High-Level Synthesis of Approximate Hardware under Joint Precision and Voltage Scaling

Seogoo Lee, Lizy K. John, and Andreas Gerstlauer
Department of Electrical and Computer Engineering, The University of Texas at Austin
{sglee, ljohn, gerstl}@utexas.edu

Abstract— In recent years, approximate computing has emerged as a promising approach to trade off quality of computed outputs for energy savings. In this paper, we present an approximate high-level synthesis (AHLS) approach that outputs a quality-energy optimized register-transfer-level implementation from an accurate high-level C description. Existing AHLS work only considers switching activity for energy savings under hardware approximations. By contrast, we aim to provide a general AHLS solution that also considers voltage scaling given a reduced processing time. To maximize voltage and associated energy reductions, we include both operation-level approximations by bit rounding and more aggressive operation eliminations as approximation techniques. Optimally exploiting scaling opportunities under such approximations requires tight interaction with scheduling tasks. We address this problem by combining an optimization pass that estimates the scheduling impact of approximations with fast yet accurate quality-energy models and an efficient optimization solver to find near-optimal solutions constructively. Results show that when considering voltage scaling, up to 24.5% higher energy savings can be achieved compared to approaches that only consider switching activity. Our heuristic solver is able to find solutions within 0.1% of average energy savings compared to an exhaustive search, all while being up to 1,400× faster than simulation-based methods.

I. Introduction

Approximate computing (AC) introduces quality as a new design metric to be traded off for lower energy consumption or better performance. At the custom hardware level, approximations are achieved by reducing the precision of individual datapath operations at the expense of accuracy in final outputs. While many ad-hoc designs have been proposed for individual applications, large design spaces and non-obvious tradeoffs require systematic and effective approaches to automatically derive quality-energy optimal implementations.

In this paper, we address the problem of approximate high-level synthesis (AHLS). Our AHLS tool is concerned with synthesizing a quality-energy optimized register-transfer level (RTL) implementation of a high-level C description under hardware approximations. For approximations at the hardware level, both voltage and precision scaling approaches have been explored. However, performing each indepedently is suboptimal. Pure voltage over-scaling leads to timing violations, which can cause unpredictable, early, and catastrophic errors [8]. By contrast, energy savings in precision scaling stem from logic simplifications, which reduce switching activity. However, they also reduce logic delay, which can be exploited to scale voltages without causing timing violations. Due to the quadratic relationship of voltage to power, this often has a higher impact on energy savings, but many existing approaches ignore this aspect.

Some work [10, 6] includes voltage scaling as a secondary consideration after precision scaling in a non-HLS context. This, however, ignores their inherent and non-obvious interactions. Under uniform voltage and iso-performance assumptions, scaling opportunities arise from reductions in critical path delay or clock latency, i.e. total processing time. Especially the latter can have a large impact, but requires rescheduling coupled with scaling of precision or potentially more aggressive source-level optimizations to enable chaining or complete elimination of operations and clock cycles. Likewise, for a given quality goal, optimal precision scaling to minimize the critical path and thus maximize slack across different operations and clock cycles requires a similarly tight interaction with scheduling.

We formulate this problem in an HLS context as an additional quality-energy optimization pass that uses mobility information from an initial one-time scheduling to estimate the latency reduction and voltage scaling impact of precision scaling. We then pass the precision-optimized solution to existing synthesis task, followed by final post-processing optimizations that further re-balance operation precisions to minimize the critical path in the scheduled design. Our optimization flow is aided by a fast yet accurate, semi-analytical quality and energy estimation pass that determines overall quality degradations and energy savings in a control data flow graph (CDFG) under both bit rounding and more aggressive operation elimination approximations. Furthermore, we use a statistical approach to capture input-dependence of quality using one-time simulation only, and our latency and voltage estimation requires only one final re-scheduling. Finally, we propose a novel constructive optimization heuristic that is effective in finding near-optimal designs in a short runtime. We integrate our optimizations into an existing HLS framework to provide a comprehensive AHLS solution supporting general C input descriptions under joint precision and voltage scaling.

A. Related Work

There exists only limited work on AC in HLS flows. The authors in [7] propose an integer linear programming (ILP) formulation with a statistical precision scaling model integrated into traditional scheduling and binding tasks. However, the use of linear quality and energy models limits consideration of general application and hardware behavior. The approach only supports dataflow graphs without control flow, and does not consider voltage scaling.

Other approaches apply hardware approximations in the form of independent pre- or post-synthesis tasks [3, 6, 13]. At the source level, the authors in [3] combine simulation and analysis for quality estimation with parameterized ALU models using gate-level synthesis for energy estimation. Recent work [6] presents analytical quality and energy modeling techniques that avoid simulation and synthesis overhead, including an energy cost function that considers voltage scaling. These approaches use generic meta-heuristics for solving the optimization problem, which suffer from complexity and optimality issues. Such generic heuristic solvers have an issue that can easily fall into local minima. The work in [13, 10]

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

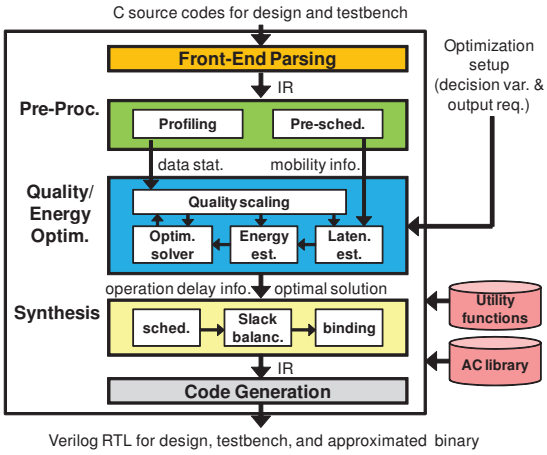


Fig. 1. Overview of our high-level synthesis (HLS) flow.

performs post-processing optimizations of an existing RTL description. They identify approximations at the logic level through iterative simulation and synthesis of structural [13] or behavioral [10] HDL code. All these approaches assume that a baseline design is given. As such, they do not perform true AHLS and ignore interactions with scheduling and binding.

B. Overview

Fig. 1 shows our proposed AHLS flow. Inputs are the C source code of the accurate design, its testbench, a quality-energy configuration including decision variables defining what data and operations to approximate, and a quality requirement for computed results. The output is an approximate RTL implementation that minimizes energy consumption while meeting the given quality constraint.

Area and delay values of approximate hardware units under different approximation levels are given in an AC library. We support precision reduction by rounding as hardware approximation mechanism. We also utilize more aggressive operation eliminations, which can be considered an extreme form of rounding that removes all data bits and completely eliminates operations. We currently consider AC libraries and applications with adders and multipliers only. Extension to other operations will be targeted in future work.

Our tool first converts the given C code into an intermediate representation (IR) using standard front-end optimizations. In a following pre-processing step, we perform simulation and pre-scheduling to collect data statistics and mobility information for all operations. The actual quality-energy optimization is performed by our heuristic solver. It uses a quality scaling model together with latency and energy estimates determined using the mobility and data statistics collected during the pre-processing step. The precision-optimized solution with updated operation delays is then passed to the scheduler, which finds a latency-minimized design under given resource constraints. A post-synthesis optimization step further rebalances precision and timing slack across clock cycles to minimize the critical path and maximize voltage scaling. Finally, the best solution is passed to the binding step, and the final Verilog RTL code and testbench are generated.

II. Pre-Processing

Using a given testbench, we first run a C simulation of the accurate design to obtain statistical information about the application data under user-provided stimuli. We insert a profiling function at each input and output of all intermediate operations to capture values of associated variables and calculate their means μ and variances σ^2 later used in our quality estimation. We similarly profile branch probabilities.

We further perform an ASAP and ALAP pre-scheduling of the accurate design to obtain mobility information for each operation. We run both ASAP and ALAP with user-provided resource constraints on the number of arithmetic operators and memory ports.

III. Quality-Energy Optimization

Quality and energy optimization is at the core of our AHLS tool. Formally, the optimization problem is to minimize energy cost under quality constraints:

$$\min_{\mathbf{s}} E(\mathbf{s}), \text{s.t.} \forall j : Q_j(\mathbf{s}) \leq Qr_j. \quad (1)$$

Here, $\mathbf{s}$ is a vector $(s_0, \ldots, s_{N-1})$ of decision variables s_i, and $Qr_j, 0 \leq j < M$ are the quality constraints at different computation outputs provided by the user. A decision variable s_i is the number of rounded bits at the i-th approximation point. By default, our tool automatically identifies approximation points to round off data for all inputs and all internal multiplier outputs, which affect the entire application and double the number of bits, respectively. The user can optionally specify the approximation points manually. As input to the tool, the user can define groups of approximation points in the C source code that will share the same number of rounding bits. Each group maps to a decision variable.

A. Quality Scaling

Given a candidate solution $\mathbf{s}$, we estimate its quality degradation $Q_j(\mathbf{s})$, and at the same time identify operations that can be eliminated from an accurate input design. Optimally identifying operations to eliminate is closely related to quality estimation since it depends strongly on data and error statistics. We determine operation elimination by regarding it as an extreme form of bit rounding: At each approximation point, if the number of rounding bits s at an approximation point makes the rounding error larger than the data to be rounded, not using the data gives a smaller error than rounding off s bits. In this case, the error becomes the data itself, and the data after the approximation becomes zero, which provides opportunities for further downstream operation eliminations.

For quality estimation, we employ a semi-analytical approach similar to [6], but our approach is different in three key aspects: we support control flow, we consider joint bit rounding and operation elimination, and we account for mutual dependences among individual errors.

Fig. 2 shows our quality estimation for an example of two multiplications and one addition having three approximation points, each mapped to a decision variable. We traverse a given input design's CDFG in a breadth-first order from the first operation. During this graph traversal, we run an error generation function at every approximation point, and an error propagation function at every operation. Each sub-function computes three values, error statistics (ER), data statistics (D), and the number of rounded, zero-valued bits (ZB). Error and data are represented by their signal power, i.e. $\mu^2 + \sigma^2$.

The error generation function is applied at the i-th approximation point where we round off data by s_i bits as summarized in Table I. We model a basic rounding error as an additive and uniformly-distributed error with a mean of zero and variance $\sigma^2_{\mathrm{R},s} = 1/12 \times 2^{2s}$ [6], but with two exceptions. For basic bit rounding, the output error $ER_{\mathrm{O},i}$ is $\sigma^2_{\mathrm{R},s_i}$, the

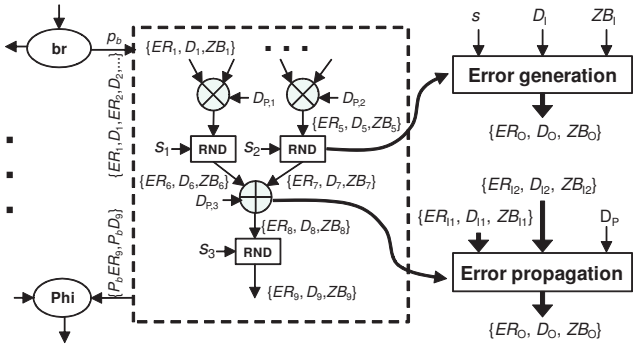


Fig. 2. Quality Estimation.

TABLE I: SUMMARY OF ERROR GENERATION.

Condition	ER_{O}	D_{O}	ZB_{O}
if $ZB_{\mathrm{I}} > s$	0	D_{I}	ZB_{I}
else if $\sigma^2_{\mathrm{R},s_i} > D_{\mathrm{I},i}$	D_{I}	0	bw
otherwise	$\sigma^2_{\mathrm{R},s}$	D_{I}	s

output data power $D_{\mathrm{O},i}$ remains unchanged, and the number of zero-valued lower bits $ZB_{\mathrm{O},i}$ is set to s_i. By contrast, if $ZB_{\mathrm{I},i}$ in the input is larger than s_i, all bits are already zeros. Thus, there is no additional error generated by rounding s_i bits, no change in the output data, and $ZB_{\mathrm{O},i}$ becomes $ZB_{\mathrm{I},i}$. Finally, if s_i is large enough to make $\sigma^2_{\mathrm{R},s_i} > D_{\mathrm{I},i}$, we approximate the data to zero and trigger elimination of subsequent operations, where the error becomes the input data itself, the output data becomes zero, and all the bits corresponding to the native bitwidth of the i-th operation output (bw_i) become zero-valued bits.

The error propagation function defines how generated errors propagate through adders and multipliers, and it identifies operation eliminations. For the k-th operation, we apply a basic mean and variance propagation method from [6] to compute $ER_{\mathrm{O},k}$. We also update $D_{\mathrm{O},k}$ and $ZB_{\mathrm{O},k}$ as shown in Table II. For $D_{\mathrm{O},k}$, the data statistics $D_{\mathrm{P},k}$ obtained during profiling of the k-th operation's output are normally propagated into the next operation. However, if either input is previously approximated to zero, the other input is propagated for an adder or the output becomes zero for a multiplier, and we mark this operation as eliminated.

In case of control flow, when the quality estimator encounters a Phi instruction matching branch b and using a result r from a previous basic block, it multiplies the result's error ER_r and data statistics D_r by the previously profiled branch probability p_b, and propagates these adjusted values to the successor. ZB_r values are passed through unchanged.

We use a signal-to-noise ratio (SNR) quality metric for all constraints Qr_j in our framework. For each constrained output j, we compute its SNR from the estimated data and error power as $Q_j = D_j/ER_j$.

B. Latency Estimation

Timing gains from approximations can contribute not only to reduced critical path delays, but, depending on scheduling, also to reductions in latency, i.e. the total clock cycle length. Both reductions can be exploited for voltage scaling and hence energy savings under a constant performance goal. Note that we apply uniform voltage scaling for the complete hardware block, and we do not scale voltage down to a level that causes timing violations. In general, not all approximations lead to a latency reduction in the scheduled design. We use mobility information from pre-processing, and operation elimination annotations from the quality scaling

TABLE II: SUMMARY OF ERROR PROPAGATION.

Operation	ER_{O}	D_{O}	ZB_{O}
Adder	$ER_{\mathrm{I1}} + ER_{\mathrm{I2}}$	D_{I2}, *if* $D_{\mathrm{I1}} = 0$ D_{I1}, *if* $D_{\mathrm{I2}} = 0$ D_{P}, *otherwise*	$\min(ZB_{\mathrm{I1}}, ZB_{\mathrm{I2}})$
Mult.	D_{P}, *if* $D_{\mathrm{I1}} = 0$ *and* $D_{\mathrm{I2}} = 0$ $ER_{\mathrm{I1}} D_{\mathrm{I2}} + ER_{\mathrm{I2}} D_{\mathrm{I1}}$, *otherwise*	0, *if* $D_{\mathrm{I1}} = 0$ *or* $D_{\mathrm{I2}} = 0$ D_{P}, *otherwise*	$ZB_{\mathrm{I1}} + ZB_{\mathrm{I2}}$

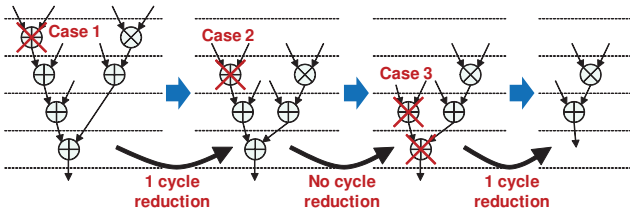


Fig. 3. Operation elimination and cycle reduction.

pass to estimate clock cycle reductions. This will in turn help drive the optimization solver towards solutions that eliminate operations on the latency-critical CDFG path.

Fig. 3 shows three examples of operation eliminations. Operations with zero mobility will not always reduce latency by one cycle when eliminated. For the first and third cases shown in Fig. 3, the elimination can reduce clock cycles by the eliminated operation itself (case 1) or its successor (case 3), both of which have zero mobility. By contrast, even though the eliminated operation has zero mobility, the second case does not contribute to a cycle reduction due to another operation scheduled in the same cycle. Our latency estimator accounts for all such cases. When there are multiple operations eliminated, we traverse the graph in breadth-first order to successively check the cycle reduction and update the mobility of the remaining operations.

C. Energy Model

Energy savings can come from both reduced switching activity and voltage. To estimate switching activity, we use an area-proportional model in units of 1-bit full adders similar to [5]. Estimation of voltage reductions requires accurately capturing the relationships between (1) an approximation and processing time $T(\mathbf{s})$, and (2) processing time and voltage $V(T)$. We obtain $T(\mathbf{s}) = d_{\mathrm{crit}}(\mathbf{s}) \times L(\mathbf{s})$ as the product of critical path delay across all clock cycles $d_{\mathrm{crit}}(\mathbf{s})$ and previously estimated latency $L(\mathbf{s})$ under approximations $\mathbf{s}$. To estimate $d_{\mathrm{crit}}(\mathbf{s})$, we capture the relationship between precision and delay of individual arithmetic units $d(s)$ from gate-level synthesis of all adders and multipliers under different approximation levels s. Finally, for $V(T)$, we adopt the approach from [6], where we run HSPICE simulations of all standard cells under different voltage levels, and then fit a quadratic function to model voltage V as a function of T. We capture these relationships in an AC library.

With this, our estimated energy cost function becomes $E(\mathbf{s}) = V^2_{\mathrm{ref}}/V(T(\mathbf{s}))^2 \times \sum_{j \in M} SW_j$. Here, V_{ref} is the nominal voltage, SW_j is the estimated switching activity of operation j from the AC library, and M is the total number of operations. During quality-energy optimization, we only consider the estimated latency in computing $T(\mathbf{s})$ and $E(\mathbf{s})$, and we use the critical path delay of the accurate design obtained from pre-scheduling. In general, pre-synthesis estimation of critical path delays under approximations is difficult in the presence of operation elimination and chaining. We instead consider the actual $d_{\mathrm{crit}}(\mathbf{s})$ during post-synthesis slack-balancing optimizations.

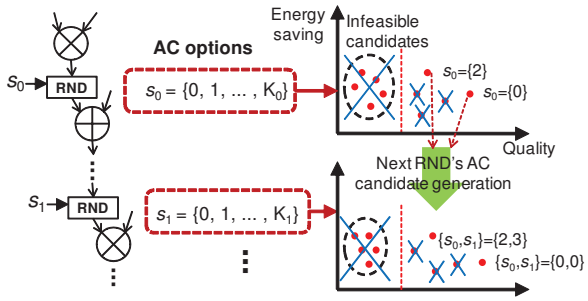


Fig. 4. Breadth-first search with early termination.

D. Optimization Solver

We introduce an efficient heuristic for solving the inherently non-linear quality-energy integer optimization problem in eq. (1). Existing general meta-heuristics, such as random greedy searches [10] or simulated annealers [6] have fundamental limitations on efficiency and optimality, and do not capture optimization-specific features.

Our heuristic solver is inspired by gate-level buffer insertion algorithms [11]. Starting from the decision variable that is associated with the first operation in the CDFG, it traverses the graph and greedily evaluates decision variables in a breadth-first manner and thus in order of dependencies as required by the quality model. (Fig. 4). When all possible solution candidates for one decision variable are examined, it prunes out infeasible candidates that violate given quality constraints. In addition, to further reduce the search space, the heuristic also prunes candidates that are dominated by other solutions, i.e. it only leaves the dominating candidates for evaluation at the next variable. After evaluation of the last decision variable, it selects the minimum energy solution as the final one. Although the heuristic can fall into local minima, our results show that it provides candidates that are very close to Pareto-frontiers from a full search, with a dramatically reduced search complexity.

Algorithm 1 details our optimization heuristic. The algorithm takes a CDFG G, a list of decision variables $DecVars$ and scheduling information Φ as input. It starts by initializing the set of feasible and dominating solution candidates $Cand$ with a single all-zero vector of decision variables representing an accurate design. It then processes decision variables in breadth-first search order. For each decision variable, the algorithm explores all possible values combined with every partial solution in the candidate set found so far. Each new candidate solution is evaluated for quality and energy, and checked for feasibility against given quality constraints. Finally, the algorithm checks whether the solution is dominated by any existing candidate, and added to the candidate set otherwise. After processing the last decision variable, the minimum energy solution is selected and returned.

While the heuristic itself searches for the single best solution, it also saves near-optimal candidates within user-specified energy and quality thresholds from the best one. We pass this near-optimal solution set to our final synthesis step as post-scheduler candidates for further slack balancing.

Algorithm 1 Quality-energy optimization

```
1: procedure OPTIMIZER(G, DecVars, Φ)
2:   Initialize Cand ← {(0, ..., 0)}
3:   for all i ∈ DecVars in G's BFS order do
4:     S ← Cand
5:     for all s ∈ S do
6:       for s_{i,min} ≤ v ≤ s_{i,max} do
7:         s(i) ← v
8:         q ← Q(G, s)
9:         e ← E(G, s, Φ)
10:        if ∃j : q_j < Qr_j then
11:          continue
12:        Dom ← false
13:        for all c ∈ Cand do
14:          if ∀j : q_j < Q_j(G, c) ∧ e > E(G, c) then
15:            Dom ← true
16:        if not Dom then
17:          Cand ← Cand ∪ {s}
   return arg min_{c∈Cand} E(G, c, Φ)
```

IV. Synthesis

We run final scheduling and binding passes for the best solution obtained from quality-energy optimization. We use pre-characterized $d(s)$ from the AC library to provide rounding-adjusted operation delays to the scheduler. Furthermore, using the scheduler output and near-optimal candidates collected during optimization, we apply a post-scheduling slack balancing optimization to determine if there exists any other candidate that has a smaller d_{crit} and corresponding lower energy. For each candidate $\mathbf{s}$, we use updated $d(s)$ to compute the maximum delay over all chained operation sequences in all clock cycles, and we choose the candidate with the minimum critical path delay as the final synthesis solution.

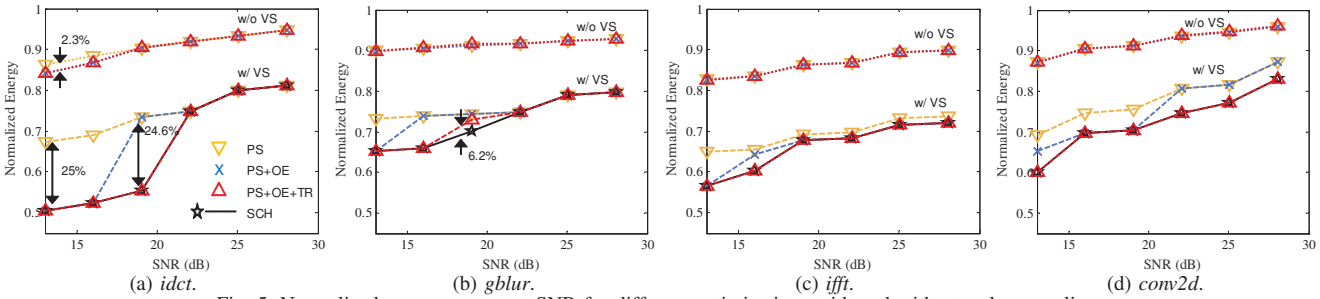
V. Experiments and Results

We have implemented our tool as additional optimization passes integrated into Legup [1], an open-source C-to-RTL HLS tool based on LLVM [4]. LegUp uses the scheduling algorithm in [2], which is built on a linear programming (LP) formulation. We used 4 different applications from [6] and the SD-VBS benchmark suite [12]: *idct* is a 1D inverse discrete cosine transform, *gblur* is a Gaussian filter, *ifft* is a 64-point fast Fourier transform, and *conv2d* is a 2D convolution. All examples are modified to use integer arithmetic. Data sets are from JPEG and the SD-VBS benchmark suite [12]. We collect near-optimal candidates for post-synthesis optimizations to be within 100% of the best quality and energy solution. Experiments are performed on an 2.67GHz Intel Core i7 machine using a Synopsys 32nm technology library.

A. Energy vs. Quality Tradeoffs

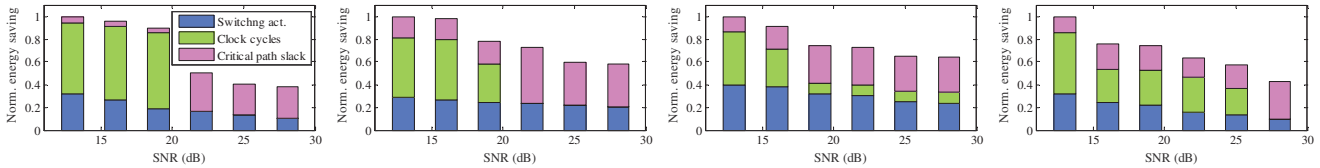
We first evaluate effectiveness of our quality-energy optimizations. Fig. 5 presents energy for different SNR constraints from 13 dB to 28 dB in 3 dB steps. Energy values are from our model and normalized against the accurate baseline design with different optimizations enabled, and with or without considering voltage scaling (VS) when computing final post-optimization energy savings. Evaluated optimization levels include precision scaling only (PS [6]), precision scaling with operation eliminations (PS+OE, similar to [9]), and both scaling and elimination while also optimizing for associated processing time, i.e. clock latency and critical path reductions (PS+OE+TR, our work). We compare results against a full search (SCH) that feeds all feasible solutions into the scheduler to find the one having the smallest energy under precision and voltage scaling.

Achievable energy savings depend on the application, target SNR, and baseline design quality. The largest overall and average gains over the evaluated SNR range are 50% and 30% in the *idct* example. Results show that the energy gains from voltage scaling are higher than savings due to reductions in switching activity. The average gains across all examples and SNRs with and without VS are 31% and 10%, respectively, giving an average of 21% extra savings from voltage scaling.



(a) *idct*. (b) *gblur*. (c) *ifft*. (d) *conv2d*.

Fig. 5. Normalized energy vs. target SNR for different optimizations with and without voltage scaling.



(a) *idct*. (b) *gblur*. (c) *ifft*. (d) *conv2d*.

Fig. 6. Decomposition of normalized energy savings for different target SNRs under all optimizations and with voltage scaling.

Comparing different optimizations, without VS there are little to no benefits beyond precision scaling alone. As expected, processing time reductions do not contribute to energy savings without voltage scaling. Even aggressive operation eliminations only provide small gains of up to 2.3% in the *gblur* example at 13 dB SNR. By contrast, our proposed optimizations are effective in maximizing energy savings when voltage scaling is considered. Compared to approaches that only consider precision scaling (PS) [6], operation eliminations contribute to up to 25% additional savings at low SNRs in all examples. The gains come from clock cycle reductions that have a higher impact on processing time and thus voltage savings than bit rounding alone. Note that the *ifft* has savings even at high SNRs. This is due to constant zero twiddle coefficients that give operation elimination opportunities regardless of SNR. When comparing PS+OE and PS+OE+TR, we see benefits of optimizing for latency and critical path delay reductions. The maximum difference is 24.6% at 19 dB in the *idct*. Overall, we observe that energy reductions using PS+OE+TR are almost the same as in SCH, i.e. our approach is able to find near-optimal solutions. The largest difference is 6.2% at 19 dB in *gblur*.

We further study how energy savings decompose into gains from reduced switching activity, clock cycles, and critical path delay with all optimizations (PS+OE+TR) and voltage scaling (VS) enabled (Fig. 6). Energy savings are normalized against the maximum savings at the lowest SNR level. In all cases, the gain from switching activity reductions is gradual and proportional to SNR requirements. Overall voltage scaling savings depend on the application itself, but in general, at high SNR levels gains mostly come from critical path reductions, while at low SNRs, they are from reduced clock cycles. Lower SNR requirements open more opportunities for operation elimination, which can result in more aggressive clock cycle reductions compared to bit rounding only. Bit rounding can also reduce clock cycles, but only when previously separated operations can now be chained into one clock cycle. However, such chaining opportunities are limited due to factors such as memory accesses and graph dependencies. Again, the *ifft* has clock cycle reductions even at high SNRs due to the constant zero twiddle coefficients.

From Figs. 5 and 6 we can observe the reasons for benefits of considering the scheduling impact and optimizing for latency and critical path reductions under voltage scaling. For *idct*, *gblur* and *ifft* examples, clock cycle gains start to appear or increase, and PS+OE and PS+OE+TR start to differ at 16 $\sim$ 19 dB. In this intermediate SNR range, there can exist multiple alternative approximation options. For example, there can be two solution candidates with different precision levels and two different operations out of which only one can be eliminated. If only one of these two operations contributes to a clock cycle reduction when eliminated and the other candidate has overall lower switching activity, a pure PS+OE approach will end up picking a globally non-optimal solution. At lower SNRs, both operations will end up being eliminated, and the difference disappears. At higher SNRs, neither operation can be eliminated and there are no differences between the approaches. For *conv2d*, differences due to latency optimizations exist in an even wider range from 13 dB to 25 dB. In addition, *conv2d* shows non-latency-related PS+OE+TR versus PS+OE gains already at 28 dB. These are due to benefits of our slack balancing optimizations.

B. Optimality

We further study Pareto-optimality of our heuristic solver. Fig. 7 shows the candidates evaluated by a full search (SCH) as compared these against the dominating Pareto set found by our heuristic. Results show that the PS+OE+TR solutions are very close to the true Pareto frontier. To quantify optimality, we use the metric from [14], which averages the normalized Euclidean distances between a solution in our set and its closest Pareto front. The table in Fig. 7 also shows the Pareto diversity as measured by the number of solutions in the Pareto set. Results are shown for all examples at the 28 dB target. Examples show no or only a small difference in distance of less than 0.02 using the metric in [14]. For diversity, results only differ by up to 7 fewer solutions (5.6%). Results confirm that our dominating set is close to or, in case of *idct* and *conv2d*, on the true Pareto front.

Final energy savings obtained from gate-level synthesis of the accurate and approximated designs are shown in Table III. Energy E of all designs is obtained by scaling dynamic power numbers P reported by Synopsys DesignCompiler for default switching activity at a nominal V_{dd} of 1.05V. Scaled operating voltage V and final energy E of each approximate design are determined using our $V(T)$ model. Designs are

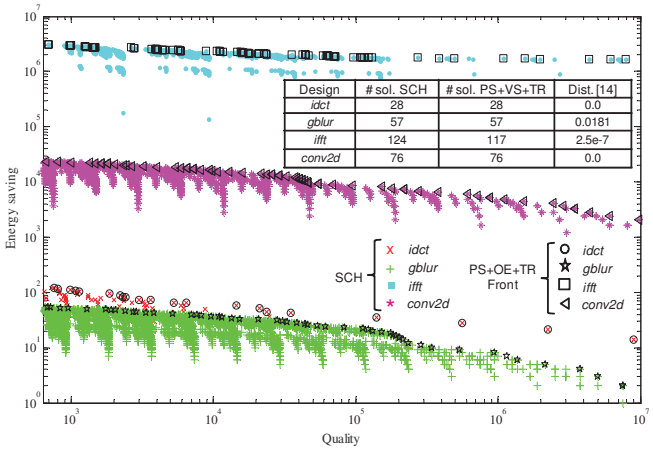


Fig. 7. Pareto-frontiers and Pareto optimality measures.

TABLE III: SYNTHESIS RESULTS.

Target SNR	Design	L [cyc]	P [mW]	V [V]	E [J]	SNR
∞	*idct*	12	6.70	1.05	123.8	-
	gblur	10	5.72	1.05	51.5	-
	ifft	128	10.98	1.05	14,238	-
	conv2d	29	3.66	1.05	874.2	-
28 dB	*idct*	12	6.43	1.04	118.77 (-4%)	32.7 dB
	gblur	10	5.17	1.00	46.5 (-9.6%)	25.5 dB
	ifft	124	10.46	0.94	10,138 (-28.8%)	30.1 dB
	conv2d	29	2.20	0.97	725.65 (-17%)	33.7 dB
13 dB	*idct*	5	0.31	0.81	5.81 (-95.3%)	16.2 dB
	gblur	8	1.76	0.87	15.85 (-69.2%)	14.4 dB
	ifft	92	3.03	0.83	3,926 (-72.4%)	15.2 dB
	conv2d	21	0.7	0.79	230.52 (-73.6%)	13.2 dB

scaled to have the same performance as the accurate one. All unscaled processing times are computed as the product of latency L achieved by the Legup scheduler and the critical path delay reported by DesignCompiler. Energy savings at 28 dB SNR range from 4% to 28.8%. At this SNR level, energy gains are mostly from bit rounding. They are smaller than in Fig. 5 due to bit rounding overhead not accounted for in our estimation. Since the *idct* has a relatively lower base signal power, less noise and only a smaller number of approximated bits can be introduced for a given SNR constraint. Using instead a more typical peak signal metric and as compared to a floating-point reference, our fixed-point *idct* has a PSNR of 48 dB at infinite SNR, and only a slightly lower PSNR of 40 dB at 28 dB SNR.

At lower SNR = 13 dB, real savings are larger than estimates in Fig. 5, ranging from 69.2% to 95.3%. This is due to upper sign bits not considered in our model. Since our model only accounts for switching activity of lower bits, gains of eliminated operations are relatively lower than in reality.

We perform quality validation using simulations of the approximated binaries generated by our tool. In all cases, estimated SNRs track the true ones obtained from simulations within a 6 dB range. SNR results for synthesized designs are shown in Table III. Remaining errors are due to independence and additiveness assumptions in our quality model.

C. Complexity and Runtime

Fig. 8 compares the complexity of our heuristic solver versus SCH that only prunes out infeasible candidates. We compare the number of candidates considered in the search as the key measure of search complexity and optimization time. We compare these against the size of the raw design space. Our heuristic evaluates up to 7× fewer candidates than SCH, and up to 252× fewer than the full design space.

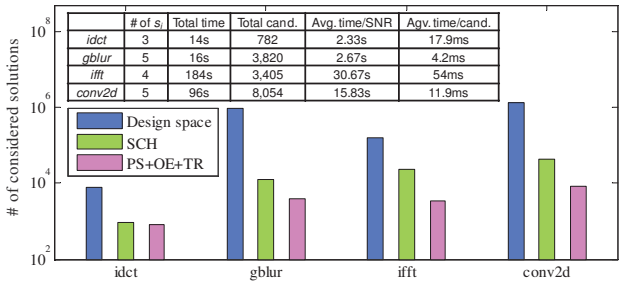


Fig. 8. Search complexity and runtime analysis of heuristic.

The table in Fig. 8 summarizes the runtime of our tool. Overall, runtimes range between 14s and 184s for exploration across all 6 SNR levels. This translates into 2.33s $\sim$ 30.67s per execution of the tool for a single quality constraint, and an average of 22ms per evaluation of one solution candidate. By contrast, the work in [10] requires 30s for simulation- and synthesis-based evaluation of every candidate. The work in [7] uses an ILP formulation, which is known to be exponential in complexity. Overall, this represents a significantly better efficiency of our approach.

VI. SUMMARY AND CONCLUSIONS

In this paper, we propose an approximate C-to-RTL high-level synthesis tool that jointly explores precision and voltage scaling to maximize energy savings under a given quality constraint. We apply a fast and accurate formulation of the quality-energy optimization problem that combines a semi-analytical, statistical quality model and an energy model considering savings in switching activity and scheduling impact of voltage scaling with an efficient and effective heuristic solver. Our tool can achieve near-optimal results with low runtimes, demonstrating energy savings of, on average, more than 77.6%. In the future, we plan to focus on improvements in complexity of the heuristic solver, and consideration of other forms of hardware/software approximations.

ACKNOWLEDGMENTS

This work is supported by an Intel grant.

REFERENCES

[1] A. Canis et al. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In *FPGA*, 2011.
[2] J. Cong and Z. Zhang. An efficient and versatile scheduling algorithm Based on SDC formulation. In *DAC*, 2006.
[3] J. Huang et al. A methodology for energy-quality tradeoff using imprecise hardware. In *DAC*, 2012.
[4] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *CGO*, 2004.
[5] D.-U. Lee et al. Accuracy-guaranteed bit-width optimization. *IEEE TCAD*, 25(10), Oct 2006.
[6] S. Lee et al. Statistical Quality Modeling of Approximate Hardware. In *ISQED*, 2016.
[7] C. Li et al. Joint precision optimization and high level synthesis for approximate computing. In *DAC*, 2015.
[8] J. Miao et al. Modeling and synthesis of quality-energy optimal approximate adders. In *ICCAD*, 2012.
[9] K. Nepal et al. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE*, 2014.
[10] K. Nepal et al. Automated high-level generation of low-power approximate computing circuits. *IEEE TETC*, (99), 2016.
[11] L. P. P. P. van Ginneken. Buffer placement in distributed rc-tree networks for minimal elmore delay. In *ISCAS*, 1990.
[12] S. K. Venkata et al. SD-VBS: The San Diego vision benchmark suite. In *IISWC*, 2009.
[13] S. Venkataramani et al. SALSA: Systematic logic synthesis of approximate circuits. In *DAC*, 2012.
[14] E. Zitzler et al. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE TEVC*, 7(2), April 2003.