Divide and Conquer High Level Synthesis Design Space Exploration

BENJAMIN CARRION SCHAFER, NEC Corporation
KAZUTOSHI WAKABAYASHI, NEC Corporation

A method to accelerate the design space exploration (DSE) of behavioral descriptions for high level synthesis based on a divide and conquer method called Divide and Conquer Exploration Algorithm (DC-ExpA) is presented. DC-ExpA parses an untimed behavioral description given in C or SystemC and clusters interdependent operations which are in turn explored independently by inserting synthesis directives automatically in the source code. The method then continues by combining the exploration results to obtain only Pareto-optimal designs. This method accelerates the design space exploration considerably and is compared against two previous methods: an Adaptive Simulated Annealer Exploration Algorithm (ASA-ExpA) that shows good optimality at high runtimes, and a pattern matching method called Clustering Design Space Exploration Acceleration (CDS-ExpA) that is fast but suboptimal. Our proposed method is orthogonal to previous exploration methods that focus on the exploration of resource constraints, allocation, binding and/or scheduling. Our proposed method on contrary sets local synthesis directives that decide upon the overall architectural structure of the design (e.g. mapping certain arrays to memories or registers). Results show that DC-ExpA explores the design space on average 61% faster than ASA-ExpA, obtaining comparable results indicated by several quality indicators e.g. distance to reference Pareto-front, hypervolume and Pareto dominance. Compared to CDS-ExpA it is 69% slower, but obtains much betters results compared to the same quality indicators.

Categories and Subject Descriptors: H.4.0 [**Information Systems Applications**]: General

General Terms: Algorithms, Design

Additional Key Words and Phrases: High level synthesis, design space exploration, acceleration

1. INTRODUCTION

Pressure for faster time to market is growing as IC costs keep increasing and time to recover the investment (time in the market) is decreasing. The key to shorten design cycles is to have tools that convert system level descriptions, in any high level language, into efficient hardware designs as easy and fast as possible.

Currently system designers typically develop their models in high level languages such as C or C++. This allows an easy and fast way to estimate system performance, verify the functional correctness of the design and offers higher levels of abstraction, which helps also for the re-usability of the code. It also offers faster simulations and the possibility to use already existing legacy code and libraries. Hardware designers must then analyze the code, figure out a suitable hardware architecture for the code and either decided to use HLS or create the HW architecture from scratch using any Hardware Description language (HDL). This task is extremely time consuming and needs a great amount of experience as this step will decide upon the ultimate system

B. Carrion Schafer is with NEC Corporation, System IP Core Laboratory, Kawasaki, Kanagawa 211-8666 Japan (phone: +81 44 435 9486; fax: +81 44 435 9491; e-mail: schaferb@bq.jp.nec.com).
K.Wakabayashi is with NEC Corporation, Design Methodology Laboratory, Kawasaki, Kanagawa 211-8666 Japan(e-mail: wakaba@bl.jp.nec.com).

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

performance. Raising the level of abstraction has a distinct advantage over traditional RTL design approaches. Multiple designs can be easily and quickly generated for the behavioral code, while RTL designs require major rework in order to modify the underlying architecture. Moreover higher levels of abstraction combined with high level synthesis allow the architectural trade-off exploration of the behavioral description. The main problem with architectural exploration is its exponential complexity with the number of explorable operation, making it practically impossible to fully explore the entire design space.

Previous work tightly integrates the exploration and the High Level Synthesis (HLS) steps in order to deterministically estimate the effect of each transformation (e.g. loop unrolling) and therefore targeting directly during the exploration the transformation that lead to Pareto-optimal designs. In this work we address the architectural exploration considering HLS as a black box using a commercial HLS tool [NEC 2011]. This HLS tool performs different transformations and applies different heuristics depending on the input description which are unknown before hand, making it impossible to deterministically establish which transformation will lead to better results.

The main objective in design space exploration (DSE) is to find optimal implementations with respect to several, often conflicting, objectives. In this work we restrict the exploration objectives to area and latency, but other objectives like power could also be added. These optimal implementations are called Pareto-optimal designs. The objective is to find all the designs at the efficient frontier (also called Pareto front). The tradeoffs can easily be explored within this set rather than considering the entire design space, which would be impractical and irrelevant to the designer. The main problem in DSE is its exponential nature. A brute force approach would eventually find all the Pareto-optimal solutions for smaller designs at a cost of extremely high running time. Heuristics have been developed to reduce runtime, at the expense of finding less Pareto-optimal designs and some that are not really Pareto-optimal. These designs are called non-dominated designs as they dominate the exploration result obtained so far, but not the overall exploration space. Fig. 1 shows the eventual result of some heuristics classified by regions[1]. Region 1 shows non-dominated designs that are not Pareto-optimal found by the heuristic. Further design space exploration would eventually end up finding the real Pareto designs. Region 2 shows that some heuristics might miss Pareto-optimal solutions and region 3 illustrates that some of the Pareto solutions on the efficient frontier are actually found. The main problem with multi-objective optimization problems is how to proof Pareto-optimality. In this work we can only proof Pareto-optimality for small benchmarks as these where compared against a brute force method. For larger designs this is virtually impossible and we can only create solutions that are non-dominated.We will consider the combination of the best non-dominated designs obtained by all the heuristics presented in this work as the reference Pareto front. [2].

In particular, in this work we present a divide and conquer method to accelerate the design space exploration and compare it with two previously presented exploration methods based on simulated annealing [Schafer et al. 2009] and pattern matching [Schafer and Wakabayashi 2009]. The first method called Adaptive Simulated Annealer Exploration method (ASA-ExpA) leads to a good result finding Pareto-optimal designs compared to a brute force approach at the expense of high running time. The second method called Clustering Design Space Exploration Acceleration (CDS-ExpA) is based on a pattern matching method, finding fix sets of clustered operations and as-

[1]Note that in general these regions are not necessarily adjacent.

[2]The terms non-dominated and dominating design will be used interchangeably. The reference Pareto front is the combination of the non-dominated designs of all three exploration methods.

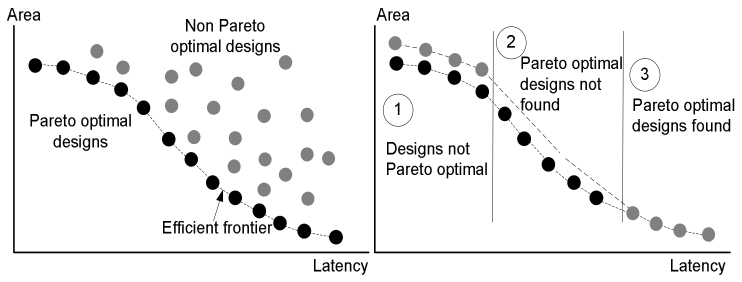


Fig. 1. Efficient frontier with Pareto-optimal designs and design space exploration results.

signing a fixed set of attributes to each of them based on a global cost function (GCF). This method is extremely fast at the expense of not finding many of the Pareto-optimal designs. These two methods will help assessing the quality of the exploration results obtained by our newly proposed method as the first is optimized for quality while the latter for runtime.

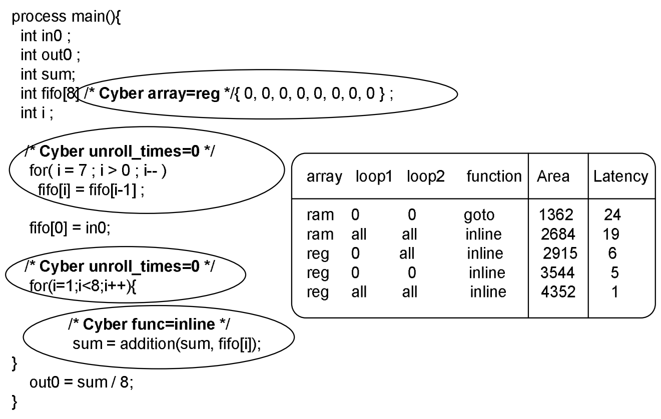
The contributions of this work can be summarized as follows:

(1) Introduce a design space exploration method called Divide & Conquer Exploration Acceleration (DC-ExpA) to accelerate the design space exploration for behavioral descriptions given in C or SystemC. DC-ExpA clusters all interdependent explorable operations and explores these separately. It then merges the results in order to obtain the Pareto-optimal designs.
(2) Investigate the effect of using the latency reported by the HLS tool vs. simulation cycles on the quality of the optimal designs obtained compared to the exploration runtime.

The paper is organized as follows. Section II shows a motivational example to illustrate the goal of DSE. Section III presents a comprehensive literature review of high level synthesis architectural exploration. Then in section IV we introduce the two previously developed methods and describe in detail the newly proposed method to accelerate the design space exploration. Section IV provides a set of experimental results to show the efficiency of our method. Finally section V gives concluding remarks.

2. MOTIVATIONAL EXAMPLE

Fig. 2 shows the source code of a process that continuously reads in 8 bit numbers and calculates the average of the last 8 values read (this program corresponds to benchmark ave8 of the experimental section). The explorable operations have been highlighted and consist of an array where the last 8 numbers are stored, 2 loops and 1 function. These operations are considered *explorable* because they can be synthesized in different ways resulting in very different area vs. performance implementations e.g. the loop can be unrolled partially or completely and the array can be mapped to registers or memory. The table next to the source code shows the result of the HLS for different synthesis attributes specified directly at the highlighted explorable operations using pragmas. As seen, the difference between the smallest but slowest design and the fastest but largest is substantial, with the area ranging from 1,362 to 4,352 and the latencies from 24 to 1 cycles. There are a multiple Pareto-optimal combinations in-between these designs based on different attribute combinations as well as sub-attributes like the number of memory ports in the array, but only 5 combinations are shown here for practical reasons. Manually editing the source code in order to explore the different area vs. performance trade-offs is a tedious and time consuming task. An automatic efficient design space exploration method is therefore highly de-



```
process main(){
  int in0 ;
  int out0 ;
  int sum;
  int fifo[8] /* Cyber array=reg */{ 0, 0, 0, 0, 0, 0, 0, 0 } ;
  int i ;

  /* Cyber unroll_times=0 */
   for( i = 7 ; i > 0 ; i-- )
    fifo[i] = fifo[i-1] ;

   fifo[0] = in0;

  /* Cyber unroll_times=0 */
   for(i=1;i<8;i++){

      /* Cyber func=inline */
      sum = addition(sum, fifo[i]);
}
   out0 = sum / 8;
}
```

array	loop1	loop2	function	Area	Latency
ram	0	0	goto	1362	24
ram	all	all	inline	2684	19
reg	0	all	inline	2915	6
reg	0	0	inline	3544	5
reg	all	all	inline	4352	1

Fig. 2. Motivational example.

sirable. The main problem in DSE is how to explore the design space in a reasonable time, finding as many Pareto-optimal designs as possible.

3. RELATED WORK

In order to deal with shorter time to market design cycles, high level languages extended with hardware specific constructs are being used in combination with high level synthesis tools. Some examples of C/C++ extensions include SystemC [SystemC 2011], BDL [Kobayashi et al. 1994], SA-C [SA-C 2011] and SpecC [Gajski et al. 2005] and JHDL for Java [JHDL 2005]. There are currently a multitude of commercial and academic HLS tools available. Some commercial ones include CyberWorkBench [NEC 2011], CatapultC [Calypto 2011], Cynthesizer [Forte 2011], CtoSilicon [Cadence 2011] and some academic ones e.g. LegUp [LegUP 2011] (open source) or open binaries e.g xPilot [xPilot 2011], C2See [C2See 2011], GAUT [Gaut 2011] and Spark [Gupta et al. 2003].

These high level language subsets simplify the design process as designers do not need to deal with low level Hardware Description Languages (HDLs). However, in order to efficiently synthesize behavioral descriptions with any of the previously mentioned HLS tools, designers still have to analyze the original description to specify e.g. bit widths, parallelism, operator binding and array mappings. The design space exploration does this step automatically generating a number of designs that meet a set of constraints (i.e area, latency and power).

Much of the previous research has been focused on system level design exploration where the number and the type of processing elements and bus size are explored [Haubelt et al. 2008; Kim et al. 2006;Mamagkakis et al. 2006]. Givargis et al. [2002] proposed a system-level exploration technique for 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. Although we use a similar approach, they target a fixed architecture and the design space is relatively small (1014 possible configurations). Moreover the designer has to manually specify these interdependencies among parameters, which indicates the explorer the direction to follow.

We call this macro-architectural design space exploration vs. micro-architectural which is the method presented in this work. Previous work in HLS design space exploration has been focused on the exploration of resource constraint and/or scheduling, binding and allocation algorithms or on applying source code transformation starting from CDFGs using multi-objective function optimizations. Ahmad et al.[1994] studied the tradeoffs between the control step and area in data flow graphs using genetic algorithms. Holzer et al.[2007] used a similar approach using an evolutionary multi-objective optimization approach to generate Pareto-optimal solutions. Krishnan et al. [2006] use a genetic algorithm for HLS DSE to concurrently perform scheduling and allocation to find better schedules. Haubelt et al. [2003] use Pareto-Front-Arithmetics (PFA) to reduce the search space in embedded systems decomposing a hierarchical search space. Early estimators of area and delay for FPGA implementations where used in [Haubelt and Teich 2004] to evaluate the design space before any behavioral synthesis. A compiler approach to perform hardware design space exploration is presented in [So et al. 2002] where parallelization techniques are used to map computations to FPGAs. So et al. [2003] developed a DSE technique using compiler directed techniques to perform several code transformations. The starting point in most of these approaches is the direct transformations at the CDFG level applying different compiler optimizations techniques to generate new architectures combined with quick estimators. Most of these approaches mainly use Genetic Algorithms (GA) as GA is easy to implement and has shown to produce good results.

In this work we explore the design space of untimed behavioral descriptions using a commercial high level synthesis tool [NEC 2011] seen as a black box by inserting synthesis directives directly into the source code. We do not have access to it and execute it every time a new set of directives is generated. Previous work estimates the impact of each transformation as they have full control over the resultant synthesized circuit and can deterministically establish the cost of each transformation. The number of transformations allowed in these cases is very limited and are normally restricted to the number and/or type of functional units (FUs) and in some cases to loop unrolling. Our proposed approach could be considered orthogonal to the previous work, as our method fixes the underlying architecture e.g. if arrays are mapped to memories or registers or decided upon how many function instantiations the process should have. Then, previous DSE approaches could continue by exploring the resource constraints, scheduling, allocation and binding stages for each fixed synthesis scenario created by our explorer.

The commercial HLS tool used in this study has over 200 different synthesis directives (pragmas for individual operations) in addition to global synthesis options and behaves differently in each case based on different internal heuristics. The design space grows exponentially with the size of the explorable operations and therefore cannot be exhaustively searched in reasonable time. Moreover, this make is impossible for the explorer to accurately model and predict the behavior of the HLS tool, which leads to the need to adopt a different approach.

One further element of study in this work is how *performance* is measured. In most high level synthesis DSE work, either control unit state count or latency reported by the HLS is used to measure the circuit performance. This is only true in some cases. In other cases e.g. in designs with unbounded loops or data-dependent loops, this does not hold true anymore and a cycle accurate simulation is needed in order to measure the design's performance. In this work we used the latency reported by the HLS tool in order as a measure of design performance. In order to study the impact of this assumption on the quality of the DSE results we extended the flow generating a cycle accurate C model for each design explored and executing it in order to obtain actual design performance information. We compare the default vs. the extended method and investigate the results in the experimental section.

Table I. Explorable Operations

Operation	Attribute	Description
Loop	unroll=0	do not unroll the 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
	func=pipeline_opr	function inst as pipeline opr
	func=operator	function treated as a function unit
Array	array=RAM	array synthesized as memory
	array=logic	constant arrays synth. as logic
	array= expand	expand array
	array=reg	synthesize array as registers

4. DESIGN SPACE EXPLORATION

The design space exploration method proposed in this work generates a set of non-dominated designs for a given behavioral description in untimed C or SystemC 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 synthesizes the instrumented source code accordingly. In particular our method explores loops, arrays and functions. Table I shows all the explorable operations and their pragmas. A more comprehensive explorer could also explore global synthesis options and the number of functional units. The goal of the exploration is to find as many designs as possible on the efficient frontier, also called Pareto front, or as close as possible. The tool developed around this exploration methods is called cwbexplorer. Fig. 3 shows an overview of the exploration flow. The inputs are a C or SystemC description and the delay and area library for the chosen target technology. The behavioral description is read and a new unique set of attributes is generated for the explorable operations found in the source code. The global synthesis options, which are kept constant throughout the entire DSE, the newly instrumented source code and a functional unit constraint file are then passed to the HLS tool which then synthesizes the new designs. The result of the synthesis is read back by cwbexplorer to analyze the synthesis results (area and latency). cwbexplorer has been extended to run in two modes: The first uses the latency reported by the HLS tool as a measure of design performance. The second mode calls a cycle accurate model generator right after HLS, which generates a cycle accurate C++ model for the design just synthesized. It then calls g++ to compile this model and executes the binaries. The simulation cycles are read back by cwbexplorer to get an accurate measure of the performance of each design. Therefore, the dominating results obtained running the cycle accurate simulation are the true dominating designs as the main objective of DSE is to find area vs. performance tradeoffs and the design latency reported by the HLS tool is only a proxy for performance.

The next two sections will describe briefly the previously developed methods ASA-ExpA and CDS-ExpA,which will be used to benchmark our proposed method. Then our newly proposed method will be described in detail. The first method is based on an adaptive simulated annealer [Schafer et al. 2009] and the second on a pattern matching method of explorable operations clusters assigning a fix set of attributes to these clusters based on a global cost function (GCF) state [Schafer and Wakabayashi 2009]. Both previously developed methods will allow us to asses the quality of the results of our new method as ASA-ExpA has shown to generate good results compared to a brute force approach at the expense of high runtime. On the other side CDS-ExpA has shown to be 90% faster on average than ASA-ExpA at the expense of only finding around 50% of the Pareto-optimal designs.

4.1. Adaptive Simulated Annealer Design Space Exploration (ASA-ExpA)

The ASA-ExpA method is based on adaptive simulated anealing. Each attribute specified in the input library has an associated weight based on its ability to *typically* contribute either to minimize latency or area. These have been previously pre-characterized, but can also be manually set by the user. A unique set of attributes is generated for each new design. Each new design is incrementally generated from the previous design based on the given Global Cost Function (GCF). The GCF of the simulated annealer is given by GCF = $\alpha A + \beta L$, where the area (A) and latency (L) are scaled due to their units' differences. The weight factors α and β represent the importance of minimizing the total area (A) or total latency (L). These weights are adaptively modified during the exploration in order to explore the entire design space, when after N consecutive designs no dominating design is created. If $\alpha >> \beta$, attributes with highest minimizing area weights have a higher probability of being used. The randomness of ASA-ExpA allows it to escape from local minima.

The GCF weights are initially set to find designs with the objective of minimizing area (α=10 and β=0). A unique new set of attributes is created based on the GCF and each attribute's weight, which are used as probabilities by the annealer. The HLS tool is then called to synthesize the new design with the new attributes set. After the synthesis ASA-ExpA verifies if all the constraints have been met. If the design has not improved the previous designs (area or latency) the ASA-ExpA will either restore the previous attribute list or based on the current annealer *temperature* keep it.

The exploration finishes when either N designs are consecutively generated that do not improve the previous designs or do not meet the constraints, where N is specified by the user or when the GCF has explored the entire design space starting from (α=10,β=0),(α=8,β=2),(α=6,β=4) to (α=0,β=10). The exploration is bounded by $O(a^M)$ where a is the number of explorable attributes for each operation and M is the number of explorable operations in the source code.The most time consuming step in the procedure is the synthesis performed for each design. The running time for the attribute generation is negligible compared to the synthesis. This method has shown to create very good results compared to a brute force method at the expense of high running time [Schafer et al. 2009]. For small and mid size benchmarks it could find all Pareto-optimal designs, while for larger benchmarks it created very close Pareto-fronts. It should be mentioned that in this case we can proof their optimality as we compared this method against a brute-force search.

4.2. Pattern Matching Design Space Exploration (CDS-ExpA)

CDS-ExpA is based on a pattern matching method assigning a fix set of attributes to explorable operations clusters. The method starts by parsing the input C or SystemC code and building a dependency tree of all the explorable operations (loops, functions and arrays). Fig.4 shows the parse tree of the motivational example (ave8). The parse tree is then traversed using a breadth first search method and a tree pattern matching algorithm applied to find subgraphs matching a given external operation clusters library. The clusters library contains the sequences of operations that form a cluster and the set of pre-defined attributes associated to each cluster for different optimization targets e.g. reduce area or latency. The cluster types are all possible 2-3-4-tuples combination of arrays, functions and loops. A Global Cost Function (GCF) indicates what exploration objective should be minimized. The GCF changes dynamically during the exploration and the attributes assigned to each operation are re-assigned to minimize the GCF objective.

Two version of CDS-Exp have been developed. The first, called CDS-ExpA(min) tries to find subgraphs of operations as small as possible. This approach reduces the design

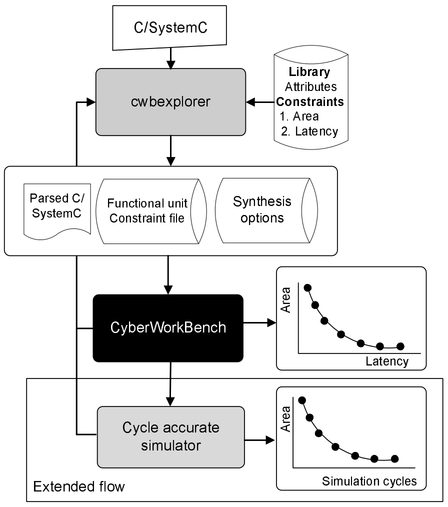


Fig. 3. Exploration flow overview.

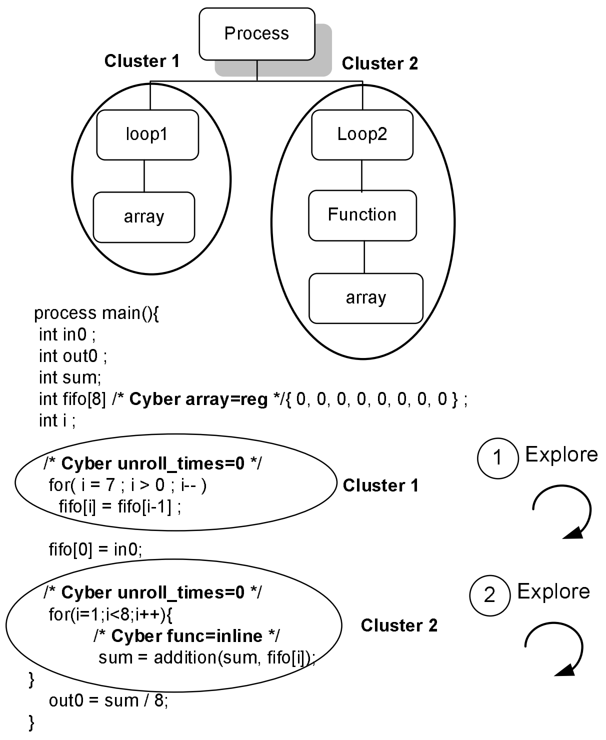
space compared to the ASA-ExpA method, while at the same time still allows a large number of cluster combinations combined with non-clustered operations that leads to a smaller probability of missing Pareto-optimal designs. The second approach called CDS-ExpA(max) findest the largest possible subgraphs reducing the design search space compared to the CDS-ExpA(min) approach and hence accelerating the design space exploration even further at the expense of missing more Pareto-optimal designs and generating more non-Pareto-optimal designs. In this work we will use CDS-ExpA(max) to compare with our method as this as this version leads to the smallest running times.

The fixed attributes assigned to each subgraph have been empirically determined based on the study of the typical impact on each attribute on the synthesis on a set of different benchmarks. In order to avoid local minima a probabilistic component is added to the attribute assignment of each cluster, allowing clusters to be assigned attributes that do not minimize the GCF objective. The GCF has 3 states. Minimize area, minimize latency and an intermediate state. Each state has a unique set of attributes for each cluster stored in the external cluster library. These states target the exploration of Pareto solutions that minimize area, minimize latency and intermediate points in the curve. If after N designs no more Pareto designs are found the exploration moves to a new state updating the cost function and re-assigning the clusters' attributes.

After each HLS CDS-ExpA checks if the design is non-dominated. If it is, it is deleted. The exploration continues until no more Pareto-optimal designs are found or a given exit criteria is met e.g. N number of non-optimal designs are created consecutively.

4.3. Divide and Conquer Design Space Exploration (DC-ExpA)

DC-ExpA starts similarly as CDS-ExpA by parsing the untimed behavioral description (C or SystemC) and building the dependency parse tree of all explorable operations (hierarchical decomposition). It continues by clustering every set of interdependent operations i.e. all operations on the same parse tree leaf. These are explorable constructs that affect each other if their synthesis type changes, e.g. an array in a loop. Fig. 4 shows the parse tree of the motivational example (ave8) and the clusters built. Each cluster is explored independently, by applying all attributes' combination only to op-



```
process main(){
 int in0 ;
 int out0 ;
 int sum;
 int fifo[8] /* Cyber array=reg */{ 0, 0, 0, 0, 0, 0, 0, 0 } ;
 int i ;
 /* Cyber unroll_times=0 */
  for( i = 7 ; i > 0 ; i-- )
   fifo[i] = fifo[i-1] ;

  fifo[0] = in0;

 /* Cyber unroll_times=0 */
  for(i=1;i<8;i++){
        /* Cyber func=inline */
         sum = addition(sum, fifo[i]);
}
  out0 = sum / 8;
}
```

Fig. 4. Parse tree of ave8 example, clusters and exploration order.

erations in the cluster being currently explored. No synthesis attributes are assigned to the *explorable* operations in the other clusters. Once all combinations have been exhausted or a given exit criteria is reached the exploration continues with the next cluster. E.g. in Fig. 4 cluster 1 would be explored first by creating combinations of attributes for the loop and the array until the exit condition is reached. During this time no synthesis attributes are assigned to the explorable operations in the second cluster. Our method continues exploring cluster 2 once cluster 1 has been fully explored. After all clusters have been explored independently DC-ExpA proceeds by merging the results of the local search combining the attributes of all clusters together to obtain the optimal designs. This step evaluates the result of combining the attributes based on the synthesis result of each design previously generated and determining if it could lead to an optimal design. Out method generates a new design by combining all attributes together and re-synthesizing it. Basically the method tries to minimize the latencies and areas of each individual cluster separately in order to select the combination of attributes that lead to the global Pareto-optimal designs. Fig. 1 summarizes the procedure of our design space exploration method. The method consists of 4 main steps:

Step1: Subgraph Construction: For the given untimed C or SystemC source code SC a dependency parse tree T with the explorable operations E (loops, functions and arrays) is built. All interdependent operations are then clustered together in a subgraph S_i as any synthesis directive (attribute A) applied to any of the clustered operations will only impact the operations in this same cluster. Exploring independent cluster operation limits the effect of the HLS transformations to the cluster itself . The

ALGORITHM 1: Summary of our design space exploration method, DC-ExpA, for exploring Pareto-optimal designs.

```
DC-ExpA: Design and Conquer Exploration($SC$,$E$)
/* $SC$: Source code for HLS (untimed C or SystemC)
   $E$: Library with explorable operations their attributes
/* Step 1 Cluster construction */
• Parse source code($SC$) and generate parse tree ($T$)
   for all explorable operations $E$ (loops, arrays, functions);
• Create clusters $S$ of interdependent set
   of explorable operations;
/* Step 2  Cluster exploration */
foreach(cluster ($S_i$)) do /* explore each clusters */
  do {
      • Generate a new attribute set ($AL_i$) for this cluster;
      • Insert attribute list ($AL_i$) in source code;
      • Perform HLS on the instrumented source code;
      • Backannotate HLS results (area and latency);
  }while(AL != 0); /* while unique sets of attrs can be generated */
endforloop;
/* Step 3  Create non-dominated designs*/
do {
      • Combine attr lists ($ALs$) of clusters to build new Pareto;
      • Insert attributes in source code;
      • Perform HLS of the instrumented source code;
      if(exit condition==true) /* exit condition set by user*/
          break;
}while(($P$)); /*while new Pareto designs are created*/
/* Step 4  Delete dominated designs */
• Analyze all generated designs
   and delete all dominated designs;
return Pareto Optimal designs (($P$));
```

only explorable operation that can have a global effect on the total synthesis are arrays accessed by different clusters and shared functions. We define terms:

Definition: *Subgraph* (cluster) of parse tree T of all explorable operations E (Table I column 1), is a subset S of $\forall E$ in T that satisfies: (i) for each $S_i = \{E_1, E_2, ..., E_x\}$ such that $T = \{S_1, S_2, ..., S_z\}$ there is a $E_i \in S$ such that E_i is unique in T, (ii) if $S_1 \cap S_2 \neq 0$ then $\forall E$ in $S_1 \cap S_2$ have the same A and (iii) each E gets assigned a single A. While exploring single subgraphs S_i the ∀E∈ $T - S_i$ have no A assigned to it. It should be noted that the HLS tool has built in a default synthesis behavior for each E meaning that all unassigned E have assigned the default A to it.

Step2: Clusters Exploration: Our method explores each subgraph S separately generating a unique new set of attributes $\{A_1, A_2, ..., A_n\}$ for each operation E in the given subgraph, performing HLS on the entire design for each new combination. The result (area and latency) of the HLS is backannotated and stored with each unique set of attributes. This step is repeated until either all combination of attributes are exhausted or a given exit criteria is reached e.g. max number of combinations per cluster given by the user. The method continues until all existing cluster are explored. Formally defined:

Exploration of S_1, S_2,..., S_n is performed separately obtaining a set of designs such that $S_1 = \{d_{1,1}, d_{1,2}, ..., d_{1,x}\}$, $S_2 = \{d_{2,1}, d_{2,2}, ..., d_{2,y}\}$,..., $S_n = \{d_{n,1}, d_{n,2}, ..., d_{n,z}\}$, where $d_{1,1}$

is the first design belonging of cluster 1, such that each design has a unique list of attributes $d_{i,j} = \{A_1, A_2, ..., A_y\}$ for each $E \in S$.

Step3: Creating Pareto-optimal designs: This step merges all the attributes $A_1, A_2, ..., A_n$ of a design $d_{i,j}$ generated during subgraph S_i exploration in order to get Pareto-optimal P designs. This step is based on a greedy algorithm. First, the smallest design $d_{n,m}$ based on the combination of all clusters attribute is found. This serves as the initial Pareto reference design. The method recursively finds new Pareto-optimal designs closest to the previously found Pareto design. The closest Pareto-optimal design candidate is built by merging the list of attributes of each design, where $d_{new} = \{A_1, A_2, ..., A_n\} = d_{i,j} \cup d_{k,l}, ... \cup d_{y,z}$. This procedure continues until all Pareto-optimal designs are found. It is important to stress that the merging stage only considers designs explored, which have the same attributes for all shared explorable operations (shared arrays and functions).

Step4: Delete non-dominating designs: The last step involves deleting all non-dominating designs. In some cases (especially for small benchmarks) some designs obtained during the cluster exploration are actually Pareto-optimal. In this case these have to be included into the solution space. For this purpose a postprocessing step is needed to check all generated designs for Pareto optimality and non Pareto-optimal designs are deleted.

The main weakness of DC-ExpA is the estimation part in step 3, where of the impact of the combined set of attributes on the final design is estimated. Based on this estimate a set of attributes is chosen over the other sets. In many cases the combination of all attributes allows the HLS to do some global optimizations that cannot be captured by our method leading to the generation of designs that are sub-optimal.

The most time consuming part in DC-ExpA is step 2 and in particular the inner while-loop which is bounded by $O(p^n)$, where p is the number of explorable attributes for each operation and n is the number of explorable operations in the cluster. Although the order of complexity is exponential, n is normally small between 1 to 3. The total exploration runtime is bounded by $O(kp^n)$, where k is the number of clusters. In the worst case when k=1, our method has the same order of complexity as an exhaustive approach. In the best case when p=1 the order of complexity is linear as the runtime is equal to the summation of the exploration of each cluster, which is in turn only composed of a single operation.

5. EXPERIMENTAL RESULTS

First, we describe the experimental setup for the evaluation of our proposed method. Then, we show a set of comprehensive results obtained, together with the explanation and implication of the analysis of the data.

5.1. Experimental Setup

10 different test cases written in C and SystemC are used to validate our method. 8 are used at in-house designs and 2 taken from the CHStone open source HLS benchmarks suite [Hara et al. 2009] for reproducibility purpose, shown in Table II. The first benchmark (ave8) corresponds to the motivational example case. Gfilter is a graphical filter, CSC, FD_shrink, FD_ISS and FCU are part of a face detection IP, where CSC is the color space conversion, 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. Adpcm is a adaptive differential pulse code modulation decoder and encoder and aes the advanced encryption standard. These benchmarks belonging to the CHStone suite were modified to make them synthesizable by the HLS tool used in this work.

Table II. Attributes of explorable operations

Bench	Type	#lines	explorable operations
ave8	C	58	loop(8),loop(8),array(1)
gfilter	C	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	2940	loop(20),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)
adpcm	C	547	loop(10), loop(22), loop(10), loop(6), loop(6), loop(24), loop(11), loop(30), loop(6), loop(100), loop(100), loop(100), array(9), func(12)
aes	C	723	loop(14), loop(16), loop(d), loop(13), loop(32), loop(16), loop(8), loop(4), loop(8), loop(8), loop(4), loop(33), loop(4), array(3), func(9)

The first column shows the benchmark's name. The second column indicates if it is a C or SystemC (SC) design. The third column shows the size of the benchmarks denoted by the total number of lines of code. It should be noted that 1 line of code of a high level language description is approximately equivalent to 10 lines of RTL code [Coussy and Moraweic 2008]. The last column depicts the total number of explorable constructs, where the number 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.

We compare our proposed method DC-ExpA with two previously presented methods: ASA-ExpA based on an adaptive simulated anealer based method and CDS-ExpA(min) based on the clustering of operation and assignment of fixed attributes. For each method the number of non-dominated designs found, the complete exploration runtime, the number of gates of the smallest design and the latency of the fastest design are recorded. As these methods are probabilistic they are executed 5 times using different random seeds. The best results of all three methods are combined to obtain the reference Pareto-front used to compare the different methods. The running time reported for each test case is the mean of all 5 runs.

The HLS tools reports the maximum, the minimum and the average latency. Our method uses the average latency to measure performance, which is indicated here. As the anealer can take extremely long time to run, especially for the larger benchmarks, it was decided to exit the exploration if after 100 newly generated designs no new Pareto-optimal design is found as this number was found experimentally to be a good compromise between runtime and quality of results.

The main problem when comparing different multi-objective function optimization 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 can be found in the literature that address the problem of comparing approximations of the trade-off surface in a quantitative manner. Most popular are unary quality measures, i.e. the measure assigns each approximation set a number that reflects a certain quality aspect, and usually a combination of them is used ([Kalyanmoy et al. 2000], [David et al. 2000]). A multitude of unary indicators exist e.g. hybervolume indicator, average best weight combination, distance from reference set and spacing. Zitzler et. al provide

Table III. Experimental results

				ASA-ExpA		
Bench	Type	# lines	# Pareto	Run (s)	Min Gates	Min Latency
ave8	C	58	6	2,792	1,496	2
gfilter	C	420	5	11,266	10,461	3
CSC	C	644	7	19,920	18,764	5
FD_shrink	C	783	6	31,193	45,996	17
FD_ISS	C	357	4	4,323	18,882	6
FCU	C	2,940	8	127,500	356,545	21
Reed Solomon	SC	530	7	11,512	8,327	7
FPU mult	C	720	6	22,821	153,039	5
adpcm	C	547	3	27,599	8,974	4
aes	C	723	6	54,547	98,820	7
Geomean			6	18,321	25,187	6

			CDS-ExpA(min)			
ave8	C	58	3	78	1496	2
gfilter	C	420	4	428	11898	3
CSC	C	644	3	561	20837	5
FD_shrink	C	783	3	2106	65743	18
FD_ISS	C	357	4	180	19167	6
FCU	C	2940	5	31517	356545	21
Reed Solomon	SC	530	3	328	13,740	7
FPU mult	C	720	1	3,169	186,711	5
adpcm	C	547	2	7,887	10,574	4
aes	C	723	4	9,788	98,820	7
Geomean			3	1,335	29,174	6
Ratio CDS/ASA			0.51	0.07	1.16	1.01

			DC-ExpA			
ave8	C	58	6	349	1,496	2
gfilter	C	420	5	2,163	10,461	3
CSC	C	644	7	2,317	18,764	5
FD_shrink	C	783	5	3,780	45,996	17
FD_ISS	C	357	4	1,364	19,005	6
FCU	C	2,940	10	73,455	356,545	21
Reed Solomon	SC	530	6	8,346	8,327	7
FPU mult	C	720	3	14,190	154,787	5
adpcm	C	547	3	23,142	8,974	4
aes	C	723	5	12,885	98,820	7
Geomean			5	5,456	25,232	6
Ratio DC/ASA			0.88	0.30	1.0	1.0
Ratio DC/CDS			1.71	4.09	0.86	0.99

a good review of all existing methods in [Zitzler et al. 2003], indicating that there isn't 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 suggest by [Knowles et al. 2006]. In our case we measure the quality of the different methods using the following criteria described in [Ascia et al. 2007]:

(1) *Distance*: This measure (D) indicates how close a Pareto-front is to the reference front. It can be defined as follows:

$$\boldsymbol{D}_{avg.} = \sum_{\forall x_i \epsilon P} min_{\forall y_j \epsilon P}(d(x_i, y_i))$$

$$\boldsymbol{D}_{max.} = max_{\forall x_i \epsilon P}(min_{\forall y_j \epsilon P}(d(x_i, y_i)))$$

Table IV. Experimental results. Relative comparison between previous methods and DC-ExpA.

Bench	# Pareto DC-ASA [%]	# Pareto DC-CDS [%]	Runtime DC-ASA [%]	Runtime DC-CDS [%]	Min Gates DC-ASA [%]	Min Gates DC-CDS [%]	Min Latency DC-ASA [%]	Min Latency DC-CDS [%]
ave8	0.0	50.0	-700.0	77.65	0.0	0.0	0.0	0.0
gfilter	0.0	20.0	-420.85	80.21	0.0	-13.74	0.0	0.0
CSC	0.0	57.14	-759.73	75.79	0.0	-11.05	0.0	0.0
FD_shrink	-20.0	40.0	-725.21	44.29	0.0	-42.93	0.0	-5.88
FD_ISS	0.0	0.0	68.45	-216.94	-0.65	-0.85	0.0	0.0
FCU	20.0	50.0	42.39	-73.58	0.0	0.0	0.0	0.0
Reed Solomon	-16.67	50.0	27.5	-37.93	0.0	-65.01	0.0	0.0
FPU mult	-100.0	66.67	37.82	-60.82	-1.14	-20.62	0.0	0.0
adpcm	0.0	33.33	16.15	-19.26	0.0	-17.83	0.0	0.0
aes	-20.0	20.0	76.38	-323.34	0.0	0.0	0.0	0.0

Table V. DSE Quality Results Assessment

	Distance Avg(%)/Max(%)		
Bench	ASA-ExpA	CDS-ExpA(min)	DC-ExpA
ave8	0.00/0.00	1.25/4.54	0.00/0.00
gfilter	0.00/0.00	0.68/0.68	0.00/0.00
CSC	0.04/0.04	2.45/5.67	0.89/0.89
FD_shrink	0.13/0.13	3.24/12.25	0.22/0.64
FD_ISS	0.25/0.25	5.21/10.45	1.20/1.20
FCU	0.85/4.67	4.12/14.32	0.64/3.76
Reed Solomon	0.00/0.00	3.74/10.36	0.78/4.34
FPU_mult	0.00/0.00	2.31/7.25	0.59/5.67
adpcm	0.00/0.00	10.23/10.23	0.00/0.00
aes	3.56/9.56	8.36/25.53	0.02/0.02
Avg.	0.48/0.1.47	4.16/10.13	0.54/1.65

	Hypervolume (%)		
Bench	ASA-ExpA	CDS-ExpA(min)	DC-ExpA
ave8	0.00	50.47	0.00
gfilter	0.00	20.64	0.00
CSC	19.34	62.87	18.74
FD_shrink	16.52	78.67	23.56
FD_ISS	13.12	72.35	14.27
FCU	14.24	65.88	11.45
Reed Solomon	0.00	63.71	20.20
FPU_mult	0.00	47.99	24.45
adpcm	0.00	26.56	0.00
aes	27.63	82.41	6.67
Avg.	9.09	57.82	11.83

	Pareto Dominance (%)		
Bench	ASA-ExpA	CDS-ExpA(min)	DC-ExpA
ave8	100.00	50.0	100.00
gfilter	100.00	80.00	100.00
CSC	88.89	33.33	88.89
FD_shrink	83.33	16.67	66.67
FD_ISS	80.00	20.00	80.00
FCU	81.82	27.27	90.91
Reed Solomon	100.00	28.57	85.71
FPU_mult	100.00	50.00	66.67
adpcm	100.00	66.67	100.00
aes	50.00	16.67	86.22
Avg.	88.40	38.92	86.22

where x_i and y_j are vectors whose size are equal to the number of objectives M and *d()* is the Euclidean distance. The lower the distance value (D) is, the more similar two Pareto sets are. E.g. 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 ($D_{avg.}$) 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 weekly 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. The smaller the value, the higher the quality of the result is.

(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) *Miscellaneous Indicators*: 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 results. We also report the smallest and the fastest design. These designs are important, because these are the designs that in most of the cases will be finally chosen by the user.

The results reported where computed by comparing the Pareto-front of each method compared to the reference Pareto-front(the combination of the best non-dominated results of each method over the 5 runs).

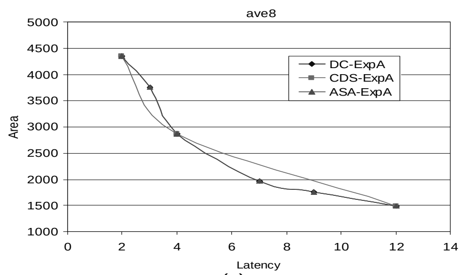
Another set of experiments presented in this section is the comparison of the dominating designs found by our proposed default method which uses the average latency reported by the HLS tool compared to the exploration results when a cycle accurate simulation is performed during the exploration using the simulation cycles as design performance measure.

For these tests the same benchmarks where re-run in simulation exploration mode using 1,000 test vectors used for the verification of these designs. It is important to use real data as random data could miss-capture the real circuit's behavior. Control signals need to be synchronized; if not we risk the case that designs iterate through the same state. The HLS tool used in this work includes a cycle accurate simulation model generator, which generates a cycle accurate C++ model and the make files for each newly generated design. The compiler (g++) is in turn called by our method in order to create the simulation model binary, execute it and read back the simulation cycles.

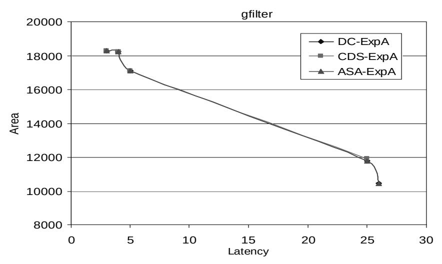
The experiments were run on an Intel Xeon running at 3.20GHz machine with 3Gbytes of RAM running Linux Red Hat 3.4.26.fc3 and we used CyberWorkBench [Kobayashi et al. 1994] for HLS. The running time given comprises the entire exploration process including the HLS.

5.2. Experimental Results

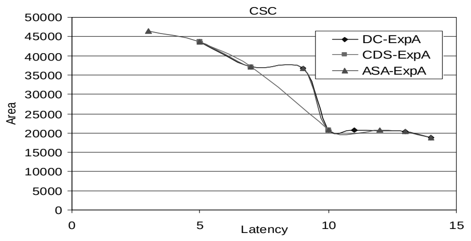
Table III and Table IV show the results of the different DSE methods. The average values in Table III represent the geometric mean (geomean), as the absolute indicators vary significantly between benchmarks, due to their size differences. The ratio between



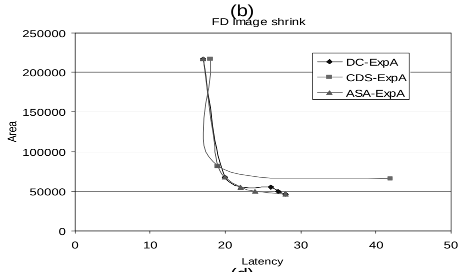
(a)



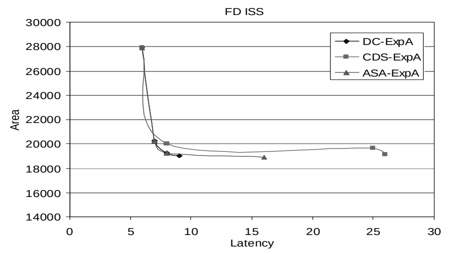
(b)



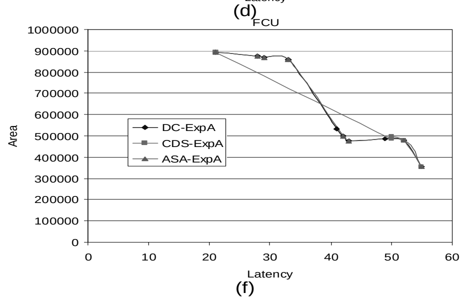
(c)



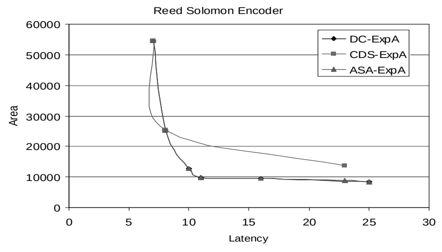
(d)



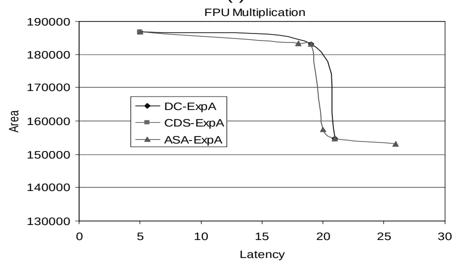
(e)



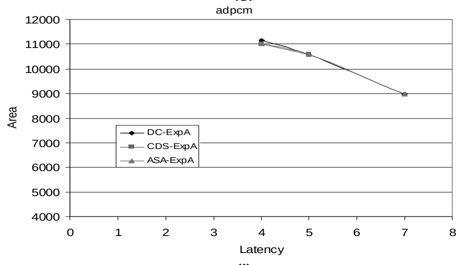
(f)



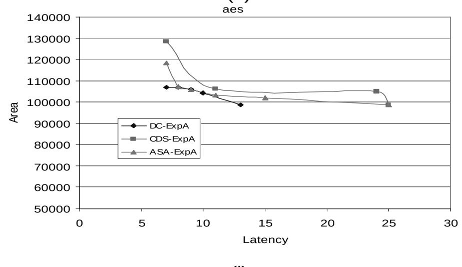
(g)



(h)



(i)



(j)

Fig. 5. Pareto-fronts of benchmarks (a) ave8 (b) gfilter (c) csc (d)FD image shrink (e) FD ISS (f)FCU (g) Reed Solomon Enconder (h) FPU multiplication (i) adpcm (j) aes.

Table VI. Experimental results DC-ExpA using Latency vs. DC-ExpA with cycle accurate simulation

	DC-ExpA(Latency)		DC-ExpA(Sim)		
Bench	#	Run[s]	#	Run[s]	Correlation
Ave8	6	349	5	770	0.97
gfilter	5	2,163	6	4,678	0.97
CSC	7	2,317	16	3,057	0.83
FD_Shrink	5	3,780	16	78,725	0.83
FD_ISS	4	1364	15	7,698	0.33
FCU	10	73,455	8	220,365	0.49
Reed Solomon	6	8,346	9	29,631	0.78
FPU_mult	3	14,190	5	78,088	0.83
adpcm	3	14,190	5	78,088	0.83
aes	5	14,190	5	78,088	0.83
Geomean	5	5,456	9	15,541	
Ratio	1	1	1.8	2.84	
Avg.					0.77

the results indicates the proportionality of these differences. Table IV compares each benchmark individually for a more detailed view of the differences between the three methods.

From the experiments it can be observed that our divide and conquer method is on average (geomean) 70% faster than the annealer method while around 4x (75.5%)slower than the pattern matching method. Column 4 in Table III shows the cardinality (number of non-dominated designs) of the different methods. Our method finds 12% less non-dominated designs compared to ASA-ExpA, but finds on average 1.7x (40%) more than CDS-ExpA. In terms of finding the smallest and fastest design our method finds in almost all cases the smallest and fastest design compared to ASA-ExpA. The smallest design is found in all of all of the cases with the exception of FD_ISSS and FPU_mult, while the designs with smallest latency were found in all cases. This is important, because in most cases these are the designs that are finally chosen by the designer and provide the boundary of the exploration space. Table IV shows a detailed comparison of each test case. E.g. testcase FD_shrink shows that our method finds 20% less non-dominated designs, compared to ASA-ExpA and 40% more than CDS-ExpA. Moreover our newly proposed method executes 725.21% faster than ASA-ExpA, while only 44.29% slower than CDS-ExpA. The last four columns indicate that for this test case our method could find the same smallest and fastest design found by ASA-ExpA, while the smallest design was 42.93% smaller and the fastest design 5.88% faster than the smallest and fastest designs found by CDS-ExpA.

Fig. 5 show the Pareto-front (efficient frontier) of the three methods for all the benchmarks. This gives a quick and intuitive visual summary of the exploration results. It can be observed that the ASA-ExpA and DC-ExpA fronts are almost identical in most of the cases, further proving the validity of our method.

Table V shows the quality assessment of the different explorations methods based on the quality measures described previously, compared to the reference Pareto-front. All indicators show that our method outperforms the fixed pattern assignment method, while leading to very close results to the simulated annealer method, while being around much faster.

Based on these results it is safely to conclude that our method very effectively explores the designs space compared to the two previously developed designs and shows a very good compromise between running time and quality of results. It should also be noted that many different combinations of attributes lead to designs with the very same area and latency characteristics. In this case only one design is reported.

Table VI shows the results of the default DSE method vs. the extended method where a cycle accurate simulation is ran during DSE. The result show that the latency reported by the HLS tool is in some cases a very effective measure of performance while in some other cases quite poor. The correlation column indicates how latency correlates with simulation cycles. The correlation is computed taken the latency reported by the HLS and the simulation cycles reported by the cycle-accurate model simulation for each design generated during the entire exploration. Especially FCU and FD_ISS benchmarks show a poor correlation between total simulation cycles and average latency. These results indicates that in order to perform DSE accurately a full circuit simulation is needed as the latency reported by the HLS is not a reliable measure for performance. The drawback is that the running time increased by a factor of 2.84x. It is also interesting to observe that using a cycle accurate simulation increases the number of Pareto-optimal design by a factor of 1.8x due larger combination of designs with different simulation cycles.

6. CONCLUSIONS

High level synthesis is becoming a must in state of the art hardware designs. Designers can no longer describe and model entire SoCs in low level languages and need to raise the level of abstraction. Tools that bridge the gap between untimed high level languages and RTL are needed. In this paper we present a design space exploration method to speed up the exploration of high level language descriptions given in C and SystemC. The presented method, called DC-ExpA, is based on a divide and conquer method that clusters explorable operations and explores these separately and then combines the attributes in order to obtain Pareto-optimal designs. We compare our method with two previously developed methods based on adaptive simulated annealing (ASA-ExpA) and pattern matching method (CDS-ExpA). Results show that the DSE with out newly proposed method (DC-ExpA)dramatically reduces the runtime compared to ASA-ExpA (61.3% faster) while obtaining comparable results. Compared to CDS-ExpA our method is 68.6% slower, but generates much better results. We believe that our proposed exploration method provides a very good quality vs. runtime tradeoff. In this work we also examined the effect of using the average latency as a measure of performance vs. a cycle accurate simulation concluding that in many cases average latency is not an accurate measure for performance and that a simulation is needed in order to obtain the actual non-dominated designs. Finally, one extra benefit of our method is that due to division of the exploration in independent clusters, runtime could be further decreased by performing each single cluster exploration on different processors/machines. The speed-up that could be achieved is linearly proportional with the number of clusters (considering that the the computation takes much longer than the initial and final communication overhead).

REFERENCES

AHMAD, I., DHODI, M., AND HIELSCHER, F. 1994. Design–space exploration for high–level synthesis. *Computers and Communications*, 491–496.

ASCIA, G., CATANIA, V., NUOVO, A. D., PALESI, M., AND PATTI, D. 2007. Efficient design space exploration for application specific system-on-a-chip. *Journal of Systems Architecture* 53, 733–750.

C2SEE. 2011. http://er.cs.ucla.edu/c2see/.

CADENCE. 2011. http://www.cadence.com/products/sd/silicon_compiler.

CALYPTO. 2011. http://www.calypto.com/catapult_c_synthesis.php.

COUSSY, P. AND MORAWEIC, A. 2008. *High-Level Synthesis from Algorithm Digital Circuit*. Springer.

DAVID, A., VELDHUIZEN, V., AND B.LAMONT, G. 2000. On measuring multiobjective evolutionary algorithm performance. *Congress on Evolutionary Computation 1*, 204–211.

FORTE. 2011. http://www.forteds.com.

GAJSKI, D., ZHU, J., DOEMER, R., GERSTLAUER, A., AND ZHAO, S. 2005. *SpecC: Specification Language and Methodology*. Springer.

GAUT. 2011. http://www-labsticc.univ-ubs.fr/www-gaut/accueil.php.

GIVARGIS, T., VAHID, F., AND HENKEL, J. 2002. *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems 10*.

GUPTA, S., DUTT, N., R.GUPTA, AND NICOLAU, A. 2003. Spark: a high-level synthesis framework for applying parallelizing compiler transformations. *16th International Conference on VLSI*, 461–466.

HARA, Y., TOMIYAMA, H., HONDA, S., AND TAKADA, H. 2009. Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis. *Journal of Information Processing 17*, 242–254.

HAUBELT, C., SCHLICHTER, T., KEINERT, J., AND MEREDITH, M. 2008. In SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models. *DAC*, 580–585.

HAUBELT, C. AND TEICH, J. 2003. In Accelerating Design Space Exploration. *International Conference on ASIC*, 79–84.

HAUBELT, C. AND TEICH, J. 2004. In CHARMED: A Multi-Objective Co–Synthesis Framework for Multi–Mode Embedded Systems. *ASAP*, 28–40.

HOLZER, M., KNERR, B., AND RUPP, M. 2007. In Design Space Exploration with Evolutionary Multi–Objective Optimisation. *Proc. Industrial Embedded Systems*, 125–133.

JHDL. 2005. http://www.jhdl.org.

KALYANMOY, D., AGRAWAL, S., PRATAP, A., AND MEYARIVAN, T. 2000. A fast elitist non–dominated sorting genetic algorithm for multi–objective optimization: Nsga–ii. *Parallel Problem Solving from Nature*, 849–858.

KIM, N., BANERJEE, S., DUTT, N., AND VENKATASUBRAMANIAN, N. 2006. In Design space exploration of real–time multi–media MPSoCs with heterogeneous scheduling policies. *CODES-ISSS*, 16–21.

KNOWLES, J., THIELE, L., AND ZITZLER, E. 2006. A tutorial on the performance assesment of stochastic multiobjective optimizers. *Tech Rep. No. 214, Computer Engineering and Networks Laboratory, ETH Zurich*.

KOBAYASHI, N., WAKABAYASHI, K., TANAKA, H., SHINOHARA, N., AND KANOH, T. 1994. Design experiences with high-level synthesis system cyber i and behavioral description language bdl. *Asia Pacific Conf on Hardware Description Languages*.

KRISHNANA, V. AND KATKOORI, S. 2006. A genetic algorithm for the design space exploration of datapaths during high–level synthesis. *IEEE Transactions on Evolutionary Computation 10,* 3, 213–229.

LEGUP. 2011. http://legup.eecg.utoronto.ca/.

MAMAGKAKIS, S., ATIENZA, D., POUCET, C., CATTHOOR, F., SOUDRIS, D., AND MENDIAS, J. 2006. In Automated exploration of pareto–optimal configurations in parameterized dynamic memory allocation for embedded systems. *DATE*, 874–875.

NEC. 2011. http://www.cyberworkbench.com.

SA-C. 2011. http://www.cs.colostate.edu/cameron.

SCHAFER, B. C., TAKENAKA, T., AND WAKABAYASHI, K. 2009. In Adaptive Simulated Annealer for High Level Synthesis Design Space Exploration. *VLSI–DAT*, 509–519.

SCHAFER, B. C. AND WAKABAYASHI, K. 2009. In Design Space Exploration Acceleration through Operation Clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 29,* 1, 153–157.

SO, B., C.DINIZ, P., AND HALL, M. W. 2003. In Using Estimates from Behavioral Synthesis Tools in Compiler–Directed Design Space Exploration. *DAC*, 514–519.

SO, B., HALL, M., AND DINIZ, P. 2002. A compiler approach to fast hardware design space exploration in fpga-based systems. *IET Computers &Digital Techniques*, 165–176.

SYSTEMC. 2011. http://www.systemc.org.

XPILOT. 2011. http://cadlab.cs.ucla.edu/soc/.

ZITZLER, E., THIELE, L., LAUMANNS, M., FONSECA, C., AND FONSECA, V. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. on Evolutionary Computation 7*.

5

Received February 2010; revised March 2011; accepted November 2011