Multi-layer Configuration Exploration of MPSoCs for Streaming Applications

Deepak Mishra, Yasaman Samei, Nga Dang, Rainer Dömer, Elaheh Bozorgzadeh
University of California, Irvine
{mishrad, ysameisy, ngad, doemer, ebozorgz@uci.edu}

Abstract

While integration of configurable components, such as soft processors, in MPSoC design enables further system adaptation to application needs, supporting system level tools need to provide an environment for systematic and efficient configuration exploration. This paper presents a multi-layer configuration exploration framework for streaming applications on MPSoCs. We introduce a novel Configuration Exploration Tree (CET) for configuration selection per processor. Integrated in a system-level design environment, our CET enables efficient and fully automatic exploration of processor configurations in MPSoC. The proposed CET supports the fast evaluation of feasible configurations by simulation at highest levels of abstraction. In addition, assuming monotonous impact of configuration values on system throughput, we use an ordering among the nodes in the CET to minimize necessary simulations. Our exploration efficiently finds all feasible configurations for a given constraint.

I. INTRODUCTION

Multi-Processor SoC (MPSoC) design paradigm provides the parallelism and flexibility in implementation of high performance embedded system applications, such as streaming multimedia applications. Figure 1 shows an example of such pipelined MPSoCs. Integration of configurable components, such as soft processors, configurable memories, and/or configurable bus interfaces, enables further system adaptation to application needs. For example, soft processors, such as LEON processors, can be configured at micro-architectural level. Similarly, the AMBA bus provides various options for data transfers between system components. System level tools for MPSoC design not only need to be equipped with simulation tools to evaluate the performance of the entire system, they also need to provide an environment for systematic design space exploration for such configurable components. This paper focuses on configuration exploration for streaming applications in MPSoC design.

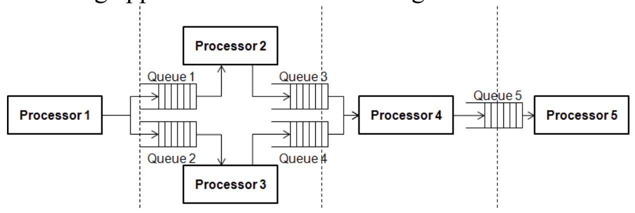


Figure 1- Target MPSoC architecture.

Due to the high amount of configuration options for configurable processors, there is a need for tools at system level to efficiently perform configuration exploration. It is imperative to simulate the configurable components at the presence of other components in MPSoC design. Hence, there is a need for efficient system-level simulation during configuration exploration of MPSoC components. However, configuration selection through simulation (instruction level or cycle accurate) of entire MPSoC is too complex and very time consuming. In order to reduce the complexity, we propose to deploy simulation tools at different levels of abstractions to evaluate the impact of configuration parameters on entire system performance. While simulation results at behavior level are not as accurate as simulation at instruction level, the execution time can be estimated within acceptable error margin depending on the configuration parameters.

In this paper, we propose a *multi-layer* framework for configuration exploration of MPSoCs with configurable components, such as soft processors and configurable bus interfaces. We use three levels of abstraction, namely behavioral, TLM, and ISA. For configuration selection per processor, our approach is based on a novel *Configuration Exploration Tree (CET)* Similar to a branch and bound technique, the nodes in the CET represent the values of configuration parameters. Our CET can capture the multi-layer ordering in configuration exploration by simulating the system at different abstraction layers. In addition, assuming that each configuration parameter has monotonous impact on system throughput, CET exploits this ordering across configuration values to avoid unnecessary simulation runs. We present our novel binary-search-based algorithm at each CET level to find all feasible configurations.

Our exploration tool is integrated in the system-level design environment SCE **[1]**, which allows efficient evaluation of MPSoCs at several abstraction levels. We have integrated a soft-processor model LEON3 with various micro-architectural configuration parameters and configurable AMBA bus into the system design framework. For a given throughput constraint of an application, our automatic configuration exploration finds the feasible configurations for MPSoC platform.

II. RELATED WORK

A wide variety of approaches have been followed to tackle the problem of configuration selection in MPSoCs. Broadly, they can be divided into two categories, System exploration techniques and Micro-architectural exploration techniques. [2], [3], [4], [5] propose micro architectural design tools with ISA level simulation with no evaluation from higher levels of abstractions for exploration. [6] proposes a design methodology for reconfigurable pipelined MpSoCs to execute streaming applications. However, it does not use simulation tools for evaluation of design. [7], [8], [9] present frameworks for system level design space exploration which uses simulation for evaluation but does not provide systematic or hierarchical configuration exploration.

In addition, there are performance estimation techniques for MPSoCs using processor modeling. [10] proposes an analytical method with minimum simulation run to rapidly estimate the runtime of a pipelined MPSoCs subject to estimation error up to 16.45%. [11] presents a symbolic system synthesis approach that can prune the design space in case real-time constraints are violated. In MILAN [12], a design space exploration methodology was proposed to use both symbolic constraint satisfaction to prune the search space and low-level simulator to evaluate remaining candidate designs. However, it does not consider communication interface in MPSoCs. In [13] the parameters of a soft-core microprocessor are adjusted to reach the best performance point for a specific application. As opposed to processor modeling and analytical methods, we focus on multi-layer configuration and simulation to explore the design space of MPSoCs. There are other related works in the area of custom instruction generation [14] and instruction extension [15]. However, none uses an integrated framework of higher level evaluation with ISA simulation.

III. MULTI-LAYER CONFIGURATION EXPLORATION

We describe our systematic approach for exploration of configuration parameters for a pipelined MPSoC target architecture. Our goal is to efficiently explore the complete set of feasible configuration parameters for the processors and system busses so that the system satisfies a given throughput constraint. We will first describe the layering of our framework into three abstraction levels, then discuss the available configuration parameters at appropriate layers, and finally introduce our configuration exploration algorithm.

A. Multi-layer Framework

To avoid exponential complexity, we organize our exploration framework into three major abstraction levels. At each level, a different set of configuration parameters is explored. The highest level, called *behavioral exploration*, considers the different types of processors and their major configuration parameters, such as CPU frequency. The second level, *TLM exploration*, evaluates communication options, e.g. processor bus speed. Finally, in *ISA exploration*, we explore low-level instruction set architecture parameters for each processor.

The reason for this layering is that the evaluation (i.e. simulation) of a design configuration at high level is typically an order of magnitude faster than at the next lower level. Hence, we want to explore the design space first at higher levels of abstraction and eliminate unsuitable configuration options early in design flow.

Figure 2 shows our multi-layer exploration methodology. The design specification and architecture mapping are processed and refined step-by-step by the three exploration phases at the behavioral, TLM, and ISA level. Each phase evaluates a different set of configuration parameters (see parameter segregation below) and relies on corresponding component databases. Note that each exploration phase eliminates infeasible configurations and only passes feasible ones as candidates to the next exploration phase. At the end, we obtain the complete set of configuration parameters that satisfy the given through-put constraint.

B. Configuration Parameter Segragation

The task of *parameter segregation* in Figure 2 is to divide all configuration parameters into three groups, one for each abstraction level. Here, we let the system designer specify the appropriateness and priority of parameters for each phase. Specifically, we evaluate overall processor parameters that are insensitive to specific data at the behavioral level. Examples of such configuration parameters include the processor frequency, multiplier/divider latency (in clock cycles), and support for floating-point arithmetic.

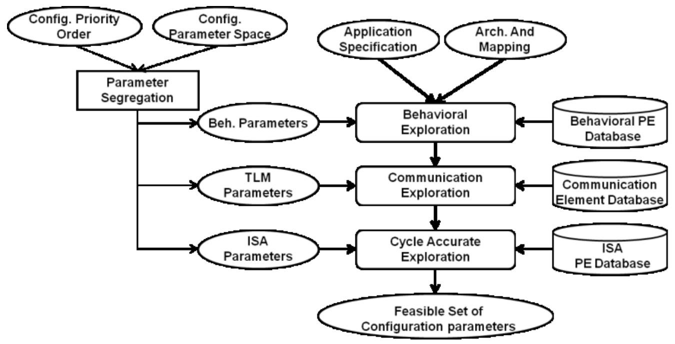


Figure 2- Proposed Multi-layer Configuration Exploration Framework

On the other hand, there are configuration parameters that need to be evaluated at the ISA level, such as register windows size, instruction and data cache size, load delay, and pipeline depth. In between these two groups, we separately explore communication parameters with medium impact. Table 1 shows typical parameters for each abstraction level.

Table 1- Abstraction Levels and Configuration Parameters

Abstraction	Configuration Parameter
Behavioral Level	Frequency, Mul/Div Latency Floating Point Ins. Latency etc.
TLM (Communication) Level	Link Frequency, Burst Size, Transfer Mode etc.
ISA Level	I/D Cache Size/Assoc/Line, Reg. Window, Pipeline Depth, Load Delay, etc.

Note that many high-level parameters have a monotonous impact on performance. That is, they either increase or decrease performance with increasing value. While some configuration parameters such as processor frequency and functional unit cycle count (e.g., multiplier cycle count) are intrinsically monotonic, other configuration parameters such as cache size can be monotonic only within a range of the parameter depending on application behavior (Figure 3). We exploit the monotonicity property of configuration parameters for efficient design space exploration.

In this work, we do not consider any priority or ordering of parameters based on their impact on performance. Configuration parameters are assigned to various layer based on the capability of simulation tool of the corresponding layer to reflect the impact of configuration parameter in overall system performance. Deploying existing methods for ordering the configuration parameters at each layer can further provide efficient exploration [13].

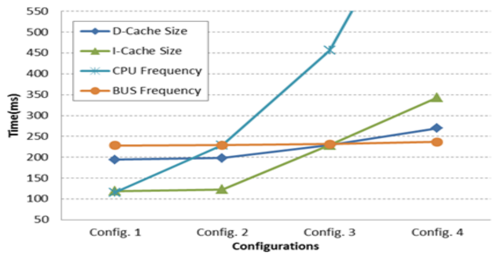


Figure 3- Performance Evaluation for Monotonic Parameters

Table 2- CET Definitions

Definition	**Description**
Level(v)	Level/depth of node *v* in CET which represents the configuration parameter corresponding to *v*
*Conf *(v)*	Best configuration or the configuration of the left most path in the sub-tree beginning at node *v*
Time(v)*	Throughput of the CPU with *Conf*(v)*
Child(v, i)	i_{th} child of node *v*
Parent(v)	Parent of node *v*

C. *Configuration Exploration Tree (CET)*

In order to efficiently explore all possible monotonous configuration parameter values, we represent the feasible configurations by a rooted tree called *Configuration Exploration Tree (CET)*. Each level in the CET corresponds to the exploration of one configuration parameter, and each node represents a single configuration parameter value. Since configurations are explored one parameter at a time, the levels in the tree reflect an ordering based on which the configuration parameters are explored. The level where a configuration parameter is placed in the CET depends on its priority. Parameters closer to the root of the tree are explored earlier and possibly at higher levels of abstraction. Child nodes in the CET represent the feasible values of the next configuration parameter in order of priority. A directed path in the CET from the root to a leaf node represents a feasible combination of these parameter values.

Figure 4 shows the CET layout with the configuration parameters across design abstractions. We first explore the behavioral level configurations and build the CET. For feasible configurations, we then explore TLM and ISA parameters and successively append feasible parameters to the CET. Therefore, we propose a top-down processing of configuration parameters in CET. The configuration parameters considered in our CET are monotonous. Let's assume that the children of each node in CET are arranged from *left to right* based on their impact on system execution time. The arrangement is such that the leftmost child represents the best parameter value for performance and the rightmost child the worst. For example, the nodes representing clock frequency values are ordered from highest to lowest. The relation $\leq$ between any two child nodes v_1 and v_2 refers to such ordering, i.e., v_1 is to the left of v_2. Figure 4 shows a CET layout with the left to right arrangement of children v_1 to v_4 for parameter 1.

Table 2 shows definitions for each node v_i in CET. *Conf*(*v_i*)* refers to the directed path including node v_i which results in the best execution time among all the paths from v_i in CET. *Time*(*v_i*)* refers to the CPU throughput corresponding to this path.

Lemma 1 shows that the ordering between two child nodes in CET imposes a partial ordering between *Time*(*v_1*)* and *Time*(*v_2*)* in throughput.

Lemma 1: *Given any two nodes v_1 and v_2 in CET, if Parent (v_1) = Parent (v_2) and $v_1 \leq v_2$, then Time*(v_1) $\leq$ Time*(v_2)*

Based on Lemma 1, Lemma 2 shows that the leftmost path through node v_i in CET is the best configuration of the system including node v_i.

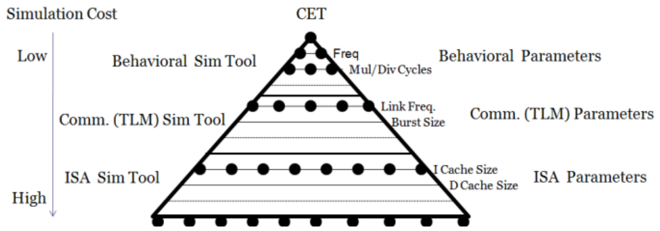


Figure 4- Configuration Exploration Tree (CET)

Lemma 2: *Conf*(v_i) is always the left most path through v_i. If node V is the root of CET, Conf*(V) is the best parameter configuration and Time*(V) is the best throughput.*

As a result, if *Time*(*v_1*)* cannot meet the system timing constraints, none of the paths from both node v_1 and v_2 is feasible. Vice versa, if *Time*(*v_2*)* meets the timing constraint, *Time*(*v_1*)* meets the requirement as well and does not need to be simulated. Similar argument can be made between the child nodes of the nodes v_1 and v_2 as follows:

Lemma3:*Given any two nodes v_1 and v_2 in CET: if Level (v_1) = Level (v_2) and $v_1 \leq v_2$, then*

$$Time^*(Child\ (v_1, i)) \leq Time^*(Child\ (v_2, j))\ \forall\ j \geq i$$

Corollary: $Time^*(Child\ (v_1, i)) \leq Time^*(Child\ (v_2, i))$

Since each path from the root to a leaf in CET represents a single configuration of the system, the problem translates to finding all the feasible paths in the CET for which the execution time of the system is less than the timing constraint T. By our proposed top-down ordering of the configuration parameters according to their abstraction layer as well as ordering of the nodes in CET based on monotonous configuration parameters, finding feasible paths in CET can be performed efficiently.

IV. CET CONSTRUCTION

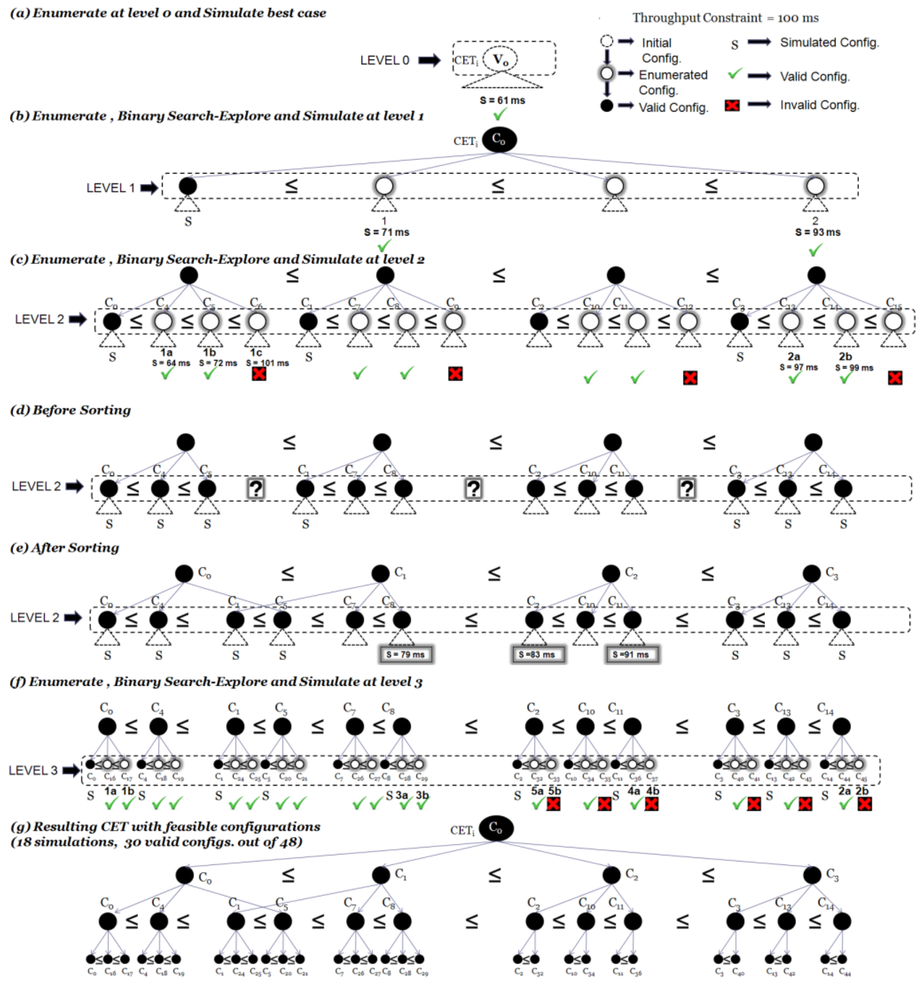
During our configuration exploration, we construct a *feasible CET* for each processor. Initially, the configuration parameters at the behavioral level are considered and a feasible CET is constructed by pruning those paths in the tree which do not satisfy the throughput constraint. After the behavioral level CET construction, we extend the existing paths in the CET by considering the TLM parameters and again prune those to keep only the feasible paths. Finally, we explore the ISA level parameters the same way, leading to a complete CET of all feasible configurations.

A. *CET Construction Algorithm*

In order to prune the CET at each level, we follow a breadth-first-search (BFS) approach. Our BFS algorithm begins at the root node and explores all children in the CET. Then, for each child node, it explores its unexplored children, and so on. The flow of our CET construction algorithm is shown in Figure 5. We start with the root configuration at level 0 of the CET (v_0 in Figure 5(a)) which represents the best configuration of the design as per *Lemma 2*. If this configuration satisfies the throughput constraint, we add it as the leftmost path to the CET (v_1 in Figure 5(b)). Next, we enumerate the other children of the root (nodes v_1, v_2, v_3, v_4) and evaluate their values to identify which satisfy the throughput constraint. Instead of performing this exploration exhaustively by looking at each child (which is naïve and would lead to exponential complexity), we exploit *Lemma 1* and explore these configurations efficiently using *binary-search-explore* (BSE) approach. In the example in Figure 5(b), we first pick configuration value v_2, build, and simulate the corresponding design. In this example, the throughput of 71ms meets the timing constraint of 100ms. Hence, this configuration value leads to a feasible design and is noted as such in the CET. In general, the leftmost unexplored node is evaluated first, then the rightmost node, and then the middle node in case the rightmost one fails and so on. This approach is repeated in binary-search manner until the set of feasible values is determined. In Figure 5(b), we next evaluate v_4 which is a valid configuration (throughput 93ms) and added to the CET. Note that in this case we do not need to simulate node v_3 because its throughput must be valid between the values for v_2 and v_4 as per *Lemma 1*.

Moving on to the next level, we use the CET property described in *Lemma 3* and pick the leftmost enumerated child of each node v (Child(v, 0)) and perform BSE to find out which are valid. Then, we pick Child(v, 1) of each node v. In Figure 5(c), the order of evaluation is shown with numbers 1a, 2a… corresponding to Child (v, 0) $\forall$ v, and 1b, 2b… corresponding to Child(v, 1) $\forall$ v, and so on. After all valid nodes at level 2 are determined, we enumerate their children and go to level 3. Note that, in order to continue with our BSE at level 3, we need all nodes at level 2 sorted. *Lemma 3* shows, however, that this is not necessarily the case in a general CET. Thus, in order to use our BSE approach correctly at level 3 and further down, we need to sort the nodes at their upper level based on their throughput. In general, we need to keep sorting as we continue our exploration one level after another in the CET. Figure 5(d-e) show level 2 before and after sorting correspondingly (note the reversed order of nodes v_7 and v_9).

Our goal in sorting the CET nodes is to minimize the number of additional simulations. We use a binary search approach where we take the children of given two nodes and merge the sorted lists. We take the ordered list of children and find the position of the other children one by one using binary


(a) Enumerate at level 0 and Simulate best case
Throughput Constraint = 100 ms
Initial Config.
Enumerated Config.
Valid Config.
Simulated Config.
Valid Config.
Invalid Config.
LEVEL 0
CET_i
V_0
S = 61 ms
(b) Enumerate, Binary Search-Explore and Simulate at level 1
LEVEL 1
S = 71 ms
S = 93 ms
(c) Enumerate, Binary Search-Explore and Simulate at level 2
LEVEL 2
S = 64 ms
S = 72 ms
S = 101 ms
S = 97 ms
S = 99 ms
(d) Before Sorting
(e) After Sorting
S = 79 ms
S = 83 ms
S = 91 ms
(f) Enumerate, Binary Search-Explore and Simulate at level 3
LEVEL 3
(g) Resulting CET with feasible configurations
(18 simulations, 30 valid configs. out of 48)
Figure 5- Example of CET Construction Algorithm

search. As shown in Figure 5(e), first the children of the two leftmost nodes are merged; the children of 3rd node from the left are merged with the previously merged list, and so on. In the example, after sorting at level 2, we continue the exploration at level 3 (Figure 5(f)) in the same BSE manner, which required only 18 simulations. At the end, we compute the CET of the design; Figure 5(g) shows the final CET of this example with all 30 feasible configurations out of the 48 possible ones.

Table 3- Nomenclature used in CET Algorithm.

Term	Description
T	Throughput Constraint
Spec	Application Specification/Task Graph
Arch	Pipelined MpSoC Architecture
Map	Application Task Graph to Architecture Mapping
DB	Database of Processors/Communication Elements
P_i	Processor i in the MpSoC pipeline
C	$\{C_1.....C_n\}$ where C_i are configuration parameter sets of the processors *i*
C_{iK}	$\{CP_{iK1}, CP_{iK2}.......CP_{iKc(i)}\}$ configuration parameter set of processor *i* at design mode/abstraction *K*
C_i	Configuration parameter set of processor $i = \{C_{iB,} C_{iC}, C_{iI}\}$ for K = B (Behavior), C (TLM), I (ISA)
CP_{iKj}	Configuration parameter *j* of processor *i* at mode $K=\{v_1, v_2 ...v_{n(j)}\}$ where v_1, v_2 etc. are the sorted values of *j*
CET_list	Forest of $CET_i s$ where CET_i is the feasible CET of processor *i*
SNL_i	Current list of sorted nodes in CET_i, a separate list that maintains the sorted nodes at a CET level

B. CET Construction Algorithm Pseudo Code

We will now present the CET construction algorithm in detailed pseudo code for the Breadth-First-Search (BFS), Binary-Search-Explore (BSE), and the sorting of children nodes. Table 3 defines the nomenclature used in the algorithm.

The algorithm *CET Construction* explores the three abstraction levels top down (line 7) and builds the corresponding CET (line 9) for each processor. At the end, it returns a forest of CETs (line 13). Each exploration consists of parameter segregation (line 10) and BFS exploration (line 11). The BFS exploration, as outlined in the example above, first enumerates and simulates the best configuration, i.e., the leftmost path (lines 15-16), then enumerates and sorts the child configurations (lines 21-22), and finally performs the recursive BSE algorithm (lines 27-34).

Algorithm: CET Construction

```
Algorithm Input: Spec, Arch, Map, C, T
Algorithm Output: CET_list
1 Initialize: CET_list:= {};
2       C: = {$C_1$, $C_2$.....$C_i$};
3 Design = Spec;
4 Set Mode = {REFINE_ARCH, REFINE_COMM, REFINE_ISA};
5 Repeat for each processor $P_i$
6       $CET_i$ = nil; $SNL_i$ = nil;
7 For each Mode $M_k$, k є 0...2
8   Design = Refine (Design, $M_k$, DB, Arch, Map);
9   Repeat for each processor $P_i$
10      $C_{ik}$ = Segregate ($C_i$, $M_k$);
11      $CET_i$ = BreadthFirstSearchExplore ($CET_i$, $C_{ik}$, Design, T);
12      If ($CET_i$ == nil) Reject Design;

13 CET_list = {$CET_1$, $CET_2$……$CET_n$};
14 function BreadthFirstSearchExplore ($CET_i$, $C_i$, Design, T) {
15  conf = EnumerateBestConfig ($CET_i$, $C_i$); // Lemma 2
16  exec_time = Simulate (conf, Design);
17  If (exec_time < T)
18    AddPath ($CET_i$, conf);
19    If ($SNL_i$ == nil) $SNL_i$ = $CET_i$;
20    for conf_param in $C_i$
21      $SNL_i$ = SortChildren ($SNL_i$);  // Lemma 3
22      conf_list = EnumerateChildrenConfig ($CET_i$, $SNL_i$, $C_i$,
                conf_param);
23      BinarySearchAndExplore ($CET_i$, conf_list);
24    Return $CET_i$;
25  Else Return nil;}
26 function BinarySearchAndExplore ($CET_i$, List conf_list) {
27  R = sizeof (conf_list);
28  For param_value j = 0 to n (j)
29    new_R = BinarySearch ({conf_list [0][j]),…
                    conf_list[R][j])});   // Lemma 3 Corollary
30    For i: = 0 to new_R
31      AddPath ($CET_i$, conf_list[i][j]));
32    R = new_R;}          // shrink range of search
33 List SortChildren (List SNL) {
34  List new_SNL;
35  For child in Children (SNL [0])
36    new_SNL→Append (child);
37  For i = 1 to sizeof (SNL)-1
38    new_SNL = MergeUsingBinarySearch (new_SNL,
            Children (SNL[i]));
39  Return new_SNL;}
```

C. Analysis

We will now analyze the computational complexity of the CET construction where the number of simulations performed is the dominating factor. First, if none of the configuration parameter values are feasible, the very first simulation will fail, i.e. the complexity is O(1). In the general case, the complexity of the CET construction algorithm is the sum of the complexity of BSE and the complexity of sorting the children. Suppose there are k configuration parameters in the design and each configuration parameter has 1 to ni values for i=1…k, then the number of simulations required during BSE is log(n1) + (n2-1)*log(n1) + (n3-1)*log(n1*n2) + … + (nk-1)*log(n1*n2*…nk-1). Similarly, the number of simulations required for the sorting of children is, in the worst case, of the order O(n1*n2*….nk-1). This is large, but in the average case the child nodes in the CET will have only minimal overlap due to the priority ordering of parameters. Moreover, any sorting at higher levels reduces the cost at lower levels significantly.

In the worst case, the number of simulations in the CET construction algorithm is $O(\sum_{i=1}^{k} n_i * \log(\prod_{i=1}^{k-1} n_i)) + O(\prod_{i=1}^{k-1} n_i)$. The simulations are distributed across abstraction levels, thereby reducing the cost. Since we further simulate all the processors in the pipeline in parallel, our approach is much better than brute force which is in the order of $O(numprocs * \prod_{i=1}^{k} n_i)$.

V. EXPERIMENTS

We have implemented our configuration exploration in the SoC Environment [1]. We integrated the configurable Leon3 processor core for our case study. Leon3 is a SPARC v8 processor with a 7-stage pipeline and configurable I/D cache, register window, floating point unit, etc. In SCE, we integrated back-annotated behavioral and Transaction Level Model (TLM) simulators, and cycle-accurate Instruction Set Simulator (ISS) in the database. Table 4 shows the configuration parameters used in our experiments.

Table 4- Configuration Parameters

Abstraction	Configuration Parameter
Behavioral Level	CPU Frequency = 25, 50, 100, 200 MHz Mul/Div Cycles = 10, 15, 20 CPU Cycles
TLM (Communication) Level	Link Frequency = 25, 50, 100, 200 MHz
ISA Level	I-Cache Size = 16, 32 ,64 I-Cache Assoc = 2,4 D-Cache Size = 16, 32 ,64 D-Cache Assoc = 2,4

The target pipelined reconfigurable MPSoC consists of Leon3 soft processors connected by buffers connecting the processors through reconfigurable point-to-point links based on the AMBA bus (similar to Figure 1). For our exploration, we selected 3 synthetic benchmarks which are essentially streaming applications with filters in a pipeline. Each stage consists of array-based filter operations or DSP-type operations like MAC (multiply accumulate), row replacement, etc. Benchmark B1 has a single 4-stage pipeline (each stage mapped to a Leon3 processor), benchmark B2 is a multi-pipeline data-parallel version of B1 (4 stages, 3rd stage with two data-parallel units, mapped to 5 Leon3 processors), and benchmark B3 is a pipelined matrix-multiplication parallel benchmark (4 stage pipeline, 3rd stage two parallel units, mapped to 5 Leon3 processors). We ran our proposed configuration exploration algorithm over these benchmarks. The results for the first processor in the pipeline of benchmark B1 are shown in Table 5. As expected, simulation at the behavioral level is much faster than the TLM, and even much faster than the ISS simulation. Using our framework, we are able to prune 144 configuration simulations for the processor with only 9 behavioral simulations. The exploration results for all the three benchmarks are shown in Table 6. The table shows the feasible CPU configurations and simulation runtime. Compared to a brute force approach which simulates the entire potential configuration set, we save significant run-time improvement by exploring configuration space in layers of abstractions. An average ISA level simulation takes around 107.8 seconds. If all the configurations are simulated for one CPU of any benchmark, it will take 51.7 hrs of simulation time whereas we explore the entire design space in about 1-1.5 hour for all the benchmarks. Comparison of both these numbers establishes the effectiveness of our approach.

VI. CONCLUSION

In this paper, we presented a multi-layer configuration selection for MPSoC architectures with configurable processors and bus interface. Our proposed framework explores the configuration parameters at three levels of abstractions: behavioral, TLM, and ISA. We propose a configuration exploration tree to explore all the configurations per processor. By exploiting the monotonous impact of configuration parameters on system throughput, we propose a binary-search based algorithm on CET to avoid redundant simulations in searching for feasible configurations. Our preliminary results show efficient pruning of the design.

Table 5(Results for B1- CPU1)

Design Layer	Potential Configs	Successful (simulated)	Failed (simulated)	No. Pruned Solutions in CET	Simulated Time
Beh	12	11(8)	1(1)	1*4*36=144	< 0.1 sec
TLM	11*4= 44	44(8)	0(0)	0	~ 6.99 min
ISA	44*36 = 1584	506(19)	1078(6)	1078	~44.91 min

Table 6- Experimental Result

Bench-Marks	Potenti-al Configs	Feasible Solutions (simulated time in min)					Total Simulati-on Time (in min)
B1	1728*4	CPU1 506 (51.9)	CPU2 425 (54.58)	CPU3 436 (38.27)	CPU4 482 (21.33)		166.08
B2	1728*5	CPU1 506 (53.14)	CPU2 434 (62.65)	CPU3 544 (38.21)	CPU4 521 (37.8)	CPU5 488 (39.1)	230.9
B3	1728*5	CPU1 384 (63.46)	CPU2 456 (65.33)	CPU3 514 (91.11)	CPU4 434 (63.81)	CPU5 476 (80.33)	364.04

BIBLIOGRAPHY

[1] R. *Dömer*, et al., "System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design," *EURASIP Journal on Embedded Systems, Article ID 647953*, 2008 .

[2] B. C. Lee and D. Brooks, "Efficiency Trends and Limits from Comprehensive Micro architectural Adaptivity," in *ASPLOS* , 2008.

[3] C. Haubelt, T. Schlichter, and J. Teich, "Improving Automatic Design Space Exploration by Integrating Symbolic Techniques into Multi-Objective Evolutionary Algorithms," *(IJCIR), Special Issue on MultiobjectiveOptimization and Applications*, vol. 2, no. 3, pp. 239-254, 2006.

[4] C. Dubach, T. M. Jones, and M. F. P. O'Boyle, "Micro architectural Design Space Exploration Using An Architecture-Centric Approach," in *MICRO* , 2007.

[5] H. Cook and K. Skadron, "Predictive Design Space Exploration Using Genetically Programmed Response Surfaces," in *DAC*, 2008.

[6] H. Javaid and S. Parameswaran, "A Design Flow for Application Specific Heterogeneous Pipelined Multiprocessor Systems," in *DAC* , 2009.

[7] F. N. v. Wijk, J. P. M. Voeten, and A. J. W. M. T. Berg, "An Abstract Modeling Approach Towards System-Level Design-Space Exploration," in *In Proc. Of the Forum on Specification and Design Language*, 2002.

[8] F. Angiolini, et al., "An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration," in *DATE*, 2006.

[9] C. Silvano, et al., "MULTICUBE: Multi-Objective Design Space Exploration of Multi-Core Architectures," in *IEEE Annual Symposium on VLSI*, 2010.

[10] H. Javaid, A. Janapsatya, M. S. Haque, and S. Parameswaran, "Rapid Runtime Estimation Methods for Pipelined MPSoCs," in *DATE*, 2010.

[11] F. Reimann, M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich, "Symbolic System Synthesis in the Presence of Stringent Real-Time Constraint," in *DAC*, 2011.

[12] S. Mohanty, V. K. Prasanna, S. Neema, and J.Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation," in *LCTES/SCOPES*, 2002.

[13] D. Sheldon, F. Vahid, and S. Lonardi, "Soft-core processor customization using the design of experiments paradigm," in *DATE*, 2007.

[14] L. Bauer, M. Shafique, and J. Henkel, "Cross-Architectural Design Space Exploration Tool for Reconfigurable Processors," in *DATE* , 2009.

[15] U. D. Bordoloi, H. P. Huynh, T. Mitra, and S. Chakraborty, "Design space exploration of instruction set customizable MPSoCs for multimedia applications," in *International Conference on Embedded Computer Systems (SAMOS)*, 2010 , pp. 170-177.