Optimizer for Floating-Point Unit Generator

CS 229 Final Writeup

Jing PU, Mingyu GAO, Tai GUO

1 Introduction

Nowadays digital circuits become more and more complex, and keep requiring higher performance and lower cost. To design a digital circuit, such as a floating-point unit (FPU), usually needs to explore a huge design space to optimize speed, power and area. However, there exists a fundamental tradeoff between these three metrics, and usually people want to find the Pareto optimal configurations called the "design tradeoff curve".

Our project focuses on a FPU generator, *FPGen* [2], which integrates a set of configuration parameters and generates the corresponding FPU circuits. In order to find the Pareto optimal designs, currently FPGen just does brute-force sweeping in the whole design space, which usually contains more than 15 thousands design points. It costs a huge amount of time since evaluating each design point needs hours of synthesis.

The goal for our project is to build an optimizer for FPGen using machine learning algorithm to save design time significantly. Here we focus on the delay and energy tradeoff of the circuit. The optimizer is expected to predict the Pareto optimal configurations smartly based on a small set of synthesis data generated by sparse sampling on the design space. The accuracy of this predication should be close to that of the original approach.

2 Data Study and Acquisition

In the design procedure of a digital circuit, the designers need to make decisions on many design choices (which are usually called "features" in the Machine Learning terminology). While some of these features (e.g., the sizes of transistors) can be easily modeled as continuous variables, the others are discrete, including integer variables (e.g., the number of pipeline stages) and categorical variables (e.g., which topology or structure to use). While modern computer-aided design (CAD) tools can optimize many low-level features for us, we still need to make the decision for the high-level integer variables and categorical variables. The fact that the feature space has very high dimension and contains multiple types of components makes the modeling extremely difficult.

The features fall into three categories (see Table 1)

Device-level features $\zeta = (\zeta_1, \ldots, \zeta_s)$. These features are those that can be handled by the CAD tools. Thus we don't include them in our model, but just leave them to the tools.

Circuit-level features $\phi = (\phi_1, \ldots, \phi_l)$. These features can be modeled based on the circuit theory.

Architecture-level features $\theta = (\theta_1, \ldots, \theta_n)$. Changing these features often results in totally different structures. So they cannot be covered by the circuit model, and are very difficult to model.

For our data acquisition, we used FPGen from Stanford VLSI group [2] for hardware generation and design space exploration. We use 45nm bulk silicon technology from TSMC as our process

Table 1: Features

Category	Name	Range
Device-level	—	—
Circuit-level	`FPGen.FMA.PipelineDepth`	3,4,5,6,7,8,9,10,12
	`TOP_VT`	lvt,svt,hvt
	`TOP_Voltage`	0.8, 0.9, 1
Architecture-level	`FPGen.FMA.EnableMultiplePumping`	YES,NO
	`FPGen.FMA.MulShift.MUL0.BoothType`	1,2,3,4
	`FPGen.FMA.MulShift.MUL0.TreeType`	Wallace,ZM,OS2,Array

technology for mapping hardware to the physical designs. And then we evaluate the electric characteristics, such as the delay and the power consumption for each physical design using the CAD tool from Synopsys called Design Compiler.

In order to gain more confidence in the training and validation phases, we gathered as many data as possible in the limited amount of time. The design space we explored is the cross-product of a set of features. So far, we have obtained more than 15 thousands data points, each of which takes about 20 min $\sim$ 2 hours of CPU time.

3 Model

3.1 Mathematical Abstraction

Given all the design features (θ, ϕ, ζ), the delay D and the energy E of the circuit are

$$D = D(\theta, \phi, \zeta), \quad E = E(\theta, \phi, \zeta) \tag{1}$$

Our goal is under a certain requirement of delay $D \leq \widetilde{D}$, finding the the minimal energy E and the high-level design features (θ, ϕ) corresponding to it (or vice versa). The device-level features ζ are left to the design tools.

The circuit theory tells us that when θ and ϕ are given, there exists trade-off between D and E when we optimize ζ. We model these constraints as

$$E(\theta, \phi, \zeta) \geq f_{\theta,\phi}\left(D(\theta, \phi, \zeta)\right) \tag{2}$$

Here $f_{\theta,\phi}$ is a function that is monotonically decreasing. The curve given by $f_{\theta,\phi}$ is often called the Pareto optimal curve, or energy-efficiency curve.

Now we can express our optimal problem as

$$\begin{aligned} \min_{\theta,\phi} \quad & E \\ \text{subject to} \quad & E \geq f_{\theta,\phi}(D) \\ & D \leq \widetilde{D} \end{aligned} \tag{3}$$

We will first model the constraint functions $f_{\theta,\phi}$ and fit them into the training data set, and then do optimization to solve the best design. More details are explained in Section 4.

3.2 Circuit-Level Model

As we said in Section 2, the circuit-level features ϕ can be modeled theoretically based on circuit-level model. Given a circuit design (θ, ϕ), the trade-off between D and E by changing ζ, a.k.a, $f_{\theta,\phi}$, is given by the following model [1]

$$E(\theta, \phi, \zeta) = f_{\theta,\phi}\left(D(\theta, \phi, \zeta)\right) = \frac{K(\theta, \phi)}{D(\theta, \phi, \zeta) - D_0(\theta, \phi)} + E_0(\theta, \phi) \tag{4}$$

where $D_0(\theta,\phi)$, $E_0(\theta,\phi)$ and $K(\theta,\phi)$ are fitting parameters that depends on θ and ϕ.

In our circuit-level model, $\phi = (V_{\mathrm{DD}}, \mathrm{THL}, p)$, in which V_{DD} is the supply voltage, THL is the threshold voltage level, and p is the number of pipeline stages. The circuit delay and energy now can be written as the following with respective to these three features explicitly [3]

$$\begin{aligned} D(\theta, V_{\mathrm{DD}}, \mathrm{THL}, p, \zeta) &= \left(\frac{a'_D(\theta)}{p} + b'_D(\theta)p\right) V_{\mathrm{DD}}(V_{\mathrm{DD}} - V_T(\mathrm{THL}) - V_{\mathrm{on}})^{\gamma_D} \times d_{\theta,\phi}(\zeta) \\ E(\theta, V_{\mathrm{DD}}, \mathrm{THL}, p, \zeta) &= \left(a'_E(\theta) + b'_E(\theta)p\right) V_{\mathrm{DD}}^2 \times e_{\theta,\phi}(\zeta) \end{aligned} \tag{5}$$

where a'_D, a'_E, b'_D, b'_E are parameters that depend on the architecture θ, and $\gamma_D, V_{\mathrm{on}}, V_T$ are independent with θ. $d_{\theta,\phi}(\zeta)$ and $e_{\theta,\phi}(\zeta)$ are functions of ζ, modeling the dependency on the device-level features.

From (4), we assume that $D_0 \propto D$, $E_0 \propto E$, and $K \propto DE$. So the circuit model for these parameters are

$$\begin{aligned} D_0(\theta, V_{\mathrm{DD}}, \mathrm{THL}, p) &= \left(\frac{a_D(\theta)}{p} + b_D(\theta)p\right) V_{\mathrm{DD}}(V_{\mathrm{DD}} - V_T(\mathrm{THL}) - V_{\mathrm{on}})^{\gamma_D} \\ E_{\mathrm{dyn},0}(\theta, V_{\mathrm{DD}}, \mathrm{THL}, p) &= \left(a_E(\theta) + b_E(\theta)p\right) V_{\mathrm{DD}}^2 \\ K(\theta, V_{\mathrm{DD}}, \mathrm{THL}, p) &= \left(\frac{a_K(\theta)}{p} + b_K(\theta) + c_K(\theta)p + d_K(\theta)p^2\right) V_{\mathrm{DD}}^3(V_{\mathrm{DD}} - V_T(\mathrm{THL}) - V_{\mathrm{on}})^{\gamma_D} \end{aligned} \tag{6}$$

Taking (6) into (4), we can represent the model explicitly with $\phi = (V_{\mathrm{DD}}, \mathrm{THL}, p)$ and parameters that only depend on θ.

3.3 Dealing with Architecture-Level Features

Ideally, after Section 3.2, we should continue to model the relationship between the parameters and θ explicitly. However, the effects of different θ are usually unrelated. For example, using two different topology or algorithm will normally result in totally different performance. Currently no simple and accurate model can describe the effects of the architecture-level parameters.

Considering these difficulties, we don't "make up" a model without solid physical explanation, but try to bypass them by utilizing the freedom to control the sparse sampling. We require that the input training set must cover all the architecture-level features so we can learn the parameters depending on θ separately. Since the architecture-level feature space is not big, this is not a very strong requirement, and the designer can still reduce a large number of simulations by sampling sparsely in the lower level design spaces of ϕ and ζ.

4 Methodology

Figure 1 shows the methodology we use. Given a small training set from sparse sampling, we first fit (4) to get D_0, E_0 and K. Then, considering the difficulties that (6) is non-linear and there are both kinds of parameters that depend and do not depend on θ, we try to use two-step fitting for the parameters in (6). We first deal with the factor that doesn't depend on θ by factorizing the equations and dividing the data from each other. Then we use these results to fit the factor depending on θ. After that, we calculate $D_0(\theta,\phi), E_0(\theta,\phi), K(\theta,\phi)$ and obtain the Pareto optimal curves $f_{\theta,\phi}$ for all (θ,ϕ). Finally, we solve the optimization problem (3) by simply picking the minimum E over all (θ,ϕ), and give the "calculated" optimal D and E.

To increase the accuracy, ideally we should feed the calculated optimal configurations back into the synthesis tool and get their actual delays and energy costs. But due to the limited amount of time, another simpler method is used here. Instead of re-synthesizing the calculated configurations, we use (4) to get the "actual" energy E of a certain configuration (θ,ϕ) under a given D, where the parameters of (4) are learned from the complete data set. We call these result the "predicted" optimal D and E, and treat them as the final output of our model.

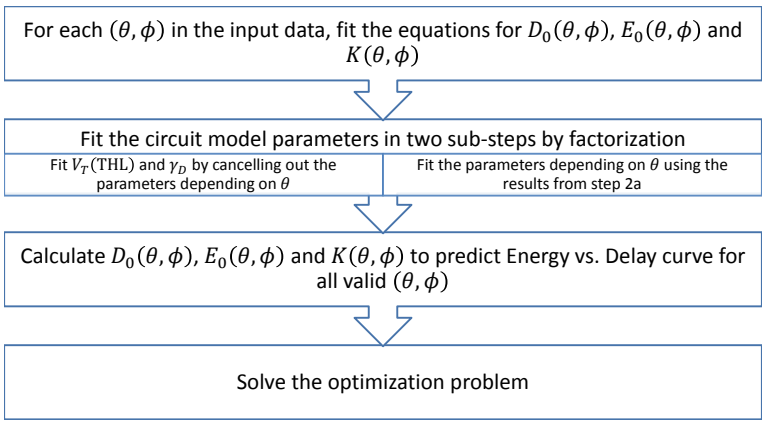


Figure 1: Methodology

5 Results

5.1 Relative Error of (4)

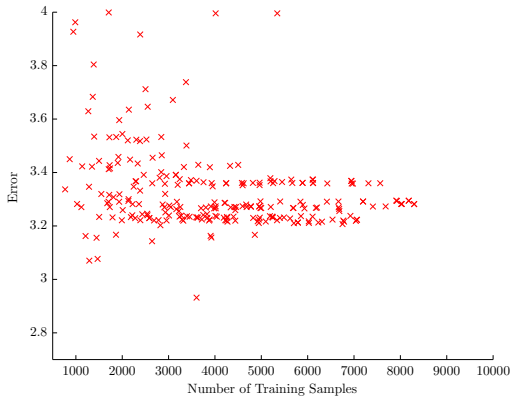
First of all, we evaluate the accuracy of our simpler method of "back synthesis". The difference between this simpler method and the real synthesis should be evaluated carefully. To do this, we use a relative error calculated as the root mean square (RMS) of the relative difference of E between (4) and the real data

$$\varepsilon = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\frac{E_{i,\text{pred}} - E_{i,\text{real}}}{E_{i,\text{real}}}\right)^2} \tag{7}$$

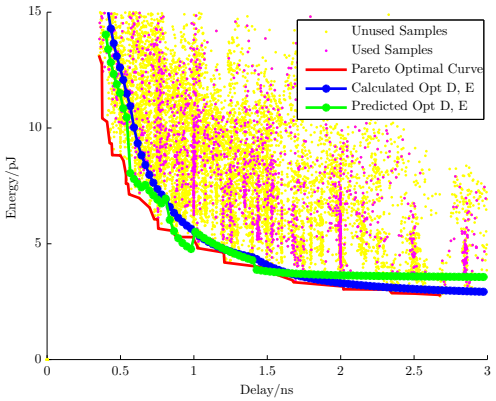
In our experiments, this value is 0.0158. Thus we conclude that the error of (4) is small enough, and it can be used for our evaluation purpose.

5.2 Errors for Different Training Set Sizes

Our primary purpose is to reduce the number of necessary syntheses to get the Pareto optimal curve. In this section, we evaluate the errors for different input training sets. The error is calculated as the average distance between the real Pareto optimal curve and the predicted optimal curve from our model. In our experiments, we reduce the training set size in different dimensions in the feature space (θ, ϕ, ζ) and see the different results.



(a) Errors for Different Training Set Sizes



(b) An example with 1725 Training Points

Figure 2: Results

Figure 2(a) shows the error vs. the training set size. We can see that generally smaller training sets have larger errors. As we use more and more samples, the error will converge to a stable value. However, to our surprise, even some very small training sets are able to predict the Pareto optimal curve with fairly small errors. But the error will vary a lot if we use different training sets. In Section 6, we try to summarize some insights on how to reduce the training set size efficiently.

To present an example, we choose a point in Figure 2(a) with 1725 training points and error equal to 3.52, and show the *D-E* plot in Figure 2(b). The three lines in the figure shows the real Pareto optimal curve (red), the calculated optimal configurations (blue), and the final output predicted optimal configurations from our model (green). The error is the distance between the red line and the green line. We can see that even with this small training set (about 10% of the entire data set), the predicted Pareto optimal curve is very close to the real one, meaning that this is an efficient reduction of the training set size.

6 Discussion

An interesting question to ask is how to choose a efficient reduction of the training set but still keep high accuracy. Here are some guidelines summarized from our experiments:

1. Make sure that the training set covers all θ. This is because we don't model the architecture-level features in the model, as stated in Section 3.3.

2. Reduce but keep a certain value (about 3 to 4) for the number of samples with same θ, ϕ but different ζ. Also it is better to sample uniformly along the curve, i.e., synthesize designs with very large and very small target delays respectively. This will affect the fitting quality for (4).

3. Reduce but keep a certain value (above 30%) for the percentage of samples that share the same θ and cover all the combinations of V_{DD} and THL. This will affect the fitting quality for those parameters in (6) that do not depend on θ.

4. Reduce but keep a certain value (about 8 to 10) for the number of samples with different pipeline stages p. This will affect the fitting quality for those parameters that depend on θ.

7 Conclusion

In this project, we characterize the FPU design space and classify the design features into three categories. Using both theoretical and empirical methods, a model is built to capture the effects of different features. The model is quite helpful to reduce the number of necessary syntheses. By applying our model, we are able to predict the Pareto optimal curve with similar accuracy to the original approach, but using only 1/10 of the original data set. That means a saving of 90% circuit synthesis time.

References

[1] 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 *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 250–255, 2010.

[2] Sameh Galal, Ofer Shacham, John S. Brunhaver II, Jing Pu, Artem Vassiliev, and Mark Horowitz. Fpu generator for design space exploration. *Computer Arithmetic, IEEE Symposium on*, 0:25–34, 2013.

[3] T. Sakurai and A.R. Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *Solid-State Circuits, IEEE Journal of*, 25(2):584–594, 1990.