Exhaustive Search to achieve a guaranteed 4% ϵ (and even less for a $\epsilon_a = 4\%$). On the other hand, CAPS Dual can achieve a guaranteed 2.4–6.0% ϵ (again, an even better ϵ_a) in 60 queries.

Design	Technology	Exhaustive Search ($\epsilon = 0\%$)		CAPS Primal ($\epsilon = 4\%$)				CAPS Dual ($k = 60$)		
				ϵ (guaranteed)		ϵ_a (actual)				
		k	CPU Time (hr)	k	CPU Time (hr)	k	CPU Time (hr)	ϵ	ϵ_a	CPU Time (hr)
StereoDepth	$90nm$	260	126	50 (19%)	49 (39%)	44 (17%)	44 (35%)	3.2%	3.1%	60 (48%)
MinSoC	$90nm$	520	27	96 (18%)	9 (33%)	87 (17%)	8 (30%)	6.0%	4.7%	6 (22%)
MinSoC	$45nm$	520	29	37 (7%)	4 (14%)	25 (5%)	3 (10%)	2.7%	2.0%	7 (24%)
MPEG2 Encoder	$45nm$	520	1824	35 (7%)	142 (8%)	21 (4%)	84 (5%)	2.4%	1.5%	232 (13%)

Table 1: Power/Performance trade-offs by CAPS. The parenthesized value of "k" and "CPU Time" is a ratio over *Exhaustive Search*.

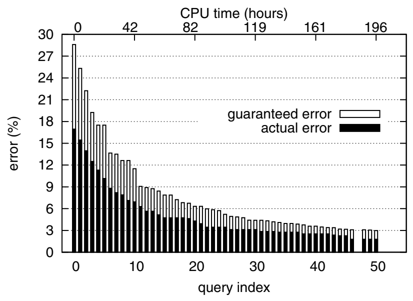


Figure 7: Error convergence of MPEG2 Encoder by query index (lower axis) and CPU time (upper axis).

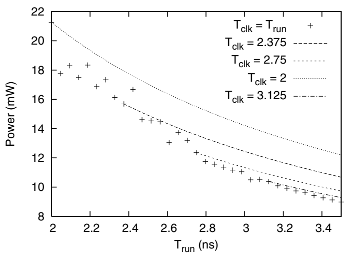


Figure 8: Synthesis and power/performance characterization.

Importance of Synthesis-Based Tradeoff Characterization. As discussed in the Introduction, characterizing the power/performance efficiency of an IP component across multiple operating points is a critical aspect of design for reusability. While we limited our experiments to varying the target clock frequency T_{clk} for the nominal voltage supply, we could easily extend our approach to variations of the supply voltage and the use of alternative technology libraries. The reader, however, may wonder why it is not sufficient to synthesize just a single implementation for the smallest possible value of T_{clk} at *design time*, and then simply reuse it also with higher values of the *operative clock period* T_{run} across various designs at *run time*. After all, this is what is typical done when applying dynamic clock frequency scaling methods for low power. The answer is given by Fig. 8 which, for the case of the *OR1200* processor core of *MinSoC*, shows that power not only scales down with the *running* frequency of the clock, but also with the clock frequency targeted during synthesis, i.e. the inverse of T_{clk}. Each of the individual points in Fig. 8 represents the estimated power consumption returned by the oracle for a given T_{clk}, i.e. corresponding to a distinct circuit implementations of the processor that can be chosen at design time. The four lines, instead, show the power trend for four of these implementations corresponding to scaling T_{run}. Notice, that the points are always below the lines, meaning that a circuit synthesized for the T_{run} at which it will operate never wastes power with respect to another circuit that was over-optimized for T_{clk} and then clocked for $T_{run} > T_{clk}$. Also, note that every point that is not below the lines is actually a dominated point in the Pareto set, which CAPS automatically eliminates. Given the importance of design for reusability and the growing trends of designing multi-core SoCs with multiple clock domains, the accurate synthesis-driven characterization of the power/performance of an IP core is essential to demonstrate that its design offers the best performance across a large dynamic range of power and frequency.

Area/Performance Tradeoffs (and Noisy Oracle Behavior). We analyze also the area/performance tradeoff because it allows us to evaluate the CAPS algorithm when the oracle manifests a noisy behavior. The implication of having a noisy oracle is that it can return a point in the area/performance plane that is not part of the Pareto set (see Fig. 9), with a probability that is not negligible. In particular, two scenarios may arise:

• *The returned point dominates* other previously-returned points. In this case, CAPS treats the query as valid and progress regularly. The only difference is that after this query the current approximation error might have temporarily increased, thus violating monotonicity. In fact, during the previous steps, the approximation of the Pareto set of the system (and, particularly, the lower-bound point that dictates the error) was based on some value which cannot be considered valid anymore because it became dominated by the new acquired point.

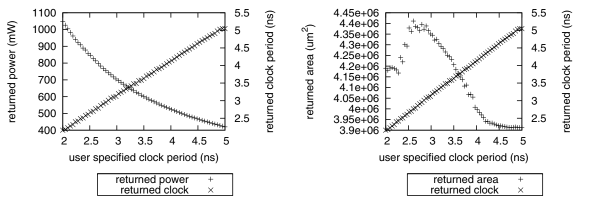
• *The returned point is dominated* by the current Pareto set. Not only is the query wasted, since it does not reduce the current approximation error of the system Pareto set, but also the portion of the space that has been just queried will likely remain the source of the biggest error (unless there is another portions that has exactly the same error) and CAPS will re-attempt the same query.

In order to overcome the source of deadlock due to the second scenario we introduced a practical heuristic that drives the selection of the following query: when a query returns a dominated point, CAPS performs a second query in the middle of the right semi-space, where the clock period is longer. The rationale behind this decision is: if the returned point is dominated, then the noise of the oracle is a consequence of very limited room for exploration in the left semi-space, i.e., the considered clock-period increase from the faster design which we already sampled (left-hand bound) is not enough to allow the tool a significant area reduction. Hence, it is likely that the space that deserves to be explored lies on the right semi-space, where the clock period increase is even bigger. However, if this exploration keeps returning a dominated point, we turn to explore the left semi-space. Finally, if a dominated point is still encountered, we search in other components.

Fig. 10(a) shows that even when CAPS has to rely on a noisy oracle, the approximation of the Pareto set converges and the target error ϵ is achieved. The missing bars represent queries that returned a dominated point. Even though the oracle is very noisy for at least one component, the returned point is dominated only in two occasions and this happens mostly at the beginning, where the space to be researched is bigger. After just a few queries, the regions of the space where a trade-off is meaningful will keep being queried, while CAPS mitigates the noise of the oracle for the other regions. Finally Fig. 10(b) shows the comparison between the estimated and the reference Pareto sets, for the area of StereoDepth at $90nm$.

5. RELATED WORK

Multiobjective Optimization is an area at the interface of operations research and computer science that has been being exten-



(a) Power/performance (b) Area/performance

Figure 9: Comparing the oracle behavior in terms of: (a) returned power and (b) area when synthesizing the *block2* component of StereoDepth at $90nm$ for various T_{clk} values. The diagonal line shows the returned clock period.

sively investigated in the literature (e.g., see [12]). While our model is very similar to the standard multiobjective framework defined in [18] (and further studied in [8, 9, 10, 22]) our algorithmic results are novel and are not implied by these works.

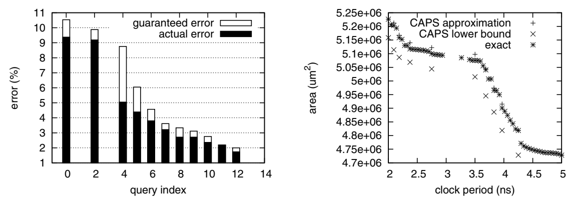
The problem size of design space exploration (DSE) grows exponentially with the number of design parameters. To tackle DSE problems efficiently, existing approaches evolve into four main categories. The first category aims at solution space pruning to accelerate DSE, e.g. by clustering parameters into super sets to handle a smaller number of combinations among the sets than the parameters [13, 19]. These approaches, however, depend on correct parameter dependencies, which are generally not available in advance. Secondly, randomized search methods have been proposed for DSE because they can flexibly encode non-linear problem instances as well as multiple objectives. Instances include Simulated Annealing [19] and Genetic Algorithms [11, 21]. Nevertheless, the local search algorithms are heuristics, which cannot guarantee solution qualities, and usually require long runtime. The third category is based on *estimating* metrics of interest under given configurations, as opposed to actually running time-consuming simulation or synthesis. Example estimators include Fuzzy Systems [2], Markov Decision Process [4], and response surface modeling [17]. These approaches cannot work alone and should be incorporated with optimization frameworks. Finally, the last category includes techniques to generate *representative* Pareto sets, e.g. by transforming convex multi-objective problems into a sequence of single-objective ones, which are gradually solved to uniformly sample the solution space [20] or by using a bottom-up approach that relies on the a-priori knowledge of the configuration-performance mapping function [5]. This is fundamentally different from our top-down approach which is more general and more efficient (as it requires less component queries).

Notice that most previous works optimize the target design as a whole entity [3], but this joint-optimization approach has limited scalability with complex SoC designs. While some works have system-composition properties [5, 11], the CAPS approach is uniquely efficient in sampling components only when is required.

6. CONCLUSIONS

We have presented CAPS an adaptive algorithm for system-level design characterization by compositional approximation of Pareto sets. The algorithm is provably the best online algorithm, and empirically works even better than what is guaranteed by the theorems. The application to power/area-performance design space exploration shows that our approach is both effective and efficient.

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.



(a) Error (b) Area vs. Clock Period

Figure 10: Evolution of guaranteed approximation and actual error with the queries for StereoDepth at $90nm$.

7. REFERENCES

[1] Opencores. www.opencores.org/.

[2] G. Ascia, V. Catania, A. G. D. Nuovo, M. Palesi, and D. Patti. Efficient design space exploration for application specific systems-on-a-chip. *Journal of Systems Architecture*, 53(10):733–750, Oct. 2007.

[3] O. Azizi, A. Mahesri, J. P. Stevenson, S. J. Patel, and M. Horowitz. An integrated framework for joint design space exploration of microarchitecture and circuits. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 250–255, Mar. 2010.

[4] G. Beltrame, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(7):1083–1095, July 2010.

[5] U. Bordoloi, H. P. Huynh, S. Chakraborty, and T. Mitra. Evaluating design trade-offs in customizable processors. In *Proc. of the Design Automation Conference (DAC)*, pages 244 –249, July 2009.

[6] S. Borkar. Design perspectives on 22nm CMOS and beyond. In *Proc. of the Design Automation Conference (DAC)*, pages 93–94, July 2009.

[7] A. Darabiha, J. Rose, and W. J. MacLean. Video-rate stereo depth measurement on programmable hardware. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 203–210, June 2003.

[8] I. Diakonikolas. *Approximation of Multiobjective Optimization Problems*. PhD thesis, Columbia University, 2010.

[9] I. Diakonikolas and M. Yannakakis. Succinct Approximate Convex Pareto Curves. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorthims (SODA)*, pages 74–83, Jan. 2008.

[10] I. Diakonikolas and M. Yannakakis. Small approximate Pareto sets for bi-objective shortest paths and other problems. *SIAM Journal on Computing*, 39:1340–1371, 2009.

[11] T. Eeckelaert, T. McConaghy, and G. Gielen. Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 1070–1075, Mar. 2005.

[12] M. Ehrgott. *Multicriteria optimization*. Springer-Verlag, 2005.

[13] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip. *IEEE Trans. on Very Large Scale Integration Systems*, 10(4):416–422, Aug. 2002.

[14] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. Scaling, power, and the future of CMOS. In *IEEE International Electron Devices Meeting*, pages 9–15, Dec. 2005.

[15] ITRS. The International Technology Roadmap for Semiconductors. *Available at http://public.itrs.net*.

[16] H.-Y. Liu, I. Diakonikolas, M. Petracca, and L. P. Carloni. An optimal online algorithm for compositional approximation of pareto sets. Technical Report CUCS-014-11, Dept. of Computer Science, Columbia University, 2011.

[17] G. Palermo, C. Silvano, and V. Zaccaria. Respir: A response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1816 –1829, Dec. 2009.

[18] C. Papadimitriou and M. Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *Proc. 41st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 86–92, Nov. 2000.

[19] B. C. Schafer and K. Wakabayashi. Design space exploration acceleration through operation clustering. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(1):153–157, Jan. 2010.

[20] A. Singhee and P. Castalino. Pareto sampling: choosing the right weights by derivative pursuit. In *Proc. of the Design Automation Conference (DAC)*, pages 913–916, July 2010.

[21] S. Tiwary, P. Tiwary, and R. Rutenbar. Generation of yield-aware pareto surfaces for hierarchical circuit design space exploration. In *Proc. of the Design Automation Conference (DAC)*, pages 31–36, July 2006.

[22] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356, 2005.