Published in IET Computers & Digital Techniques
Received on 1st August 2011
Revised on 9th January 2012
doi: 10.1049/iet-cdt.2011.0115

ISSN 1751-8601

Machine learning predictive modelling high-level synthesis design space exploration

B. Carrion Schafer K. Wakabayashi

NEC Corporation, System IP Core Laboratory, 1753, Shimonumabe, Nakahara-Ku, Kanagawa, Kawasaki 211-8666, Japan
E-mail: schaferb@bq.jp.nec.com

Abstract: A machine learning-based predictive model design space exploration (DSE) method for high-level synthesis (HLS) is presented. The method creates a predictive model for a training set until a given error threshold is reached and then continues with the exploration using the predictive model avoiding time-consuming synthesis and simulations of new configurations. Results show that the authors' method is on average 1.92 times faster than a genetic-algorithm DSE method generating comparable results, whereas it achieves better results when constraining the DSE runtime. When compared with a previously developed simulated annealer (SA)-based method, the proposed method is on average 2.09 faster, although again achieving comparable results.

1 Introduction

Very large-scale integrated design is gradually moving towards higher levels of abstraction in order to take advantage of its numerous benefits over traditional register transfer level design approaches. One of these advantages is that higher levels of abstraction combined with high-level synthesis (HLS) allow the architectural trade-off exploration. The main problem with architecture exploration is its exponential order of complexity with the number of explorable constructs in the behavioural description, for example, arrays can be mapped to memory (dual, single port etc.), registers, expanded, making it impossible to perform a full design space exploration (DSE). The presence of multiple objectives in DSE gives rise to a set of optimal solutions, also known as Pareto-optimal solutions, instead of a single optimal solution. In the absence of any further information, none of these Pareto-optimal solutions can be considered to be better than the other. This demands an automatic method to find as many Pareto-optimal solutions as efficiently as possible.

The main challenge lies in how to proof Pareto optimality of the final solution space. As it is extremely hard to proof Pareto optimality, the literature normally refers to the solution space as dominating or non-dominated solutions. In this work we will consider non-dominated designs as Pareto optimal as there is no practical way to prove their optimality.

Fig. 1 shows the source code of a six-tap finite-impulse response (FIR) filter in order to illustrate and motivate this work. The explorable operations have been highlighted and consist of two arrays where the coefficients and data are stored in two loops that perform the filtering function on this data. These operations are considered explorable because they can be synthesised in different ways resulting in very different area against performance implementations, for example, the loop can be unrolled partially or completely and the arrays can be mapped to registers or memory. The table next to the source code shows the result of the HLS for different synthesis attributes (pragmas). The trade-off curve below the table is a screenshot of the exploration results. As seen, the difference between the smallest but slowest design and the fastest but largest design is substantial, with the area ranging from 8316 to 2810 and the latencies from 13 to 1 cycles. There are a multiple Pareto-optimal combinations in between these designs based on different attribute combinations as well as subattributes like the number of memory ports in the array, but only four combinations are shown here for practical reasons. Manually editing the source code in order to explore the different area against performance trade-offs is tedious and time consuming. An automatic, quick and efficient DSE method that finds as many Pareto-optimal designs as possible is therefore highly desirable.

The contributions of this work can be summarised as follows:

- Investigate different machine learning (ML) predictive models for HLS DSE in order to select the most efficient one.
- Introduce a DSE method based on a genetic algorithm ML predictive model, called GA-ML, and investigate the quality of results and runtime against a pure genetic algorithm (GA) method and a previously developed simulated annealer (SA) method. A comprehensive set of results is presented in order to evaluate the newly proposed method.

The paper is organised as follows. Section 2 presents a comprehensive literature review. Section 3 introduces our newly proposed ML-based method to accelerate the DSE. Section 4 provides a set of experimental results to show the efficiency of our method. Finally, Section 5 gives concluding remarks.

```
unsigned char filter(){
   unsigned char ary[6]  /* Cyber array=reg */ ;
   short coeff[6]={5,3,7,9,12,5} /* Cyber array=reg */;
   int sum =0;
   unsigned char value,  char i ;

  /*-- Read input data --*/
/* Cyber unroll_times=all */
   for(i=0;i<6;i++){
      ary[i] = ary_rd[i];
   }

  /*-- multiplication and summation --*/
  /* Cyber unroll_times=0 */
   for(i=0;i<6;i++){
      sum += ary[i] * coeff[i] ;
   }

   /*--- rounding ---/
   if ( sum < 0 ){
      sum = 0 ;
   } else if ( sum > ( 255 << 10 )){
      sum = (255 << 10);
   }
   value = ( sum >> 10 ) & 0xff ;
    return value;
}
```

array1	array2	loop1	loop2	Area	Latency
reg	reg	all	all	8,316	2
expand	reg	all	folding	4,668	3
reg	logic	all	0	3,347	11
ram	ram	all	all	2,810	13

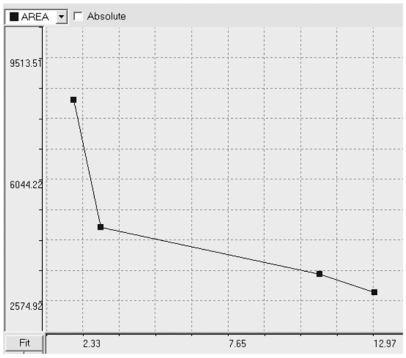

Fig. 1 *Motivational example: FIR filter DSE*

2 Related work

Previous work in micro-architectural DSE in HLS has been focused on applying source code transformation starting from control data flow graphs (CDFGs) using multi-objective function optimisations. Ahmad *et al.* [1] studied the trade-offs between the control step and area in data flow graphs using GAs. Holzer *et al.* [2] used a similar approach using an evolutionary multi-objective optimisation approach to generate Pareto optimal solutions. Haubelt *et al.* [3] use Pareto–Front–Arithmetic's to reduce the search space in embedded systems decomposing a hierarchical search space. Early estimators of area and delay for field programmable gate array (FPGA) implementations were used in [4] to evaluate the design space before any behavioural synthesis. Anderson *et al.* [5] collect system information before the exploration starts doing a configuration sweep and use a GA for the exploration of a parameterised reduced instruction set computer (RISC) processor. A compiler approach to perform hardware DSE is presented in [6] where parallelisation techniques are used to map computations to FPGAs. So *et al.* [7] developed a DSE technique using compiler-directed techniques to perform several code transformations. The starting point in all these approaches is the direct transformations at the CDFG level applying different compiler and optimisations techniques to generate new architectures combined with quick estimators. Givargis *et al.* [8] proposed a system-level exploration technique for systems on a chip (SoCs) by subdividing the design space based on system parameters dependencies, exploring these individually and then incrementally adding the results of each partial exploration together to obtain the Pareto-optimal designs. In previous works we developed exploration techniques based on SA [9] and pattern matching [10]. The most similar work we have found is by Ascia *et al.* [11] where a multi-objective evolutionary method combined with Fuzzy systems for the estimation of performance indexes are used for parametrisable very large instruction word (VLIW) processors. In this work, we not only propose the use of ML to create a predictive model, but also analyse multiple ML methods to investigate which method is better suited for HLS DSE for a commercial HLS tool seen as a black box by our explorer.

3 ML predictive exploration method

The DSE method proposed in this work generates a set of Pareto-optimal designs for a given behavioural description in untimed C or SystemC (SC) by inserting HLS directives directly into the source code. These directives are in the form of pragmas that the HLS tool processes and in turn synthesises the instrumented source code accordingly. The method presented in this work explores loops, arrays and functions. Table 1 shows all the explorable operations and their synthesis directives. A more comprehensive explorer could also explore global synthesis options and the number and type of functional units. The goal of the exploration is

Table 1 Explorable operations

Operation	Attribute	Description
Loops	unroll = 0	do not unroll loop
	unroll = *x*	partial loop unroll
	unroll = all	unroll loop completely
	folding = *N*	fold loop *N* times
Functions	func = inline	inline each function call
	func = goto	single function instantiation
	func = seq_opr	function inst as sequential opr
	fuic = pipeline	function inst as pipeline opr
Arrays	array = RAM	array synthesised as memory
	array = logic	constant arrays synthesised as logic
	array = expand	expand array
	array = reg	synthesise array as registers

doi: 10.1049/iet-cdt.2011.0115

to find as many Pareto-optimal designs as possible. These designs form the Pareto front, also called efficient frontier.

Fig. 2 shows a flow diagram of our proposed method, which consists of five steps:

Step 1: The explorer parses the C/SC code and extracts all the constructs that can be explored, for example loops, functions and arrays.
Step 2: It continues by generating a random set of synthesis directives (pragmas) for all the explorable constructs.
Step 3: It calls the HLS tool in order to synthesise the new design with the new attributes.
Step 4: It then continues by calling the cycle accurate model generator in order to generate and execute a cycle accurate model for each newly synthesised circuit in order to extract the design's exact latency.
Step 5: After X consecutive newly generated designs (empirically we found that $X = 20$ led to good results) our method creates a predictive model and estimates the error between the model and all the previously generated designs. If the error is smaller than a given threshold value, the method continues the exploration using the predictive model instead of generating random configurations with a (GA. If the model error still exceeds the given threshold value, the method continues generating unique random designs and synthesising these until the threshold value is reached. At the end of the exploration, the non-dominated designs obtained are all synthesised and simulated in order to obtain the accurate area and latency results.

Fig. 3 summarises the procedure of the GA used in GA-ML. The procedure starts by creating the initial design population P with N number of new designs $P = \{D_1, D_2, \ldots, D_N\}$ (chromosomes), each with a unique set of synthesis attributes $D_1 = \{A_1, A_2, \ldots, A_m\}$ applied to each explorable construct E_i. It then continues by evaluating the area and latency of each new design $D_1 = \{A_1, A_2, \ldots, A_p|\text{Area, Latency}\}$. The only difference between GA-ML and GA is that GA-ML uses the predictive model to evaluate the area and latency, whereas GA needs to resynthesise each new design and run a cycle accurate simulation to extract this information, which are the most time-consuming parts of the DSE. The procedure then randomly selects two designs and crosses them over by randomly selecting a cut-off point between the attribute lists of the designs. The new design combines the left-half of the attributes list of D_1 and the right-half D_2. The GA continues by mutating the new offspring. This is done by randomly selecting a synthesis attribute A_g from the list (chromosome) and changing it to another attribute randomly. The mutation and cross-over rates have to be specified by the user beforehand. Finally, the mutated offspring is evaluated in terms of area and latency and replace one of the parents if it dominates it in all the objectives (area and latency). If the offspring only dominates one of the objectives, it is randomly decided if it substitutes one of the parents or not.

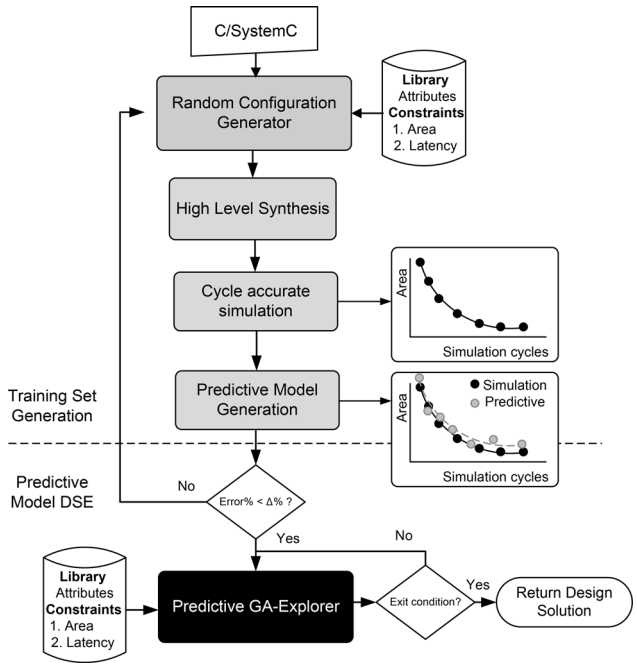

Fig. 2 *DSE flow*

GA-ML: Machine Learning Genetic Algorithm(M, E, A, P, R, C,, N)

/* M: Machine Learning Predictive Model

E :: Explorable constructs in C/SystemC code (gene)

A : List of attributes applicable to each Ei

P: Population size

R : Mutation rate

C: Crossover rate

N: Number of chromosomes per population*/

while(X != N) do /* Generate new chromosomes until exit condition is reached */

- Randomly generate an initial population of P chromosomes Xp consistent of a unique list of A for all explorable operations E.
- Evaluate each chromosome Xi with the predictive model M to obtain its Area and Latency estimates.
- Pair each member of P with another randomly selected member of P.
- Crossover 2 chromosomes Xl and Xm by selecting a random cut-point in the attribute list An of the chromosomes and combining the left half of chromosome with the right half of the other, based on the crossover rate C.
- The new offspring Xp is then mutated, randomly selecting one gene within the offspring and changing it randomly based on the mutation rate R.
- Evaluate the offspring using the predictive model M and replace one of the parents if it dominates it

endwhile;

return PO; /* return Pareto-optimal (non-dominated) designs */

Fig. 3 *Summary of GA*

As it can be observed, the predictive model used is key to the success of our method. We investigated different ML methods using Weka [12], a freely available ML tool suite, which includes numerous ML methods. In our case, we focused on the numeric prediction method as the DSE needs a ML method that can predict the area and latency of a design as a function of the synthesis attributes applied to the design as follows: $\text{Area} = f(A_1, A_2, \ldots, A_n)$ and $\text{Latency} = f(A_1, A_2, \ldots, A_n)$, where A_x represents a synthesis directive for a given explorable construct.

Our method starts by randomly generating N designs $D_1 \ldots D_n$, where each design $D_m = \{A_1, A_2, \ldots, A_n\}$ has assigned a unique set of synthesis attributes A, and inserting these directives automatically into the source code. Attributes' combinations for a particular design are referred to as

instances I (inputs) in ML terminology [13]. Each instance is an individual and independent input to be learned. Therefore each new design (D_x) with a unique set of attributes $\{A_1, A_2, \ldots, A_n\}$ is an instant I_x.

Different ML methods were evaluated in order to use the most accurate one for HLS DSE. Table 2 summarises the models evaluated. They cover most of the ML method families: rule based, tree based and function based.

In order to compare the efficiency of the different methods we extended our flow to write out at specified constant intervals all the instances generated $I_1, I_2, \ldots, I_n$ until that particular time (all designs with their area, latency and synthesis directives). A predictive model was generated for each interval for all the instances using all the ML methods indicated in Table 2 (M5P, REPTree, DecisionStump and Linear regression) and their relative error between the model and the actual area and latency computed. Fig. 4 shows the predictive model errors analysis for the CSC test case. From this analysis it could be concluded that the M5P method was consistently the best of all the methods. An interesting observation made during this analysis is that the error is not a monotonically decreasing function. This has some implications on our exploration method as the method cannot proceed with the predictive model exploration stage (step 5) once the error threshold value is reached. It has to guarantee that the predictive model is accurate enough for any combination of results. To tackle this problem it was experimentally observed that the predictive model had to meet the error threshold value (maximum error allowed) three consecutive times for predictive model generation interval of 20 instances, in order to guarantee that the error function is monotonically decreasing. The model's error, difference between the predicted area and latency (A_{ip}, L_{ip}) and the actual area and latency (A_p, L_i) obtained after HLS of design D_i, is computed using a standard 10-fold stratified cross validation method. This implies that the data evaluated until this point are divided randomly into ten parts, where each part is held out in turn and the learning scheme is trained on the remaining nine-tenths as suggested in [13]. For each cross-validation experiment the new independent error estimate is calculated. The mean of all the independent errors is then used to compare the quality of each method.

Based on the error analysis, the M5P method was chosen, as this was the most accurate of all. The M5P method is a model tree learner, essentially consistent of a decision tree with linear models at the leaves.

The most time-consuming part of the flow is the generation of the training set to generate the predictive model. Weka generates an M5P predictive model in milliseconds, whereas the synthesis, model generation and simulation of each new design can take between 10 s, for the smaller test cases, to minutes for the larger ones. This training set generation phase creates random designs with random set of synthesis attributes until the specified error margin is reached. A new predictive model is created at regular intervals and its error is verified. Once the predictive model created leads to errors within the specified error margins, the exploration continues using this predictive model instead of synthesising random design configurations. This part of the exploration is based on a GA as GA has shown in previous work [2, 14] that it can lead to good results. We call this exploration method GA-ML. Its exploration parameters follow the indications given in [14]. The number of populations created is 40, crossover probability of 0.8 and mutation probability of 0.1.

Table 2 Predictive models comparisons

Predictive model	Description
decision stump	simple decision table. One-level decision tree (tree based)
linear regression	standard linear regression (function based)
M5P	regression tree learner (tree based)
REPTree	decision tree (rule based)

4 Experimental results

First, we describe the experimental set-up for the DSE results of our proposed ML method (GA-ML) as compared with a conventional GA method and a previously developed annealer-based method (SA). The SA method has shown to generate good results as compared with a brute force approach and will serve as a good overall quality reference point [9]. Then, we show a set of comprehensive results obtained, together with explanations on the implication and analysis of the data.

4.1 Experimental set-up

Eight different test cases written in C and SystemC (SC) used in in-house designs were chosen to validate our method shown in Table 3. The first benchmark (ave8) computes the average of eight numbers. Gfilter is a graphic filter, CSC, FD_shrink, FD_ISS and FCU are part of a face detection IP, where CSC is the colour space converter, FD_image_shrink the window resizing, FD_ISS the sub-window selection and FCU the feature detection unit. Reed Solomon represents only the encoder part and FPU_mult a floating point multiplication described in C. The first column shows the benchmark's name. The second column indicates if it is a C or SC design. The third column shows the size of the benchmarks denoted by the total number of lines of code. The last column depicts the total number of explorable constructs, where the number

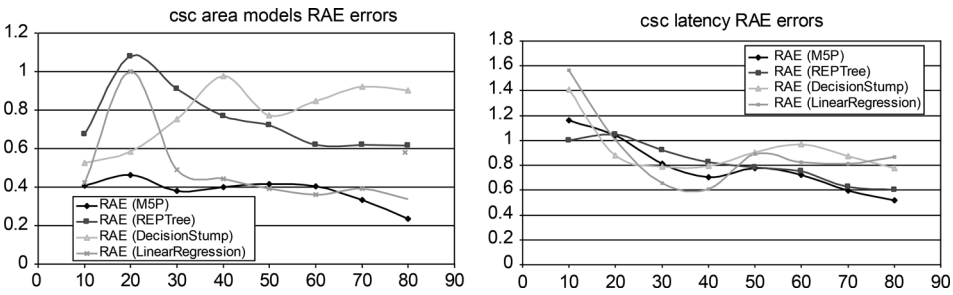

Fig. 4 *Predictive models' errors comparison*

doi: 10.1049/iet-cdt.2011.0115

Table 3 Benchmarks overview

Benchmark	Type	#lines	Explorable operations
ave8	C	58	loop(8), loop(8), array(1)
Gfilter	SC	420	loop(6), loop(7), loop(6), loop(6), array(4), func(4)
Csc	C	644	loop(256), loop(3), loop(d), array(2), func
fd_shrink	C	783	loop(255), loop(256), loop(8), loop(8), loop(d), loop(d), array(2), func(1)
fd_iss	C	357	loop(4), loop(8), loop(20), array(3), func(1)
Fcu	C	2949	loop(20), loop(20), loop(20), loop(20), loop(20), loop(20), loop(20), loop(20), loop(20) loop(8), loop(d), loop(d), loop(d), array(8), func(2)
reed Solomon	SC	530	loop(16), loop(16), loop(7), loop(7), array(8), loop(d), loop(d), loop(d), loop(d), func(4)
fpu_mult	C	720	loop(23), loop(23), loop(23), loop(23), loop(23), loop(23), loop(23), loop(23), array(8), func(4)

in brackets of the loops represent the number of iterations (*d* means data dependent), and the number at the arrays and functions indicate the total number of arrays and functions in each test case.

The main problem when comparing different multi-objective function optimisation methods is how to measure the quality of the results: closeness to the Pareto front, wider range of diverse solutions or other properties. Several studies that address the problem of comparing approximations of the trade-off surface in a quantitative manner can be found in the literature. Most popular are unary quality measures, that is, the measure assigns each approximation set a number that reflects a certain quality aspect, and usually a combination of them is used [15, 16]. A multitude of unary indicators exist, for example, hypervolume indicator, average best weight combination, distance from reference set and spacing. Zitzler *et al.* [17] provide a good review of all existing methods, indicating that there is no any single indicator able to measure the quality of the results. Nevertheless, quality measures are necessary in order to compare the outcome of the DSE. In this work, we follow the guidelines suggested by [17] and measure the quality of the different methods using the following criteria described by [11]

1. *Distance*: This measure (D) indicates how close a Pareto-front is to the reference front. The lower the distance value (D) is, the more similar two Pareto sets are, for example, a high value of maximum distance ($D_{\max}$) suggests that some reference points are not well approximated, and consequently a high value of average distance (Davg) tells us that an entire region of the reference Pareto-front is missing in the approximation set.
2. *Hypervolume*: This index measures the hypervolume of the part of the exploration space that is weakly dominated by the Pareto set to be evaluated. In order to measure this index the exploration space must be bound. In our case, we define the bounding point, as the point which has coordinates in the objective space equal to the highest value obtained.
3. *Pareto dominance*: This index is equal to the ratio between the total number of points in the Pareto set being evaluated, also present in the reference Pareto set. The higher the value, the better the Pareto set is.
4. *Cardinality*: We also report the number of dominating designs found by each method (Cardinality). A high cardinality indicates a larger number of solutions to choose from, which should be considered to be positive, although it needs to be interpreted carefully with the rest of the data.

Two types of analysis are performed, qualitative and quantitative. First, a qualitative analysis between the three approaches (GA-ML, GA and SA) is performed. For GA-ML and GA exactly the same exploration parameters are used (number of populations, designs per population, mutation and crossover rate). This means that the results ideally, if the same seed is used, should be identical for both methods, whereas GA-ML should finish faster than GA. For the quantitative analysis, the exploration runtime is restricted to a given time (half of the exploration runtime of the GA method of the qualitative analysis) and the results are compared. In this case GA-ML should create better results as it has more time to evaluate more combinations.

The experiments were repeated five times in order to minimise the stochastic behaviour of the methods used. We report the average result of all runs. The best results of all three methods are combined to obtain the reference Pareto-front used to compare the different methods. The experiments were run on an Intel Xeon running at 3.20 GHz machine with 3 Gbytes of RAM running Linux

Table 4 DSE experimental results I: runtime comparison

Benchmark	GA-ML			GA		SA	
	Training, s	Exploration, s	Total run, s	Total run, s	Diff	Total run, s	Diff
ave8	407	37	444	2792	6.29	3211	7.23
Gfilter	6035	364	6399	11 266	1.76	12 421	1.94
Csc	7676	229	7904	19 920	2.52	17 654	2.23
fd_shrink	21 173	2838	24011	31 193	1.30	33 584	1.40
fd_iss	2693	106	2799	4323	1.54	5964	2.13
Fcu	81 924	15 234	97 157	127 500	1.31	134 225	1.38
reed solomon	6572	1145	7716	11 512	1.49	13 454	1.74
fpu_mult	12 396	987	13 382	22 821	1.71	21 874	1.63
Geomean			7898	15 187	1.92	16 529	2.09

Table 5 DSE experimental results II: quality assessment

Benchmark	Cardinality			Pareto dominance			Hypervolume, %		
	GA-ML	GA	SA	GA-ML	GA	SA	GA-ML	GA	SA
ave8	4	6	6	0.67	1.00	1	0.97	1.00	1
Gfilter	4	5	5	0.80	0.83	0.83	0.97	0.99	0.99
Csc	5	7	7	0.71	1.00	1	0.94	1.00	1
fd_shrink	4	5	5	0.40	1.00	1	0.93	1.00	1
fd_iss	3	4	4	0.50	1.00	1	0.98	1.00	1
Fcu	7	10	12	0.40	0.93	0.93	0.73	0.95	0.97
reed solomon	5	6	6	0.50	1.00	1	0.63	1.00	1
fpu_mult	5	6	5	0.83	1.00	1	0.98	1.00	1
Geomean	4.63	65.92	5.92	0.62	0.97	0.97	0.89	0.99	0.99

Table 6 DSE experimental results II: quality assessment under runtime constraint

Benchmark	Cardinality			Pareto dominance			Hypervolume, %		
	GA-ML	GA	SA	GA-ML	GA	SA	GA-ML	GA	SA
ave8	5	5	5	0.75	0.75	0.75	0.60	0.61	0.61
Gfilter	4	3	3	1.00	0.50	0.5	1.00	0.78	0.78
Csc	5	5	5	1.00	0.60	0.6	1.00	0.97	0.97
fd_shrink	4	3	3	0.60	0.40	0.4	0.97	0.99	0.99
fd_iss	3	3	3	1.00	0.75	0.75	1.00	0.93	0.93
Fcu	5	7	8	0.71	0.57	0.70	0.96	0.94	0.97
reed solomon	4	5	5	0.40	0.80	0.80	0.73	0.93	0.93
fpu_mult	4	5	5	0.80	0.80	0.80	0.95	0.93	0.93
Geomean	4.20	4.31	4.38	0.75	0.63		0.89	0.88	0.88

Red Hat 3.4.26.fc3 and we used CyberWorkBench [18] for HLS. The running time given comprises the entire exploration process.

4.2 Experimental results

Table 4 shows the runtime comparison between our proposed method and a conventional GA method and the SA method. The average values represent the geometric mean (geomean), as the absolute indicators vary significantly between benchmarks because of their size differences.

It can be observed that our method is faster in all test cases with an average speed-up of $\times 1.92$ compared to the GA method whereas $\times 2.04$ compared to the SA method. It should be noted that the error threshold value chosen dramatically affects the runtime of our method. In this case, we chose a $\Delta E\%$ of 15%, which experimentally showed a good balance between runtime and quality of results. A larger error margin (maximum permissible average error difference between the predicted area and latency against the area and latency after synthesis and simulation for all designs explored so far) significantly increases the speed-up as the training set generation, which is the most time-consuming part, is executed faster, whereas reducing the error threshold value would decrease the speed-up proportionally to the training set size.

Table 5 compares the quality of the two methods. It reports the cardinality, Pareto dominance and hypervolume. Our method finds on average 22% less number of Pareto optimal designs compared to both the GA and SA methods and the Pareto dominance factor is on average 35% lower than the dominating Pareto front obtained by the GA and SA methods. The hypervolume, normalised with respect to the boundaries of the objective space and given in percentage, indicates that our method performs very well with an average of 10% smaller than the GA result and 11% smaller than the SA method. The difference between the results of GA-ML and the other two methods is basically because of the error in the model. Lowering the error threshold value could improve the results, but we observed that in order to reduce the error by 5% we had to increase the training set by almost 25%, reducing the speed-up considerably. In some cases, GA-ML was able to find some dominating designs not found by GA. These designs were mainly created during the training set generation phase.

The second sets of experiments compare the quality of the exploration results if the DSE's runtime is limited to half of the GA runtime shown in Table 4. Table 6 shows that both approaches lead to similar results in terms of cardinality, whereas our method as expected leads to better Pareto dominance and hypervolume values. This is basically because GA-ML can evaluate many more configurations compared to the GA and SA methods.

5 Conclusion

This work presents an ML predictive model HLS designs space exploration method called GA-ML. Our method creates a predictive model based on a dynamic training set that is interrupted when a specified maximum model error is reached and then continues the DSE using a pseudo GA based on the predictive mode, instead of having to re-synthesise each new design being explored. Results show that our method obtains comparable results to a pure GA exploration method with the same exploration parameters while running on average 1.92 times faster. Compared to a

doi: 10.1049/iet-cdt.2011.0115

previously developed simulated annealer-based method, our proposed method also showed comparable results, while executing 2.09 times faster. When the exploration time is restricted to half the exploration time required for a full exploration, our method generates better results than the GA and SA methods as it is able to evaluate more configurations in the given time.

6 References

1 Ahmad, I., Dhodi, M., Hielscher, F.: 'Design-space exploration for high-level synthesis', *Comput. Commun.*, 1994, pp. 491–496
2 Holzer, M., Knerr, B., Rupp, M.: 'Design space exploration with evolutionary multi-objective optimisation', *Proc. Ind. Embedded Syst.*, 2007, pp. 125–133
3 Haubelt, C., Teich, J.: 'Accelerating design space exploration'. International Conference on ASIC, 2003, pp. 79–84
4 Haubelt, C., Teich, J.: 'CHARMED: a multi-objective co–synthesis framework for multi–mode embedded systems'. ASAP, 2004, pp. 28–40
5 Anderson, I.D.L., SKhalid, M.A.: 'SC build: a computer–aided design tool for design space exploration of embedded central processing unit cores for field-programmable gates arrays', *IET Comput. Digit. Tech.*, 2009, pp. 24–32
6 So, B., Hall, M.W., Diniz, P.C.: 'A Compiler approach to fast hardware design space exploration in FPGA-based systems', *IET Comput. Digit. Tech.*, 2002, pp. 165–176
7 So, B., Diniz, P.C., Hall, M.W.: 'Using estimates from behavioral synthesis tools in compiler–directed design space exploration', *DAC*, 2003, pp. 512–519
8 Givargis, T., Vahid, F., Henkel, J.: 'System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2002, **10**, (4), pp. 416–422
9 Carrion Schafer, B., Takenaka, T., Wakabayashi, K.: 'Adaptive simulated annealer for high level synthesis design space exploration'. VLSI DAT, 2009, pp. 509–519
10 Carrion Schafer, B., Wakabayashi, K.: 'Design space exploration acceleration through operation clustering', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD)*, 2010, **29**, (1), pp. 153–157
11 Ascia, G., Catania, V., Di Nuovo, A.G., Palesi, M., Patti, D.: 'Efficient design space exploration for application specific systems-on-a-chip', *J. Syst. Archit.*, 2007, **53**, pp. 733–750
12 Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: 'The WEKA data mining software: an update', *SIGKDD Explor.*, 2009, **11**, (1), pp. 10–18
13 Witten, I.H., Frank, E.: 'Data mining' (Morgan Kaufmann, 2005). ISBN-13:978-0-12-0808407-0
14 Ascia, A., Catania, V., Palesi, M.: 'A GA based design space exploration framework for parameterized system-on-a-chip platforms', *IEEE Trans. Evol. Comput.*, 2004, **8**, (4), pp. 329–346
15 Kalyanmoy, D., Agrawal, S., Pratap, A., Meyarivan, T.: 'A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii', *Parallel Prob. Solving Nat.*, 2000, pp. 849–858
16 David, A., Veldhuizen, V., Lamont, B.G.: 'On measuring multiobjective evolutionary algorithm performance', *Cong. Evol. Comput.*, 2000, **1**, pp. 204–211
17 Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., da Fonseca, V.G.: 'Performance assessment of multiobjective optimizers: an analysis and review', *IEEE Trans. Evol. Comput.*, 2003, **7**, pp. 117–132
18 www.cyberworkbench.com