A Quantitative Approach to Development and Validation of Synthetic Benchmarks for Behavioral Synthesis

Chunho Lee and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles, CA 90095-1596, USA
Email: {leec, miodrag}@cs.ucla.edu

Abstract

We present a quantitative approach to development and validation of synthetic benchmarks for behavioral synthesis systems. The approach is built on the idea of *quantitative benchmark selection*. We briefly explain the idea and present experimental results on the quantitative selection and validation of benchmarks. We develop a synthetic design example generator which composes the behavioral level specification of a design having the properties given by a set of numerical parameters. Experimental generation and application of synthetic design examples demonstrate the effectiveness of the proposed approach and the developed algorithms.

1 Introduction

The behavioral synthesis tasks are all NP-complete and non-optimal heuristics must be employed. Consequently, comparing and evaluating the quality of final results produced by computer-aided design (CAD) systems are one of the most important issues that need to be addressed. CAD software designers heavily rely on benchmarks to quantitatively demonstrate the usefulness of their algorithms and implementations in their development process. Moreover, as observed in the past years, rapidly changing electronics markets and technologies call for flexible and adaptive benchmarks that can be quantitatively characterized so as to accommodate varying dimensions of future trends and to extend the existing benchmark results.

The applications of benchmarks can be summarized as follows: (1) comparison and evaluation of given synthesis systems, (2) fine-tuning synthesis software, (3) software validation and verification, (4) finding weaknesses and strengths of synthesis tools and (5) research guidance in identifying problems that are not solved adequately. From the desired applications of benchmarks, we derive quality criteria that will guide the design process of a benchmark suite: relevancy, compactness, comprehensiveness, and resolvability. A benchmark suite is based on real design examples representing their full diversity and complexity. A benchmark suite must contain as few designs as possible while covering all of common designs. A benchmark should result in a fair comparison among competing algorithms and implementations with good resolutions.

One advantage of synthetic benchmarks is that we can avoid the risk of inadvertent (or intended) abusive use of benchmarks by ensuring that they are realistic and unpredictable. Repeated use of a benchmark set may cause an undesirable side effect called overtuning unless we prudently use a set of *learning* benchmarks which are different from *testing* benchmarks. A series of synthetic benchmarks generated in an unpredictable manner can prevent CAD tool developers from abusive use of benchmarks, while enabling them to exploit the benefits of benchmarking in their CAD software development. We also can verify if a tool is overtuned to a particular set of benchmarks by using such synthetic benchmarks.

Additional benefits of synthetic benchmarks are: (1) we can exploit the benefits of using design examples that are of future trends, various levels of complexity and difficulties, and features of interest to a user and (2) the synthetic examples are often better representatives of real design examples than those benchmarks selected from real design examples are.

In this paper, we present our results on laying out theoretically and statistically sound foundations for addressing the key issues related to the synthesis and analysis of synthetic benchmarks. The method we used for quantitative selection of benchmarks is briefly introduced. We formulate synthetic benchmark development methodology as a statistical optimization process. We develop synthetic benchmark generator based on the formulation.

The rest of the paper is organized in the following order. A brief history of benchmarking is discussed in Section 2. After presenting the global design flow of synthetic benchmarks, synthetic benchmark design and optimization issues are discussed in Section 3. It gives details of the problem formulation and the explanation of the ideas behind the problem formulation. All the algorithms used in the synthetic benchmark generator are presented in Section 4. Sections 5 deals with experimental results. Finally, our conclusion is given in Section 6.

2 Related Work

The first benchmark in the computer architecture and compiler domains was the Gibson Mix [4], which significantly influenced the design of IBM 360 and 370 series of computers. More recently, and in particular in the last decade, benchmarking was promoted in computer architecture as the key method for research and product evaluation. Small and partially concocted examples has been replaced by larger examples which are manually analyzed for diversity [10]. Since then, many key innovations in the architecture and compiler domains, such as cache memory, memory hierarchy, vector processing, RISC in general purpose computing, and multiplier-accumulator units, bit reversal units and circular buffers in general purpose programmable DSP processors, are direct consequences of attempts to achieve superior results on targeted benchmarks.

Recently, benchmarking attracted a good deal of attention in both research [1] and industrial CAD communities. An analysis of DAC 1992 papers indicates that, while more than 30% of authors use their own benchmarks, more than 40% of the papers show results on "standard" benchmarks [1]. Popular CAD benchmarks are available for sequential test generation, logic synthesis, physical design, circuit simulation [1, 2], and DSP applications [11].

Unfortunately, however, practices of benchmark selection for behavioral synthesis systems in the past are more *ad hoc* than systematic, often times resulting in unacceptable benchmarks.

3 Synthetic Benchmark: Design and Optimization Issues

In order for algorithmically generated synthetic benchmarks to be realistic and useful, the real design space information must be incorporated into the generation of synthetic benchmark sets. A synthetic benchmark set is based on real designs only if it represents

ICCAD98, San Jose, CA, USA

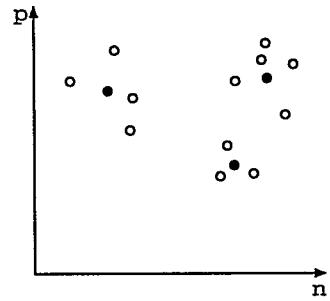

Figure 1: An example design space: black dots are ideal synthetic benchmarks for the design example space

real designs in their exact full diversity and complexity. Generation of realistic and useful synthetic benchmarks involve two phases.

First a collection of real designs is analyzed and design parameters are numerically parameterized. The characteristics of real designs are identified from the analysis. Then, a set of benchmark is selected from the collection of designs. This process ensures that the obtained parameters represent the real design space. One of the advantages of the methodology utilizing numerical parameters is that it gives very convenient representation of design mixes, which enables us to algorithmically generate realistic design examples. For more on the quantitative benchmark selection refer to [5, 6, 7]. A set of designs are selected by a simulated annealing based algorithm. The results including statistical validation of the selected benchmarks are given in Section 5.

A set of numerical parameters obtained during the first phase of the process can be said to be *characteristics* of the real design space. The characteristics are used by a synthetic design example generator to compose designs having the desired properties.

The parameters for a synthetic example is given by computing the mean values of all dimensions of all the design examples in a group. Figure 1 shows a graphical illustration. The best synthetic designs are shown as black dots. The group is identified by measuring the distances from benchmarks to the other design examples. The cluster identification problem is formally given by:

Given a set of N design examples $X = \{x_i | i = 1, 2, ..., N\}$, *and a set of benchmarks* $Y = \{y_j | j = 1, 2, ..., k\}$, *find clusters of design examples* $G = \{\{g_i\} | i = 1, 2, ..., n, \sum_k^n |g_k| = N - k\}$ *such that*

$$g_i = \{r_j | d(y_i, r_j) \equiv \min_{\forall k} d(y_k, r_j)\}. \quad (1)$$

where $r_j \in R$, $R = X - Y$, $j = 1, 2, ..., N - k$, y_i *is the* i^{th} *benchmark and* $d(x, y)$ *refers to a distance function from* x *to* y.

The synthetic example generator (SEG) problem is formally given by:

Given an n-tuple of numerical parameters $(p_1, p_2, ..., p_n)$ *and an m-tuple consisting of quantities of available CDFG components* $(c_1, c_2, ..., c_m)$, *generate a synthetic CDFG which contains exact numbers of components specified by the m-tuple of component quantities and have properties as close to those described by the n-tuple of quantifiable parameters as possible.*

The generation of a random graph satisfying a set of given parameters is not computationally tractable. Therefore, we need to employ a heuristic method.

4 Synthetic Behavioral Design Generation

Our approach is not restricted to a particular computational model. However, we use in our experiments the synchronous data flow

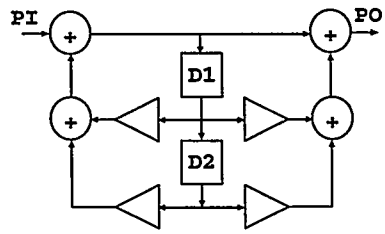

Figure 2: biquadratic second order filter

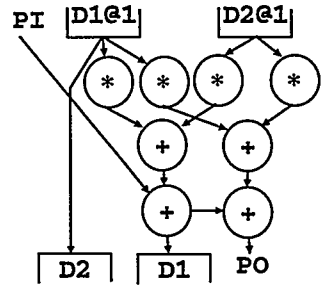

Figure 3: biquadratic second order filter with iteration boundary

(SDF) on semi-infinite streams of data [8] as the underlying computational model since we have a large number of design examples based on the model. A design example is represented by a control data flow graph (CDFG) format [9].

Figure 2 shows an example CDFG. Circles and triangles are operations and rectangles represent delays. Data flow is indicated by arrows. Figure 3 shows the same design as in Figure 2. It is the conceptual representation of a CDFG used in the algorithm. The circles represent various arithmetic operations. Rectangles with one side open are delays. Note that the set of delays on the top of the figure is exactly identical to the set of delays at the bottom of the figure. It shows all the primary inputs and delay outputs on the top of the figure and all the primary outputs and delay inputs at the bottom. Since an iteration starts with firing nodes with all the required inputs from delays or primary inputs and ends with all the primary outputs and delay inputs, our representation of CDFG clearly shows the iteration boundary of a CDFG. This representation is easy to reason about the length of the critical path, the presence of cycles, and the presence and length of loops.

Since the generation of SCDFG given a set of numerical properties is an NP-complete optimization problem, we apply the simulated annealing heuristic to the randomly constructed SCDFG to obtain an SCDFG with the given properties. The initial state of the optimization system is defined by initial temperature and initial random solution. The system moves toward an equilibrium (or frozen) state based on the following functions: cost function, neighbor solution function, temperature reduction function, nondeterministic solution acceptance function, equilibrium criterion and frozen criterion.

We constructively build the initial random solution (initial CDFG) by placing edges between pairs of components in such a way that the resulting graph is valid. There are a number of restrictions in placing edges in a CDFG. A classification of CDFG nodes used in the algorithm is given in Table 1. Table 2 summarizes the restrictions in constructing a random SCDFG.

To ensure the observations given in Table 2 are not violated, the number of edges from primary inputs and delay components should satisfy the following inequality:

node	description
primary input	external input
internal node	arithmetic operations and conditionals
primary output	external output
delay	a node that stores state information

Table 1: A classification of nodes in CDFG

node type	restrictions
primary input	must have at least one edge connecting to an internal node or delay
	cannot be connected to a primary output
internal node	must have at least one edge connecting to an internal node, delay or primary output
	cannot be connected to more than one primary output
primary output	can have only one incoming edge from an internal node or delay
delay	can have one and only one input
	can be placed in-between any pair of components (even in-between a pair of two delays)
loop	a loop L can be formed by connecting the input end of a delay component D to the output end of a node C and the output end of D to the input end of a node that is chained to the node C

Table 2: Summary of CDFG rules

$$\max(2 \times \min_j(i(n_j)) - 1, p) \leq n(e_{pi}) \leq \min(\sum_j(i(n_j)) - k + q \sum_j(i(n_j))), \quad (2)$$

where $n(e_{pi})$ = the number of edges from the set of primary inputs and delays, p = the sum of numbers of primary inputs and delays, k = the number of components with outputs, q = the sum of numbers of primary outputs and delays, and $(i(n_j))$ = the number of required inputs of the component j.

Each primary input and delay has to be connected to at least one arithmetic or delay component. If there are more edges left to be placed, we randomly choose pairs of a primary input or a delay component and an arithmetic operation or a delay component to place edges between them. Next, all primary outputs and all delay components get their respective inputs. This is done by randomly selecting a node in a CDFG other than primary inputs and connecting it to a primary output or a delay component in sequence. The conditions to be met are (1) each primary output or delay can have only one incoming edge, and (2) no more than one primary output or delays can have incoming edges originating from the same node.

The rest of the necessary edges to complete the initial construction of CDFG are placed in the next two steps. They consist of connecting all the outputs of nodes and completing all the necessary input edges of all nodes.

After building the initial solution, all the required objective functions are evaluated to measure the nearness of the current solution to the target solution. If it does not meet the requirements, it enters iteration cycles which stop either when the objectives are met (equilibrium) or when a pre-specified number of iterations are performed (frozen). We use standard geometric cooling schedule.

The next SCDFG is generated by swapping two randomly chosen edges. The new SCDFG has a new set of properties (e.g., the length of the critical path, the length of the longest loop, etc.). The acceptance function for worse solutions is based on the exponential distribution determined by the temperature and the closeness of the new solution to the best solution ever found.

1	NEC digital to analog converter
2	Motorola's 126-tap FIR filter
3	IMEC modem
4	7th order WDF filter
5	8 point 1 dimension decimation-in-time DCT
6	5th order distillation plant controller
7	2nd order Volterra filter
8	(31, 15)BCH code
9	2D 8 point VHS transform
10	Hilbert transform
11	2-D 10-th order IIR filter
12	9th-order rank-order filter

Table 3: The selected benchmark set. The quantified characteristics of this set of benchmarks are used to generate synthetic benchmarks

size[a]	diff.[b]	min.[c]	max.[d]	avg.[e]
125	0	0	0	0
100	0.4	0	2	0.7
75	0.9	0	3	1.7
50	1.2	2	6	3.2

[a] design set size(reduced)
[b] avg. difference(%) given by $D(b_1, b_2)/D(b_1, r)$
[c] min. number of different benchmarks
[d] max. number of different benchmarks
[e] avg. number of different benchmarks

Table 4: The statistical validation results by resubstitution (original set size: 150)

5 Experimental Evaluation

We collected and analyzed a set of design examples and Table 3 shows the selected benchmarks. Table 4 shows statistical validation of selected benchmarks by the resubstitution technique. A set of 150 design examples is used to obtain the initial benchmark set. We select a new benchmark set after randomly eliminating 25 design examples from the design pool. In each trial, we observe the number of benchmarks that are replaced by new ones and the difference between two sets of benchmarks. For the computation of the difference between two sets of benchmarks, we use the *Hungarian Method*. We run this sequence of tests 10 times for a given design withdrawal ratio (e.g., 25, 50, etc.) and the results are summarized in Table 4. The second column shows the average difference between the original benchmark set and a new benchmark set which is obtained after eliminating a number of design examples from the design pool. The average difference is represented by the ratio of the distance between two benchmark sets (original and new) to the total distance between the original benchmark and the rest of the designs. The third and fourth columns are for the minimum and maximum number of replaced benchmarks from the original benchmark set. In the final column, the average number of benchmarks replaced is given. When a benchmark set obtained by the resubstitution technique is compared to the original benchmark set by way of measuring the total distances from the set to all the design examples in the original set, the difference in terms of distance is less than 4%. This indicates the method presented in this paper is highly effective [3].

Table 5 shows a comparison between selected benchmarks and synthetic benchmarks in terms of the total distance between a benchmark set and the rest of the design collection. The table shows six different benchmark selection experiments with different design collection sizes and different benchmark set sizes. The first and second columns are for design set size and benchmark set size, respectively. The third and fourth columns display the total dis-

Designs[a]	Bench.[b]	Orig.[c]	Synth.[d]	Imp.[e]	Time[f]
25	5	4551	3649	20%	8
50	10	7765	6543	16%	31
75	15	902	730	19%	51
100	10	1282	1122	13%	99
125	10	22927	21234	7%	89
150	10	2262	2006	11%	109

[a] the size of collected design examples
[b] the size of selected benchmarks
[c] the sum of distances between the collected designs and benchmarks
[d] the sum of distances between the collected designs and synthetic benchmarks
[e] the percentage of reduction in the sum of the distances
[f] the run time to measure (sec.)

Table 5: The sums of distances from corresponding benchmarks to design examples

target	initial	final	diff.	diff.(%)
20,439	35,723	20,517	78	0.4
15,334	27,545	15,329	-5	0.0
28,565	46,921	27,524	-1,041	3.6
45,977	66,372	43,862	-2,115	4.6
64,261	127,544	67,387	+3,126	4.9
59,168	81,661	59,165	3	0.0
52,023	84,664	56,143	+4,120	7.9
178,554	466,142	178,591	+37	0.0
166,269	489,667	170,046	+3,777	2.3

Table 6: The quality of generated SCDFGs

tance between a benchmark set and the rest of the design collection (one for the selected benchmarks and one for the synthetic benchmarks). The distance is measured in the L_1 (*Manhattan*) distance. The improvement ratio is given in the improvement column. The running time for each case of computing the benchmark set and corresponding synthetic benchmark set is given in the right most column.

In Table 6, results of SCDFG generation experiments are shown. Of a number of synthetic designs generated for each target properties, the instance with the highest discrepancy between the targeted and the achieved are presented. The quantified properties are scaled to reflect their weights. For example, Gini coefficient ranges from 0 to 100 and sustained parallelism ranges from 3 to 33. We scale the parameters by linear functions. Note that the similarity of the sums of squared parameters does not necessarily indicate that the corresponding designs are similar.

To experimentally verify the effectiveness of synthetic benchmarks, we used generated synthetic benchmarks to tune one of our scheduling and assignment tools for area and power minimization on programmable heterogeneous platforms. The scheduling and assignment program uses a set of nine parameters which adjust scheduling and assignment strategy and choices according to recognized structure of a *typical* workload. The performance of the program varies over factor of almost two orders of magnitude, depending on the selected values for the parameters. The values of the parameters are iteratively set by using Gauss-Jordan iterative procedure through the examination of the performances on a set of learning examples.

We were able to tune the parameters of our scheduling and assignment program to the best performing values using 12 synthetic examples generated which are based on a set of selected benchmarks. The same level of performance was achievable only when we randomly selected between 41 and 83 designs (average 60), clearly indicating the usefulness of the synthetic benchmark ex-

Synth.[a]	Area				Power			
	Perf.[b]	Min.	Max.	Ave.	Perf.[c]	Min.	Max.	Ave.
6	1.92	22	52	33	5.22	20	58	41
7	1.87	23	51	37	4.01	23	59	42
8	1.59	25	57	40	3.08	32	72	48
9	1.32	38	59	43	2.47	31	72	50
10	1.17	38	71	50	1.99	38	80	54
11	1.07	41	92	58	1.41	22	81	59
12	1.00	52	97	64	1.00	41	83	60

[a] Average number of synthetic benchmarks
[b] Performance normalized with respect to the best case
[c] Performance normalized with respect to the best case

Table 7: An experimental evaluation results

amples without having overtuning problems. The more detailed information about ability of synthetic and generic benchmarks to enable convergence of the scheduling program to the most effective set of parameters is given in Table 7. The first column indicates the number of synthetic benchmarks used for a given level of effectiveness shown in the columns 2 and 6 for area and power respectively. We normalize the effectiveness with respect to the best case found. The columns 3-5 and 7-9 show the required number of randomly selected generic benchmarks to achieve the same level of the effectiveness.

6 Conclusion

The experimental results shows that the quantitative approach to benchmark selection is not only possible but also very promising. The synthetic benchmarks the characteristics of which are derived from the quantitatively selected benchmarks can be used in many benchmark application areas. In particular, as our experience demonstrates, fine-tuning optimization tools is one very important application of synthetic benchmarks. Also, quantitative approach promises a convenient and correct way of extending and expanding existing benchmarks by way of synthetic benchmarks since benchmark itself can be quantitatively studied and verified.

References

[1] F. Brglez. ACM/SIGDA design automation benchmarks: Catalyst or anathema? *IEEE Design and Test of Computers*, pages 87–91, September 1993.

[2] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *1989 Int. Symp. Circuits and Systems*, pages 1929–1934, 1989.

[3] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, New York, NY, 1993.

[4] J. C. Gibson. The Gibson Mix. Technical Report TR 00.2043, IBM Systems Development Division, Poughkeepsie, NY, 1970. Widely quoted as a research done in 1959.

[5] C. Lee, D. Kirovski, I. Hong, and M. Potkonjak. DSP quant: Design, validation and applications of DSP hard real-time benchmark. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1997.

[6] C. Lee and M. Potkonjak. A quantitative approach to development and validation of synthetic benchmarks for behavioral synthesis. Technical report, Computer Science Department, University of California, 1997.

[7] C. Lee and M. Potkonjak. Quantitative selection of media benchmarks. In *Proceedings of ASP-DAC '98*, 1998.

[8] E. A. Lee and T. M. Parks. Dataflow process networks. *Proc. of IEEE*, 83(5):773–799, 1995.

[9] J. Rabaey and M. Potkonjak. Resource driven synthesis in the HYPER system. In *1990 IEEE International Symposium on Circuits and Systems*, volume 4, New Orleans, LA, USA, May 1990.

[10] SPEC. SPEC benchmark suite release 1.0. SPEC Newsletter, 1990.

[11] V. Zivojnovic, J. Martinez Velarde, C. Schlager, and M. Meyr. DSPstone: A DSP-oriented benchmarking methodology. In *Proceedings of the 5th International Conference on Signal Processing Applications and Technology*, volume 1, pages 715–720, Dallas, TX, USA, October 1994.