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

Abstract

We introduce a quantitative approach to development and validation of synthetic benchmarks in general and of synthetic benchmarks for behavioral synthesis systems in particular. The first step of the approach is numerical parameterization of characteristics of a representative set of designs at the behavioral level. Experimental results on quantitative selection and validation of benchmarks are briefly presented. Based on the quantitative benchmark selection, we lay out theoretically and statistically sound foundations for addressing the key issues related to the synthesis and analysis of synthetic benchmarks for behavioral synthesis systems.

The key component of the new approach is the synthetic design example generator which composes the behavioral level specification of a design in such a way that the design possesses the properties specified by a given set of numerical parameters. It is used to produce a set of designs which have the same selected set of properties as the available set of real designs. Experimental generation and application of synthetic design examples demonstrates the effectiveness of the proposed approach and the developed algorithms.

1. Introduction

Ever since the Turing test [16], benchmark tests played a very important role in various areas of computer science. In fact, some industrial benchmark suites have greatly helped improve the quality of contemporary digital systems. As computer hardware and software systems become increasingly more complex, testing and simulation tend to replace verification and validation. Since developers have to address the issues of correctness and performance at the same time, benchmarks play a more critical role than ever in complex system development processes.

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 CAD systems, as well as correctness and performance of the systems, are very important issues. CAD software designers heavily rely on benchmarks to verify the usefulness of their algorithms and designs in their development process since the quality of final products delivered by a CAD system is one of the performance measures. On the other hand, we have the inevitable problem of overtuning CAD tools to a particular set of benchmarks. In the context of CAD tool development assistance, synthetic benchmarks have very attractive potential.

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 fundamental questions to be answered are how to characterize individual designs and how to select a set of benchmarks from a pool of design examples.

In this paper, we present our results in 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. In Section 3, preliminary materials such as basic assumptions, definitions of terms, and design examples used for a study on quantifiable design parameters are given. Section 4 explains quantitative benchmark selection that is a stepping stone to the quantitative synthetic benchmark generation. After presenting the global design flow of synthetic benchmarks in Section 5, synthetic benchmark design and optimization issues are discussed in Section 6. 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 7. Sections 8 deals with experimental results. Finally, our conclusion is given in Section 9.

2. A Brief History of Benchmarks

It appears that the first benchmark in the computer architecture and compiler domains was the Gibson Mix [7], which significantly influenced the design of IBM 360 and 370 series of computers. It is worth mentioning the benchmarking of random number generators, where an sophisticated and theoretically sound test system is developed [10]. More recently, and in particular during the last several years, 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 [14]. 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 [2, 1], and DSP applications [17].

Although there are several often cited sets of high level synthesis benchmarks, the 5th order elliptical wave digital filter, by far the most popular benchmark, has been playing a more important role in the development of high level synthesis systems. For instance, a great number of new techniques (e.g., new scheduling and assignment algorithms and transformation mechanisms) as well as many new optimization targets (e.g., testability, fault tolerance, power and latency minimization) were first illustrated (and sometimes even motivated and justified) by presenting the implementations of this example. The example was first used by Dewilde [4], and in 1988 the IMEC group presented the full implementation of this example according to CCITT standards [3].

However, even a brief analysis of the 5th order elliptical WDF reveals many peculiar properties for a benchmark. For example, a study of several hundred DSP examples indicated that the 5th order elliptical WDF is the only example where all nodes are in cycles and where the critical path was from algorithmic delay to algorithmic delay. Furthermore, this benchmark example has regularity significantly lower than the great majority of DSP examples and exceptionally high non-uniformity of distribution of operations over time. Since wave digital filters are widely recognized as exceptionally numerically stable, it is exceptionally sensitive on the application of computational transformations. It was also noted that the structure is unnecessarily complex for the required transfer function [3].

In the outset, some authors used functionally incorrect implementations. The instances of the 5th order elliptical WDF where some operations were simply deleted have been published. Some authors simply remove feedback edges, which does not result only in functionally incorrect implementations but also in significantly reducing both the number of required interconnect and registers, giving them very improper cost advantages. Although all multiplications are by constants, and de facto standard procedure is to use expensive multipliers for implementations, the available time is set unrealistically low (equal or very close to the critical path).

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

3. Preliminaries

3.1. Definitions and Assumptions

The terms design, design example, design problem, or synthesis problem refer to an application to be synthesized. We use the terms interchangeably in this paper. By benchmark, we mean an instance of a synthesis problem that will be tested on a given set of products or, more specifically, synthesis systems. Sometimes, it also refers to a benchmark test or a benchmarking which means an application of a benchmark set on a given set of synthesis systems. Generally, a benchmark test or benchmarking is a systematic and standardized methodology of evaluating and comparing competing products. A benchmark set refers to a set of design examples that are assembled to achieve the goals of benchmark tests. A synthetic design is an artificially generated instance of a synthesis problem.

The underlying computational model for synthetic design examples is the synchronous data flow graph (SDFG) on semi-infinite streams of data [12]. A design example is represented by a control data flow graph (CDFG) format [13].

Also assumed is that the available pool of design examples, which is used to characterize design space, is reasonably complete in the sense that any new design example of interest is similar to at least one of the design examples already included in the pool of design examples in its diversity and in its distribution. Two design examples are said to be similar if there is no big discrepancy between them in all the parameters relevant to the high level synthesis process. It is assumed that a pair of similar design examples test a set of similar functionalities and features of a synthesis system, therefore, eliminating the need to use both of them as benchmarks. The discrepancy refers to relative difference between two examples compared to other examples. The determining factors to dissimilarity are: (1) how differently two design examples behave compared to other designs when they are used to test a high level synthesis system, and (2) how costly it would be to use both of them as benchmarks. The second point is related to increasing the benchmark set size. An obvious case is that all the available design examples are selected as benchmarks. Then the similarity of the benchmark set to the design example space is perfect at the cost of economy of benchmark tests.

3.2. Key Observations: What is a High Quality Benchmark?

We summarize the applications of benchmarks in this subsection. Based on the observed applications of benchmarks, we give a set of benchmark quality criteria.

Traditionally, benchmarks have been used for the purpose of comparison and evaluation of given synthesis systems. We envision that benchmarking should yield information not only about establishing proper ranking among the available synthesis tools but also about identifying which synthesis system will most likely perform best for a given mix of customer applications.

Secondly, the information from a benchmark test can be used for the finetuning purposes during the development of synthesis software. Benchmarks provide a good guidance how to tune heuristics parameters in synthesis algorithms and compilation tools to the characteristics of typical design examples.

Thirdly, benchmark test results can be used for software validation and verification purposes. Benchmarks are often an invaluable aid in software system maintenance and debugging. If a benchmark test is performed whenever a software package needs changes, updates, or bug-fixes, one can be assured to a significantly higher degree that no new bugs are introduced by the changes.

Also, benchmark test results may be used to improve the quality of synthesis tools. To that account, a benchmark test should be designed such that the results of the test give enough information to pinpoint a particular weakness of a tool. For example, if well designed, benchmark tests produce the information that characterizes the type of design examples on which a given tool does not perform as well as other tested tools do.

Finally, benchmark tests can be a useful research guidance. A well designed benchmark should help researchers to identify what are the key research areas and what design goals to optimize. For example, the comparison of the benchmark test results of currently available tools to the theoretical bounds of what can be achieved may give researchers insights as to which areas or optimization techniques should be investigated further in order to improve the currently available technology.

On the basis of the desired applications of benchmarks, we propose a set of quality criteria that will guide the design process of a benchmark: relevancy, compactness, comprehensiveness, and resolvability. The relevancy calls for a benchmark test to be based on real design examples representing their full diversity and complexity. Since the implementation of design specifications and their syntheses are often time consuming and expensive, a benchmark test must contain as few designs as possible. At the same time, it is also important to cover all types of common designs. A good benchmark set strikes balance between its size and comprehensiveness based on real life design examples. Finally, a benchmark should result in a fair comparison among competing tools and approaches with good resolutions. In other words, we do not want all the benchmarks too easy or too difficult, which may result in no differentiation among the various systems that are not equal in their quality.

3.3. Advantages of Quantitative Synthetic Benchmarks

Quantitative synthetic benchmarks should satisfy the criteria described in the previous section to be a viable tool for the benchmark application areas. That is to say that we can avoid the risk of inadvertent (or intended) abusive use of benchmarks by ensuring that they are realistic and unpredictable. As noted in the previous sections, repeated use of a benchmark set may cause an undesirable side effect called overtuning unless we prudently use a set of *learning* benchmarks which is different from *testing* benchmarks. A series of synthetic benchmarks generated in an unpredictable manner can prevent CAD tool developers from invalid 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.

Design examples can be generated by a synthetic example generator with exact specifications we need. The specifications can include future trends, the various levels of difficulties, and the features of interest to a user.

As explained later, the synthetic examples are often better representatives of real design examples than those benchmarks selected from real design examples are in the sense that they are more similar to all the members of a group it represents.

3.4. Key Ideas: How to Develop Synthetic Benchmarks?

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

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. A study on quantifiable design parameters is given in the next section. 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.

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.

4. Quantitative Benchmark Selection

The notion of quantitative benchmark selection is the key concept that makes it possible to algorithmically generate realistic synthetic benchmarks. In this section, we give an introduction to ideas and methodology of quantitative benchmark selection. A set of designs collected for quantitative benchmark selection is explained and relevant parameters of the designs are discussed later.

4.1. Problem formulation

Selecting a set of benchmarks from the real design example set is the first phase in the synthetic benchmark design methodology. The selection can be done by identifying clusters in the design example space and selecting a set of design examples that best characterizes the design example space. As an example, Figure 1 shows a conceptual design space. A set of design examples are distributed in a design example space. There are two discernible clusters in it. The design examples indicated by A and B can be selected as a pair of better representatives for the two clusters than any other pair of two examples.

The benchmark selection problem is informally described as follows:

"A sufficiently large set of N examples is given and each example is *numerically* characterized by a set of relevant properties. The task is to select k examples which according to some quantitative criteria are the best representatives of all examples."

We now define the problem using more formal Garey-Johnson format [6].

Quantitative benchmark selection problem

Instance: A sufficiently large set X of N examples and a constant M are given. Each example x_j is *numerically* characterized by a set of m relevant properties, x_{jl}, $l = 1, ..., m$.

Question: Is there a set Y of k examples y_i, $i = 1, ..., k$ such that $\sum_{r \in R} D(x_r) \leq M$, where $D(x_r) = \min_{\forall i} Dist(x_r, y_i)$ and $R = X - Y$?

$Dist(x_r, y_i)$ can be any distance measure, for example, geometric $\sqrt{\sum_{l=1}^{m} (x_{rl} - y_{il})^2}$, or Manhattan $\sum_{l=1}^{m} |x_{rl} - y_{il}|$.

Determination of the constant M is not a trivial task. For practical reasons the problem is divided into two subproblems: benchmark set size determination and benchmark selection. The benchmark selection subproblem given a benchmark set size is defined as follows:

"Given the size of a benchmark set k and a set of N examples $X = \{x_i | i = 1, 2, ..., N\}$, find a set of examples $Y = \{y_j | j = 1, 2, ..., k\}$ such that

$$\textbf{Minimize}:$$

$$D_k = \sum_{i=1}^{|R|} d_i, \tag{1}$$

where $R = X - Y$, $d_i = \min\{d(r_i, y_j) | j = 1, 2, ..., k\}$, and $d(r_i, y_j)$ refers to a distance function from r_i to y_j."

The size of a benchmark set is determined by an iterative test that examines the effect of selecting a certain size by measuring total distances from each design example to selected examples as given by the expression (1). We continue the set size determination process until we reach a point where the set size strikes balance between the economy of benchmarks and its effectiveness. We compute and compare the total distances incrementing the size one at a time until we get no significant improvement by adding more design examples to the benchmark set.

The benchmark set size problem is formally given by

"Given a set of N examples $X = \{x_i | i = 1, 2, ..., N\}$, select a benchmark set size k such that

$$\min\{k | \rho \leq \frac{D_{k+1}}{D_k}, k = 1, 2, ...N - 1\}, \tag{2}$$

where D_k is given by the expression (1) and ρ refers to a target ratio."

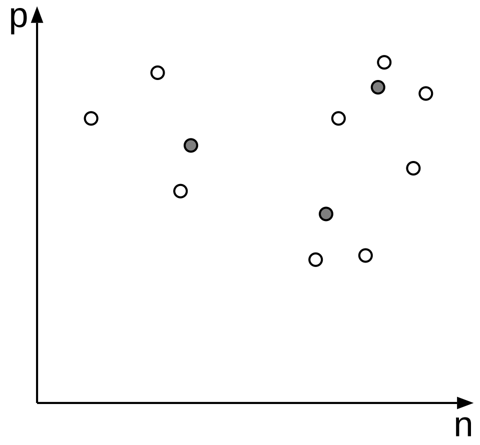

Figure 1. An example of design space and its members: a design is a point determined by its parameterized characteristics. Parameterized characteristics in this example are n and p that refer to the number of operations and the length of the critical path, respectively. Shaded points are candidate benchmarks.

The benchmark set is validated using the resubstitution technique [5]. A number of randomly chosen design examples are eliminated from the pool of design examples from which the original benchmark set is selected. And then a new benchmark set is selected. By comparing the new benchmark set with the original by the weighted bipartite matching technique, we can statistically validate the selected benchmark set. We use *the Hungarian Method* [11] for the matching problem.

4.2. Designs Used for Design Space Characterization

In order to properly characterize the typical media design examples, we assembled 150 DSP, video, image, graphics, telecommunication, and information theory application examples. The assembled design set include three versions of the low pass 5th order elliptical wave digital filter, the 7th and 9th order wave digital filters, Winograd Convolution, several different forms (direct, cascade, parallel, continued fraction and Gray-Markel ladder) of the 8th order IIR bandpass Avenhaus filters, several different Winograd transformations, several 1D and 2D DCT algorithms, VHS transform, NTSC encoder and decoder, G2 blend surface calculation using quintic polynomials, several types of LMS adaptive filter, Hilbert Transform, 2nd order Volterra filter, 3rd order truncated Volterra filter, 28th, 50th, 60th, 100th, 126th and 133rd order FIR filters with a variety of transfer function and applied windowing techniques, several PCM DA converters, several NEC CD ROM digital to radio converters, image histogram, a variety of rank-order and linear video filters, Philips radio coder and decoder, Reed-Solomon, Goppa, Reed-Muller, Fire, Hamming and BCH codes.

Some interesting properties of the examples are: maximum sustained parallelism, percentage of nodes in cycles, critical path, integer iteration bound, most dominant pattern (multiply-add), second most dominant pattern (add-add), and coverage by the pair of dominant patterns. The maximum sustained parallelism for the examples ranges from 3 to 33 and the median is 8. The percentage of nodes in cycles are observed 0% to 100% with the median of 52%. The critical path lengths of the examples are from 3 to 3,161,852 with the median of 14. The percentage of the most dominant pattern and the second most dominant pattern are from 0% to 98% and from 0% to 46%, respectively. The median values of the percentage of the patterns are 39% and 13%, respectively. The coverage by the pair of dominant pattern ranges from 0% to 98% and the median is reported 39%.

4.3. Properties of Designs

We selected the following properties as being relevant to describing the designs. We first included the number of operations, the number of types of operations, the Gini coefficient for the number of operations. The Gini coefficient is an indicator of the concentration of one type of operations [15]. The largest possible value (100) indicates that all operations are of one type, the smallest possible value (0) indicates that all types of operations are equally represented. Timing measures are quantified through the following metrics: the ratio of the sample rate to the critical path, the number and types of nodes on the e-critical network (the subset of the graph which includes all nodes and edges on paths of the length greater than $(1-e)$ times the critical path length, we used value $e = 0.2$), initial average and median scheduling slack, and the iteration bound. The concurrency measures used are the average parallelism, and the Gini coefficient for all types of operations, average and median variable density temporality [8]. Finally, we also included regularity [8] which captures the degree to which common patterns appear in the computation.

4.4. Selected Benchmarks

A set of designs is selected by a simulated annealing based algorithm. The results including statistical validation of the selected benchmarks are given in Section 8.

5. Approach: Synthetic Benchmark Design Flow

The global design flow of the synthetic benchmark is shown in Figure 2. The shaded area of the diagram indicates the quantitative benchmark selection process.

Even though the benchmark selection algorithm is not based on design space clustering, each benchmark conceptually represents a group of similar design examples. It is possible to identify those groups of design examples by measuring the similarity of each design example to each benchmark. This enables us to set synthetic design parameters in such a way that resulting synthetic benchmarks are positioned at the center of conceptual multi-dimensional sphere of a cluster. The next step in synthetic benchmark design is to develop a synthetic example generator (SEG). The SEG generates random design examples given a set of characterizing parameters.

Once the SEG is developed, a synthetic benchmark set based on the real benchmark set can be generated. Furthermore, by extrapolating the current trend in an application domain, we can easily obtain statistically predicted future applications to test to see how well a synthesis tool can perform on the future designs. After synthetic design examples are generated, they are validated using statistical techniques.

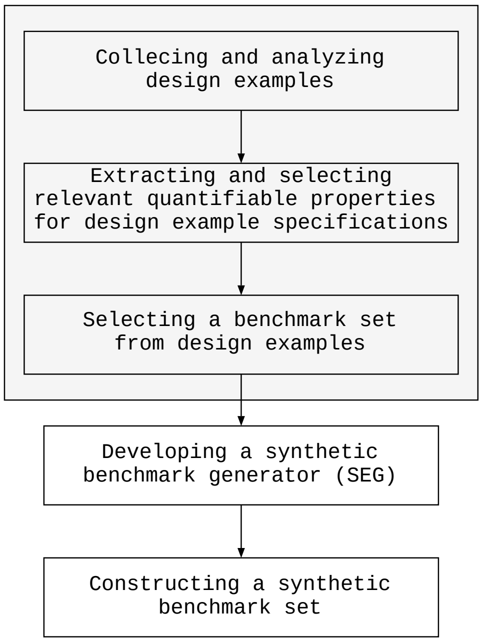

Figure 2. The global design flow of the synthetic benchmark generation

6. Synthetic Benchmark: Design and Optimization Issues

In this section, we formulate problems associated with synthetic benchmark generation. The characteristics represented by a set of numerical parameters are used to synthesize random benchmarks (synthetic benchmarks). A formal problem formulation of a synthetic example generator is given in the following subsection.

6.1. Synthetic Benchmark Parameters

From the benchmark set explained in Section 4, we define a set of parameters that are to be used to generate a set of synthetic examples. The parameters for a synthetic example is given by computing the mean values of all dimensions of all the design examples in a cluster. Figure 3 shows a graphical illustration. The design space the figure represents is the same as in Figure 1. 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

$$EDITg_i = \{r_j | d(y_i, r_j) \equiv \min_{\forall k} d(y_k, r_j)\}, \tag{3}$$

where $r_j \in R$, $R = X - Y$, $j = 1, 2, ..., N - k$, and y_i is the i^{th} benchmark."

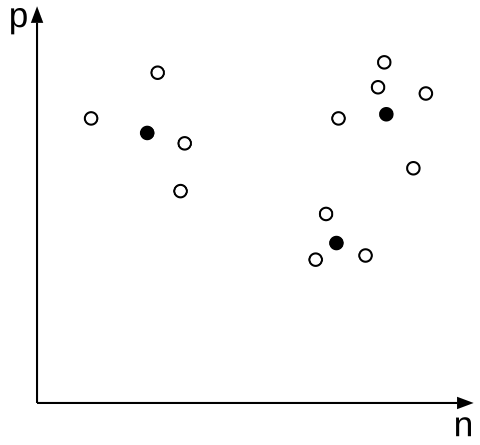

Figure 3. This example design space is identical to what is shown in Figure 1. The black dots are ideal synthetic benchmarks for the design example space

6.2. Synthetic Example Generator

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

Informally, the synthetic example generator (SEG) problem can be described as follows:

"A set of quantifiable parameters relevant to characterizing a CDFG and a set of component numbers are given. Our objective is to generate an individual synthetic CDFG such that the individual synthetic CDFG conforms to the desired characteristics imposed by the given set of parameters."

The SEG problem can be defined in more formal terms:

"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."

7. Synthetic Behavioral Design Generation

In this section, we present how synthetic behavioral designs are generated. Detailed algorithms are given in subsequent subsections. We use CDFGs to represent behavioral designs.

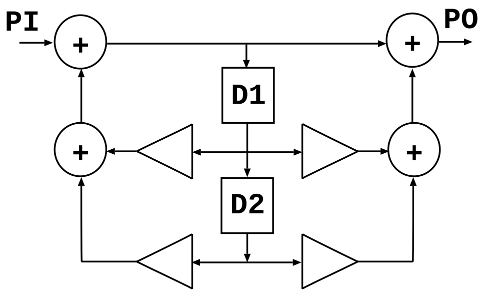

Figure 4. The CDFG representation of the biquadratic second order filter

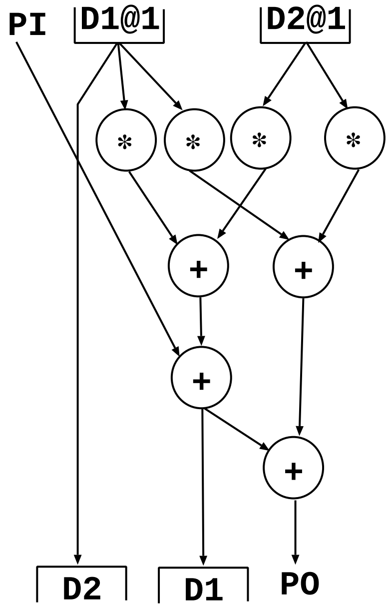

Figure 5. A CDFG representation of the biquadratic second order filter showing its iteration boundary

7.1. CDFG Representation

Figure 4 shows an example CDFG. Circles and triangles are operations and rectangles represent delays. Data flow is indicated by arrows. Figure 5 shows the same design as in Figure 4. It is the conceptual representation of a CDFG used in the algorithms for constructive generation of a synthetic CDFG. 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.

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. A summary of CDFG rules observed in synthetic CDFG generation

7.2. CDFG Rules Observed

Constructing a CDFG is 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.

7.3. Construction of an Initial Random SCDFG

The generation of a synthetic CDFG consists of two stages. First, a random SCDFG without imposing the given property parameters is constructed. Then, we iteratively swap two edges so as to make the properties of the SCDFG close to the targeted. By edge-swapping we mean the exchange of destinations of two edges.

The top-most algorithm for the SEG is given in Figure 6.

```
synthetic_CDFG ( ) {
   initial_graph ( );
   simulated_annealing ( );
}
```

Figure 6. SEG algorithm

```
initial_graph ( ) {
   connect_input_edges ( );
   connect_output_edges ( );
   connect_all_outputs ( );
   connect_all_inputs ( );
}
```

Figure 7. Initial random CDFG construction algorithm

```
connect_input_edges ( ) {
   select_number_of_input_edges ( );
   connect_primary_inputs_to_nodes ( );
}
```

Figure 8. Algorithm to place outgoing edges from primary inputs

The initial random SCDFG is constructed in four steps. First of all, to ensure the observations given in Table 2 are not violated, all primary inputs are connected to a randomly chosen set of components. Likewise, the subsequent steps given in Figure 7 are designed to follow the observations.

There are several conditions to be met when selecting the number of edges originating from primary inputs and delay components. First, there should be at least one internal node that has all the required inputs from primary inputs or delay components. Second, in addition to the previous requirement, there should be at least one internal node that has all the required inputs but one from primary inputs or delay components. These two requirements are needed to ensure there is no *inevitable* cycle in a SCDFG. Also, all the primary inputs and delay components should be connected to one or more internal nodes. Furthermore, the number of edges from primary inputs and delay components cannot be greater than the sum of all required inputs of all internal nodes and should be less than the sum of all required inputs of all internal nodes plus the number of primary outputs minus the number of arithmetic and logical operations.

The number of edges from primary inputs is given by:

$$\max(2 \times \min_j(i(n_j)) - 1, p) \leq n(e_{pi}) \leq \min(\sum_j(i(n_j)) - k + q, \sum_j(i(n_j))), \tag{4}$$

where $n(e_{pi})$ = the number of edges from the set of primary inputs, 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. The algorithm is given in Figure 9.

After randomly selecting a number satisfying Inequality (4), edges are to be actually placed. Each primary input has to be connected to at least one arithmetic or delay component. Therefore, we first put an edge from each and all the primary input to a randomly selected arithmetic operation or a delay component. If there are more edges left to be placed, we randomly choose pairs of a primary input and an arithmetic operation or a delay component

```
select_number_of_input_edges ( ) {
   generate a random number such that:
      (1) there is at least one node that has a complete set of
          inputs connected to either primary inputs or delays
      (2) there is at least one node that has at least a subset
          of required inputs from either primary inputs or delays,
      (3) the number should not be greater than all the
          required inputs of all the nodes minus the number of
          nodes plus the number of primary outputs and delays, and
      (4) the number should be greater than or equal to the number
          of primary inputs and delays.
}
```

Figure 9. Computing the number of edges from primary inputs and delays

```
connect_primary_inputs_to_nodes ( ) {
   place edges from primary inputs and delays to operations such that:
      (1) all inputs and delays have at least one edge incident to them, and
      (2) at least one node has a complete set of inputs
}
```

Figure 10. Placing edges from primary inputs and delays

to place edges between them. The edge placement algorithm is given in Figure 10.

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, as shown in Figure 11, (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. The algorithms are given in Figure 12 and Figure 13, respectively.

```
connect_output_edges ( ) {
   place edges from operations to primary outputs and delays such that:
      (1) there is one and only one edge connecting a operation and
          a primary output or a delay, and
      (2) there is no edges connecting from the same operation to
          more than one primary outputs.
}
```

Figure 11. Placing incoming edges to primary outputs and delays

```
connect_all_outputs ( ) {
   place edges from an output of an operation to
   an input of another operation such that:
      (1) there is no cycles, and
      (2) the outputs of all the operations should be used.
}
```

Figure 12. Placing outgoing edges from all operations

```
connect_all_inputs ( ) {
  place edges from an output of an operation to
  an input of another operation such that:
    (1) there is no cycles, and
    (2) the inputs of all the operations should be filled.
}
```

Figure 13. Placing incoming edges to all operations

7.4. Iterative Improvement of the Initial Random SCDFG

The generation of SCDFG that satisfies given constraints is an optimization problem. We apply the simulated annealing [9] 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.

The simulated annealing iterations are composed of three major parts: generating a new random solution by applying a controlled perturbation technique on the old solution, evaluating the objective functions, and accepting or rejecting the new solution.

By making a number of controlled random modifications to the old solution, we get a new proposal which may or may not be better. If the new proposal is better than the old, we accept it. Otherwise, we accept it with a probability which is a function of its closeness to the best solution ever found and the current temperature. The temperature controls the iteration and the acceptance ratio of worse solutions at each iteration. The temperature is reduced after trying new solutions pre-specified times. When the temperature reaches to zero, the iteration stops regardless of the quality of the best solution found.

The initial SCDFG constructed in the first step is used as the initial random solution. Then 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).

The crucial tasks, then, are to find: (1) the way a new solution is constructed and (2) the acceptance function when a new solution is worse than the old ones. 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.

This process is performed until the SCDFG has properties defined by the set of parameters or some number of iterations are done (in this case when the temperature variable that controls the acceptance ratio and iterations is zero). We use standard geometric cooling schedule.

8. Experimental Evaluation

The experimental results are presented in the following way. First we give a set of selected benchmarks and its statistical validation. From the selected benchmarks we derive synthetic benchmark characteristics. After we show that the derived characteristics can be better representative of the design collection, synthetic benchmark generation results are given. We discuss our experimental application of synthetic benchmarks for fine-tuning a CAD tool.

8.1. Selected Benchmarks and Statistical Validation

We collected and analyzed the design examples presented in Section 4.2. 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. At 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* [11].

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. As the result clearly shows there is no big difference in the configuration of benchmarks when resubstitution is applied. In addition, even though there are cases where several benchmarks are replaced by some other design examples, the difference in terms of distance between two set of benchmarks are small enough to claim that those sets are essentially similar. 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 [5].

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

design set size (reduced)	avg. difference(%) $D(b_1, b_2)/D(b_1, r)$	min. number of different benchmarks	max. number of different benchmarks	avg. number of different benchmarks
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

Table 4. The statistical validation results by resubstitution (original set size: 150). The avg. difference is the ratio of the distance between two benchmark sets to the total distance between the original benchmark and the rest of the designs. The distance between two benchmark sets are obtained by *Hungarian Method*.

8.2. Synthetic Benchmark Generation

Instead of using the characteristics of selected benchmarks, we compute synthetic benchmark parameters from the selected benchmarks. 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 distance 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.

size of design example set	size of benchmark set	original benchmark	synthetic benchmark	improvement	running time (sec.)
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

Table 5. The sums of distances from corresponding benchmarks to design examples. The first two columns are for a design example pool size and a benchmark set size, respectively. For the given design pool size and benchmark set size, the distance between the selected benchmark set and the rest of the designs in the design pool is computed and displayed in the third column. A set of synthetic benchmark parameters are derived from the selected benchmark. The distance between the synthetic benchmark set and the rest of the designs in the pool are shown in the last 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.

8.3. An Experimental Application of Synthetic Benchmarks

To experimentally verify the effectiveness of synthetic benchmarks, we used generated synthetic benchmarks to tune one of our scheduling and assignment tools for power minimization on programmable heterogeneous platforms. The program uses a set of nine parameters. 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

target properties	initial properties	final properties	discrepancy	discrepancy(%)
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 results of SCDFG generation. The values in the first three columns are obtained by summing up squared parameters of a design. The first column is for the targeted design that represents a set of synthetic benchmark parameters. The second and third columns are the sums of squared parameters before and after optimization, respectively. The discrepancy is given by summing up absolute pair-wise relative difference for each parameter. The ratio of discrepancy to the targeted value is given in the last column.

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 based on the selected benchmarks. The same level of performance was achievable only when we randomly selected between 41 and 83 designs (average 60), clearly indicating the expression power of the synthetic benchmark examples without having overtuning problem.

9. Conclusion

We introduced a quantitative approach to the design and validation issues of benchmarks. The study on characterization of design examples and benchmark selection is briefly presented. Based on the benchmark selected, a design of a synthetic CDFG generator is discussed.

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 since benchmark itself can be quantitatively studied and verified. In conclusion, it is possible to design a benchmark set which is flexible and adaptive to rapidly changing technologies and markets.

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] L. Claesen et al. Automatic synthesis of signal processing benchmark using the CATHEDRAL silicon compilers. *Custom Integrated Circuits Conference*, pages 14.7.1–14.7.4, 1988.

[4] P. Dewilde, E. Deprettere, and R. Nouta. Parallel and pipelined implementation of signal processing algorithms. In S. Y. Kung, H. J. Whitehouse, and T. Kailath, editors, *VLSI and Modern Signal Processing*, pages 258–264. Prentice Hall, Englewood Cliffs, NJ, 1985.

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

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.

[7] 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.

[8] L. Guerra, M. Potkonjak, and J. Rabaey. Concurrency characteristics in DSP programs. In *Proceedings of ICASSP-94. IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages II/433–436, Adelaide, SA, Australia, April 1994. IEEE.

[9] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] D. E. Knuth. *The art of computer programming*. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, Mass., 2nd edition, 1981.

[11] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

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

[13] 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.

[14] SPEC Newsletter, Fremont, CA. *SPEC Benchmark Suite Release 1.0*, 1990.

[15] R. A. Thisted. *Elements of Statistical Computing*. Chapman and Hall, New York, NY, 1988.

[16] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236), 1950.

[17] 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.