THE UNIVERSITY OF TEXAS AT DALLAS

Department of Electrical and Computer Engineering

Design Automation and Reconfigurable Computing

Improving the Quality of Hardware Accelerators through automatic Behavioral Input Language Conversion in HLS

Md Imtiaz Rashid and Benjamin Carrion Schafer

{MdImtiaz.Rashid,schaferb}@utallas.edu

27th Asia and South Pacific Design Automation Conference

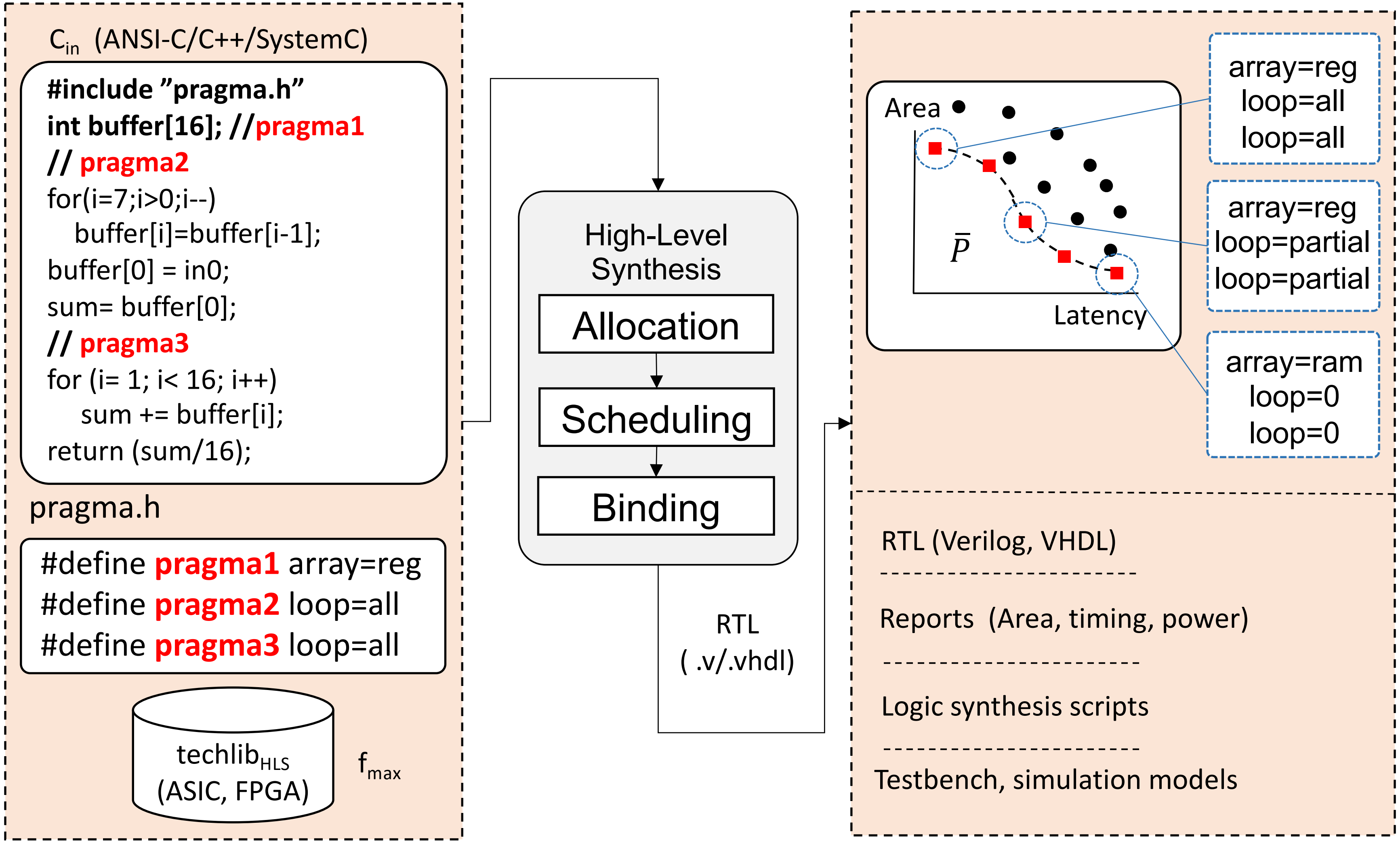
ASP-DAC 2022

High-Level Synthesis Overview

Input
- Behavioral description
- Directive setting
- Technology library, fmax

High-Level Synthesis
- Allocation
- Scheduling
- Binding

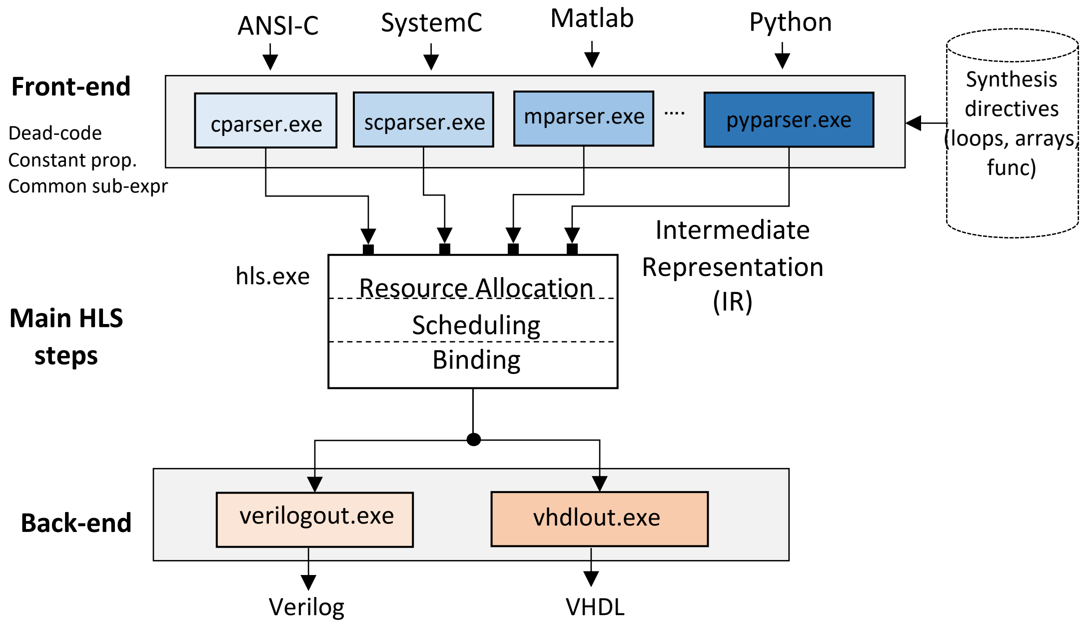
Output
- RTL Design
- QoR reports
- Script for Logic Synthesis
- Simulation Models



(a) HLS Inputs **(b) HLS** **(c) HLS Outputs**

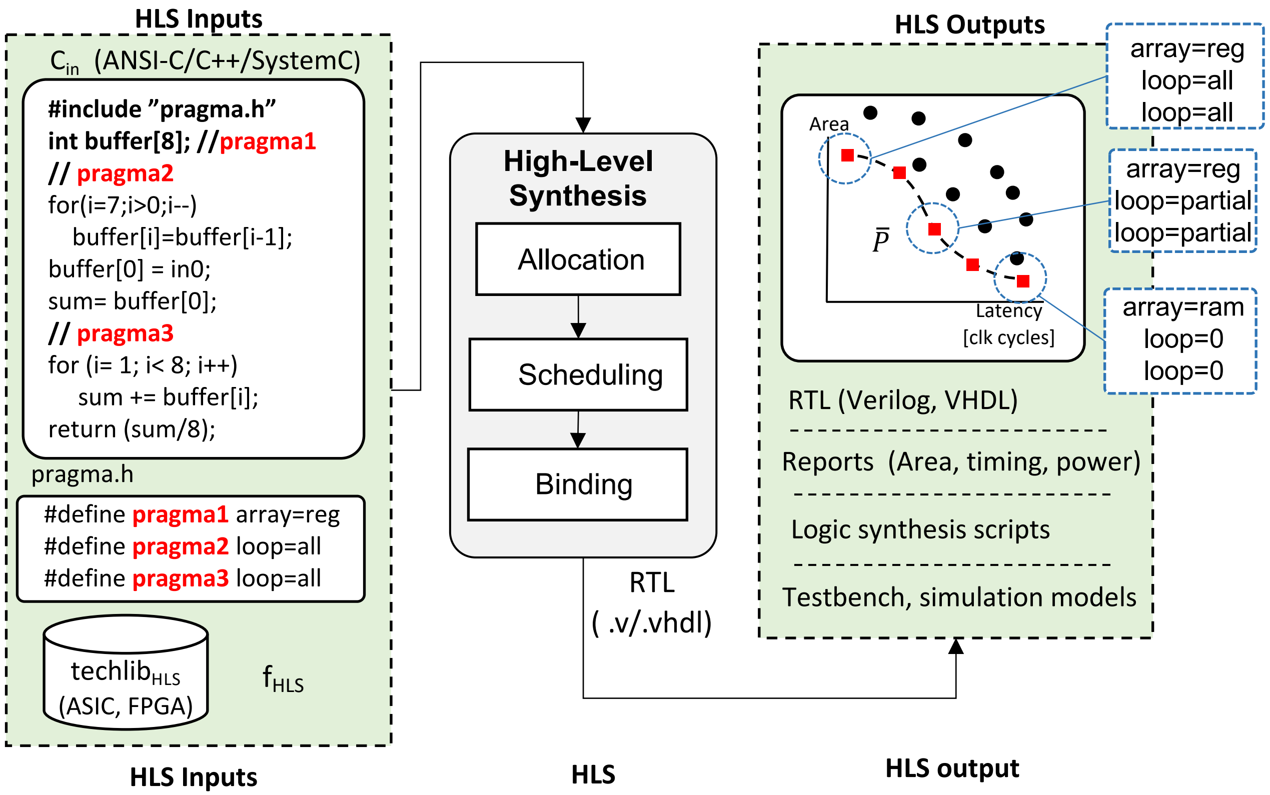
Introduction

- High-Level Synthesis tools support multiple input languages. E.g., ANSI-C, C++, SystemC, Matlab
- HLS tools are built in a modular way
- Language dependent parsers for each supported language
    - Syntax checks
    - Technology independent optimizations
    - Parsers output the optimized CDFG in a common tool format → allows to re-use the rest of the HLS flow



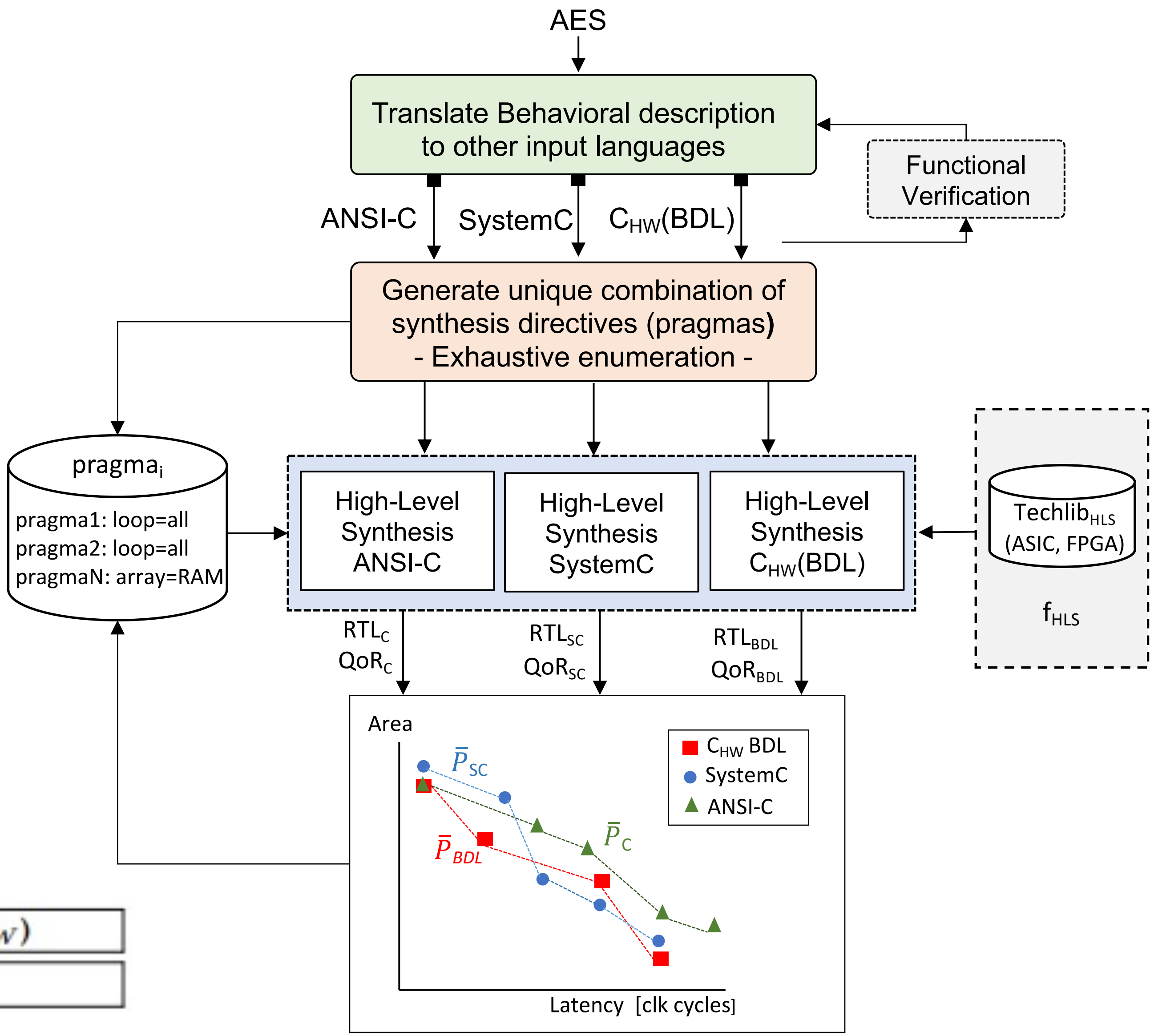
Functional Equivalent Design Generation

- HLS allows to decouple functional description from implementation through synthesis directives/pragmas
- These control how to synthesize:
  - Arrays : RAM, Reg
  - Loops: Unroll, pipeline
  - Functions: inline or not



Motivational Example

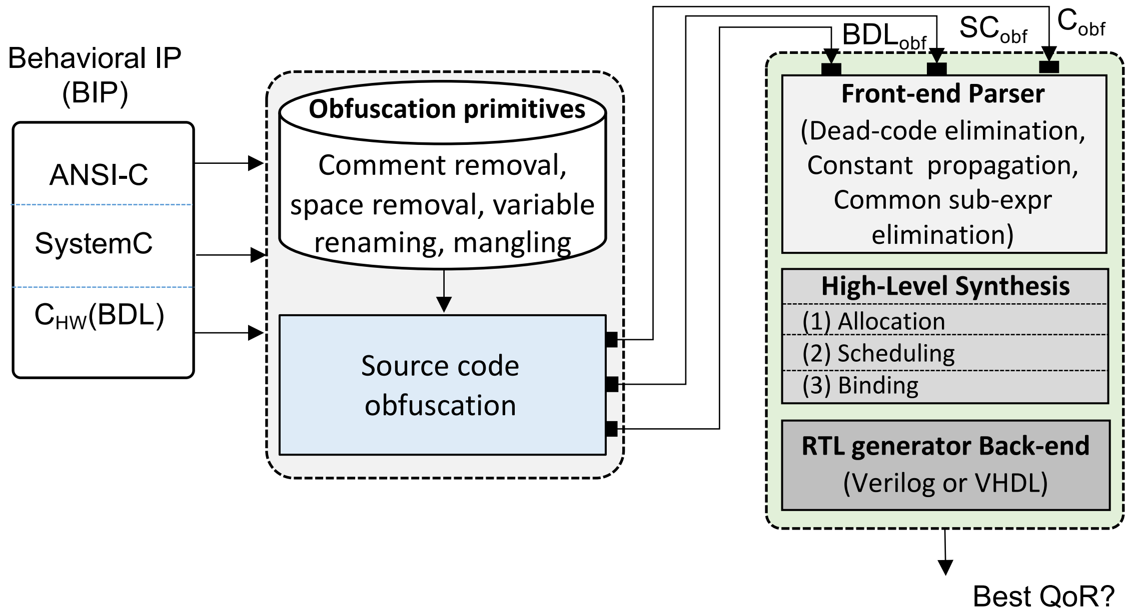
- AES block cipher written in ANSI-C, SystemC, C_{HW} (BDL) (all using same data bitwidths)
- Set different synthesis directive combinations for each AES description and compare the Pareto-optimal designs found
- ADRS: Average Distances to Reference Set: The lower the better → Different input languages lead to different trade-off curves



	ANSI-C	SystemC	BDL (C_{HW})
ADRS	5.68	7.29	3.19

Source Code Obfuscation

- Easy and inexpensive way to hide the source code
- Functional equivalent source file is generated, which is virtually impossible for humans to understand and extremely difficult to reverse engineer
- The obfuscation process typically:
  1. removes comments
  2. Renames variables
  3. adds redundant expressions → Parser dependent



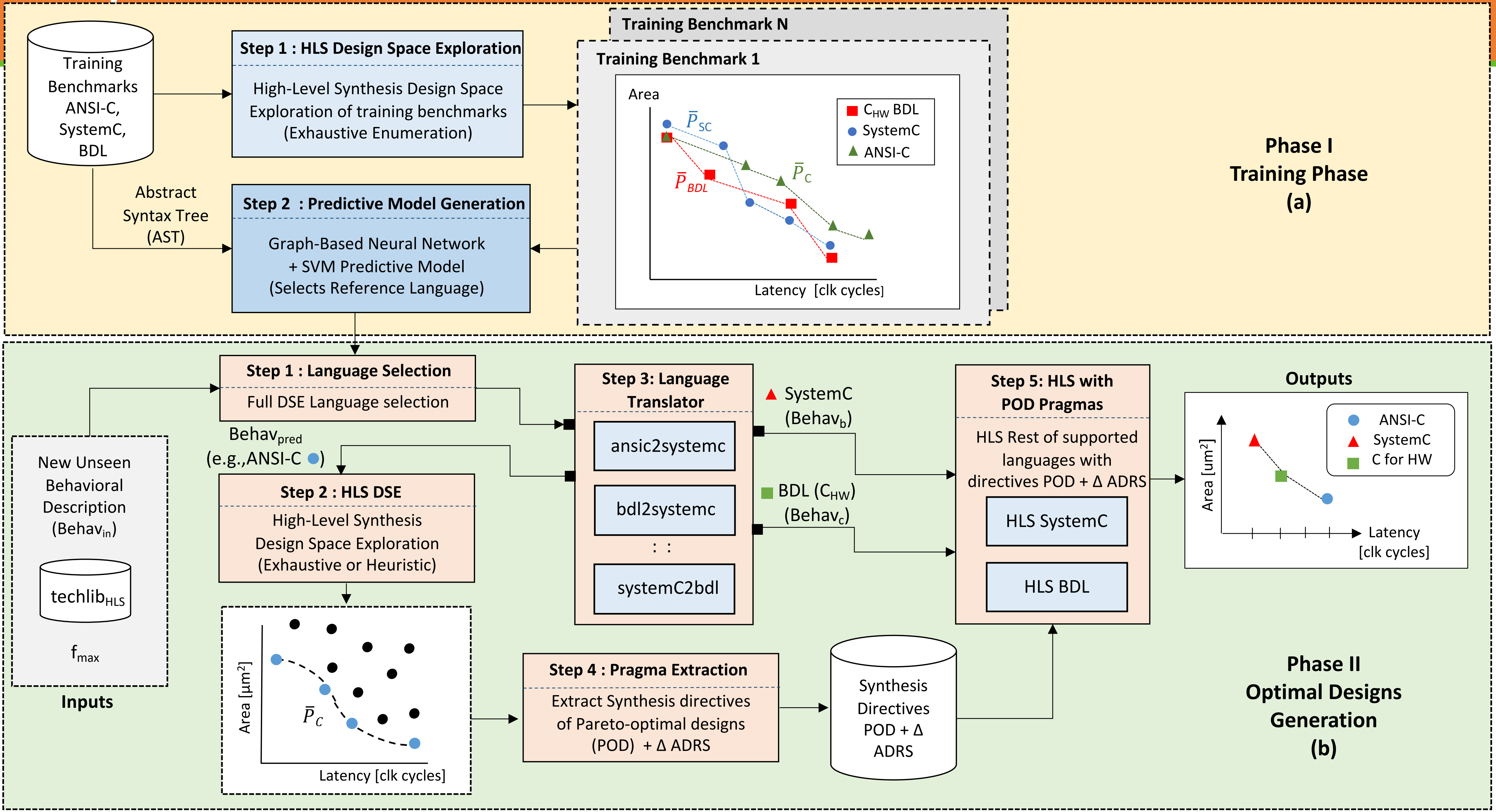
Un-Obfuscated Results

	ANSI-C	SystemC	BDL (C_{HW})
ADRS	5.68	7.29	3.19

Obfuscated Results

	ANSI-C (C_{obf})	SystemC (SC_{obf})	BDL (C_{HW}) (BDL_{obf})
$ADRS_{obf}$	12.56	7.29	9.45

Proposed Flow

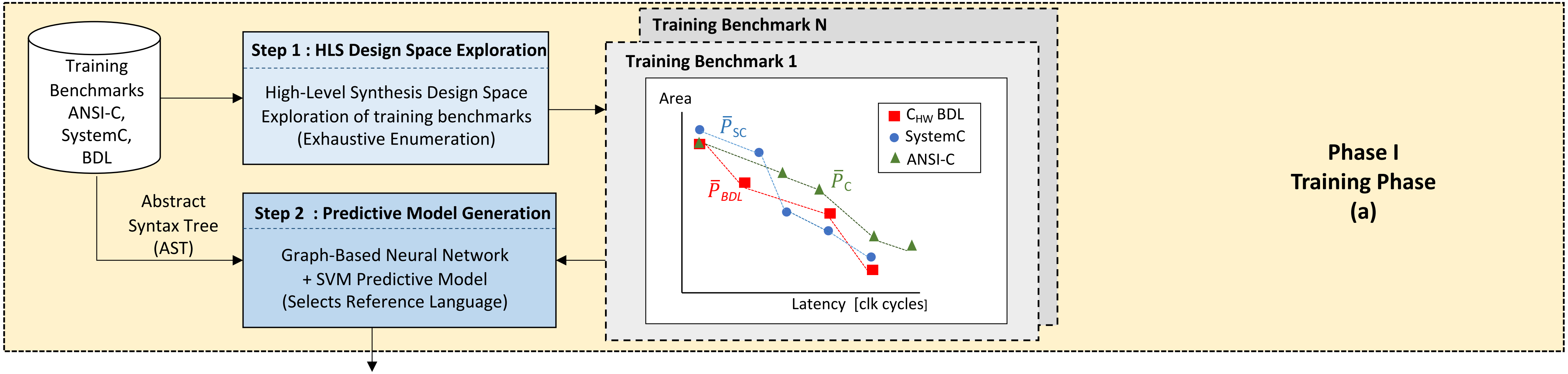


Phase 1: Training Phase

- Composed of 2 steps:

  Step 1: Exhaustive enumeration of training benchmarks for all input languages supported by target HLS tool

  Step 2: Generate predictive model based on Graph Neuron Network +SVM to select best input language based on the program structure

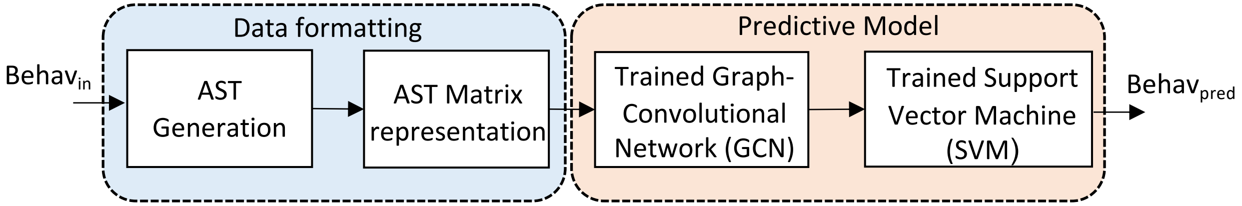


Predictive Model

- **Input:**
  - A behavioral description to be synthesized HLS
- **Output:**
  - The converted behavioral description in the language supported by the HLS tools that will lead to the best QoR
- Composed of 2 steps:

  Step 1: Data Formatting :
  1. AST Generation
  2. AST Matrix representation

  Step 2:
  1. Trained Graph Convolutional Neural Network to extract features
  2. Trained Support Vector Machine (SVM)



Phase 2: Optimal Design Generation

- Inputs:
  1. New Unseen behavioral description in one of the supported languages
  2. Techlib, fmax

Outputs:

1. Pareto-optimal designs

- Composed of 5 steps:

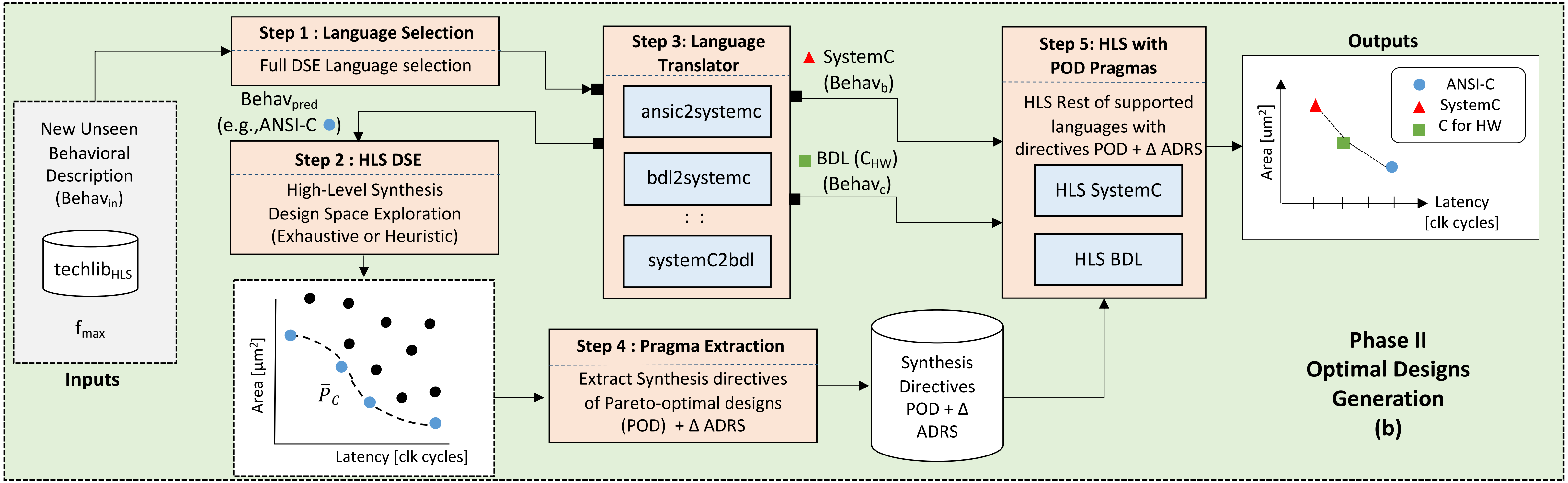
  Step 1: Input language selection

  Step 2: HLS Design Space exploration (exhaustive for small designs or heuristic)

  Step 3: Automatic input language converter

  Step 4: Pragma extraction for Pareto-optimal designs

  Step 5: Insert pragmas of Pareto-optimal designs in predicted best input language



Experimental Setup

Tools

- High-Level Synthesis tools: NEC CyberWorkBench v.6.1.1
    - ANSI-C, SystemC, C_{HW} (BDL)
- HLS technology : Nangate Open-source 45nm
- HLS synthesis frequency: 100Mhz
- Benchmarks: CHStone and S2CBench
- Source code obfuscator: Stunnix

Platform

- Computer platform
    - Intel(R) Xeon E7 with 16GBytest of RAM
    - CentOS Linux release 7.8.2003 (Core)

Evaluation

- Compare our proposed approach with exhaustive exploration of all benchmarks for each input languages
- Relax the candidate constraint by considering design candidates within 1.5% ADRS from the Pareto-optimal ones for each input language

Experimental Result: Un-Obfuscated Benchmarks

Un-Obfuscated Benchmarks

Bench	Input Language	Explore all		Predicted Language	Proposed (POD only)		Proposed (POD+1.5% ADRS)		Proposed (POD+3% ADRS)	
		ADRS	Run [min]		ADRS	Run [min]	ADRS	Run [min]	ADRS	Run [min]
adpcm	ANSI-C	0.00	15.68	BDL	0.00	9.62	0.00	10.36	0.00	11.62
ave8	SystemC	0.00	5.38	BDL	0.00	1.49	0.00	1.59	0.00	2.25
cholesky	ANSI-C	0.00	26.73	SystemC	1.25	5.13	0.00	7.13	0.00	9.70
fir	SystemC	0.00	39.65	ANSI-C	0.00	19.03	0.00	20.63	0.00	22.03
sobel	ANSI-C	0.00	60.60	BDL	0.00	26.91	0.00	27.08	0.00	29.91
interp	SystemC	0.00	68.62	BDL	2.54	13.17	1.25	15.60	0.00	17.19
kasumi	SystemC	0.00	126.82	SystemC	4.36	17.32	0 .00	18.31	0.00	21.46
aes	SystemC	0.00	524.90	BDL	6.55	135.78	2.55	138.45	0.00	140.41
Geomean			**47.14**			**13.59**		**15.02**		**17.46**
Avg.		**0.00**			**1.84**		**0.48**		**0.00**	

- **Observations:**
  - Our proposed method is effective finding the input languages that will lead to the overall best results
  - Relaxing the pool of candidates leads to better results although it slightly increases the running time, on average by 1.1x and 1.3 for the 1.5% ADRS and 3% ADRS cases respectively.

Experimental Result: Obfuscated Benchmarks

Obfuscated Benchmarks

adpcm	ANSI-C	0.25	17.32	SystemC	2.32	10.08	1.85	11.52	0.25	13.54
ave8	SystemC	1.10	5.30	SystemC	1.45	1.52	1.45	2.08	1.10	2.56
cholesky	ANSI-C	0.50	32.10	SystemC	1.25	5.09	0.50	6.23	0.50	8.23
fir	SystemC	3.69	38.44	SystemC	5.26	19.23	4.51	21.85	3.69	24.51
sobel	ANSI-C	1.55	69.32	SystemC	1.63	27.58	1.67	29.45	1.55	32.32
interp	SystemC	3.40	68.21	SystemC	5.65	13.01	3.89	15.12	3.40	18.45
kasumi	SystemC	4.36	125.01	SystemC	4.36	18.96	4.36	20.11	4.36	23.54
aes	SystemC	4.60	526.33	SystemC	8.57	137.22	7.50	142.54	6.11	149.31
Geomean			**49.36**			**13.91**		**15.94**		**18.65**
Avg.		**2.62**			**3.27**		**3.22**		**2.62**	

- **Observations:**
  - The SystemC parser does a very good job optimizing away redundant expressions introduced by the obfuscator

Conclusions

- We have shown that the quality of the synthesis result strongly depends on the input language parser
- This is even more pronounced in the case of source code obfuscation
- We have presented an automatic input language translator for typical input languages used in HLS and a GCN+SVM approach to determine which input language is more likely to lead to better results
- Experimental results show the effectiveness of our proposed approach

Thank You