UPaK: Abstract Unified Pattern Based Synthesis Kernel for Hardware and Software Systems

Christophe Wolinski
wolinski@irisa.fr
Rennes University I / IRISA
France
http://www.irisa.fr

Krzysztof Kuchcinski
krzysztof.kuchcinski@cs.lth.se
Lund University
Sweden
http://www.cs.lth.se

Adam Postula
adam@itee.uq.edu.au
University of Queensland
Australia
http://www.elec.uq.edu.au

Abstract

The $UPaK$ system is designed to be a kernel of an automatic hardware synthesis as well as code and configuration generation for architectures including ASIP processors and reconfigurable systems.

1. Introduction

The developed prototype of $UPaK$ system is designed to be a kernel of an automatic hardware synthesis as well as code and configuration generator for several architecture models. These models include processors with extended data-paths corresponding to extended instruction sets, processors with coarse grain reconfigurable systems tightly connected to data-paths, heterogeneous run-time coarse grain reconfigurable systems, and existing coarse grain reconfigurable systems.

UPaK is implemented using advanced software technologies that include graph matching and flexible scheduling techniques recently developed by the authors [4, 5]. It is written in the Java language and therefore is a multi-platform tool.

2. UPaK system

The system provides a systematic method for identification of frequent computational patterns or other patterns of interest. This method is based on graph isomorphism constraint and constraints programming; that makes it very flexible and provides the basis to integrate in one formal environment the graph isomorphism constraints, other design constraints, and the heuristic search for patterns. It takes into account the graph structure and frequency of occurrence of patterns in the application graphs, it also includes finding maximal coverage of the application graph with the patterns of interest.

The input to the UPaK system (figure 1) is the abstract Hierarchical Conditional Dependency Graph (HCDG), the architecture model, the technological parameters and the synthesis constraints. The HCDG defines behavior of the application and is generated from the Polychrony environment [2]. It was successfully used in high level synthesis and reconfigurable system synthesis before [3]. The synthesis constraints can include such parameters as execution time, area, and power consumption (currently implemented version supports only time constraints).

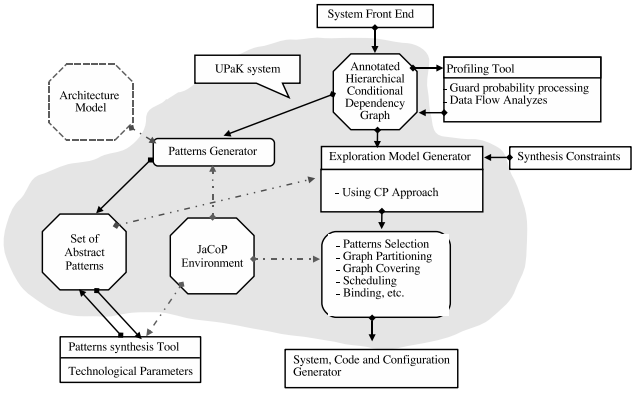

Figure 1. UPaK design flow.

The system visualizes the automatically generated patterns, the input HCDG graph covered by the selected patterns, and the abstract application execution scenario on the selected architecture model (figures 2, 3 and 4).

The UPaK hierarchical pattern-based design methodology has a number of features that make it superior in the context of HW/SW compilation for heterogeneous (reconfigurable) systems and create an opportunity to develop competitive commercial synthesis tools. These features are

following:

- computation patterns represent instructions in software, while in hardware they define components; this makes the basis for a unified representation for hardware/software compilation,
- computation patterns represent different configurations that can be mapped onto the same reconfigurable architecture,
- computation patterns are sub-graphs in HCDG and graph matching can be used for their identification as well as for mapping the HCDG graph onto hardware and/or software implementation,
- patterns in our approach form a hierarchy that helps to handle complexity of designs.

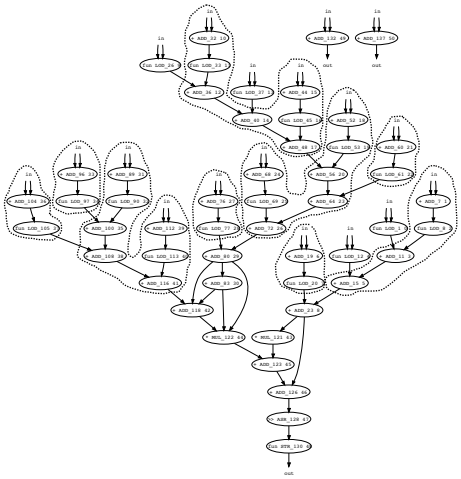

Figure 2. Covered HCDG.

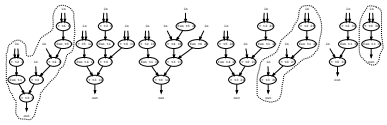

Figure 3. Patterns

Table 2 shows the quality of patterns generated by the $UPak$ system obtained for the DSP applications from the MediaBench test suite [1]. It includes a number of patterns identified by the $UPaK$ system for the original graph (step 1) as well as for graphs derived from this graph after removing all found matches of patterns (step 2). It also presents a number of patterns that are used for maximum coverage of the graph as well as the graph coverage for step 1 and the total coverage. The table shows that high coverage for application graphs with a small number of patterns in a short amount of time could be obtained. All experiments have been run on 2GHz Intel Core Duo under Mac OS X operating system.

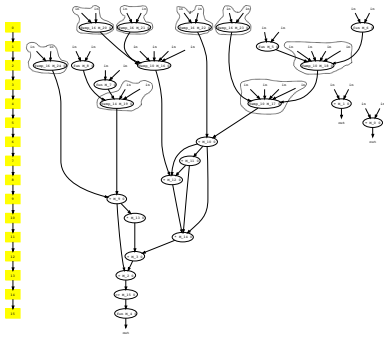

Figure 4. Abstract execution scenario.

Application	$\|V\|$	time	step 1			step 2		total
		(s)	id.	sel.	cov.	id.	sel.	cov.
JPEG Write BMP Header	106	4.99	6	4	90%	3	2	96%
JPEG Smooth Downsample	51	3.36	8	4	74%	3	1	88%
JPEG IDCT	134	20.90	7	5	69%	3	2	81%
MPEG IDCT	114	5.30	3	2	54%	6	3	71%
MPEG Motion Vector	32	0.9	8	3	93%	1	1	100%
EPIC Collapse	56	2.09	7	4	69%	3	3	85%
MESA Smooth Triangle	197	120.00	7	3	73%	2	1	87%
MESA Horner Bezier	18	0.36	9	3	83%	1	1	94%
MESA Interpolate Aux	108	22.8	3	2	85%	3	1	100%
MESA Matrix Multiplication	109	28.4	7	4	56%	4	2	86%
MESA Feedback Points	53	1.70	4	2	80%	3	3	94%
FIR	44	12.5	7	4	72%	2	2	90%
Elliptic Wave Filter	34	2.20	9	4	67%	2	2	94%
Auto Regression Filter	28	3.30	4	3	96%	-	-	-
Cosine	66	7.05	8	3	50%	7	3	74%

Table 1. Results for MediaBench test.

3. Conclusions

The UpaK system implements radically new approach to generation of computation patterns that is based on subgraph isomorphism constraints and constraint programming. The method can be used for identification of application specific instructions as well as hardware components. The main feature of our method is that it can identify both the most frequently occurring patterns as well as patterns of the largest size.

References

[1] Media benchmarks. http://express.ece.ucsb.edu/benchmark/.

[2] Polychrony. http://www.irisa.fr/espresso/Polychrony/.

[3] A. Kountouris and C. Wolinski. Efficient scheduling of conditional behaviors for high level synthesis. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, page 7(3):380412, July 2002.

[4] K. Kuchcinski. Constraints-driven scheduling and resource assignment. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, pages 8(3):355–383, July 2003.

[5] C. Wolinski and K. Kuchcinski. Computation patterns identification for instruction set extensions implemented as reconfigurable hardware. In *submitted for publication*, 2007.