Reconfigurable design automation by high-level exploration

Tim Todman, Wayne Luk
Department of Computing
Imperial College London
London, UK
{timothy.todman, w.luk}@imperial.ac.uk

***Abstract*—This paper describes a novel approach for design automation of general-purpose reconfigurable computing applications, which combines design space exploration with transformation-based high-level feedback of performance results obtained from a detailed implementation. This approach enhances effectiveness of high-level exploration by using performance estimates to guide the selection of applicable transformations. The impact of the transformations on metrics such as speed and area is pre-characterised, so that appropriate transformations likely to contribute to meeting design requirements are selected. A prototype system supporting this approach has been developed, and promising results have been obtained.**

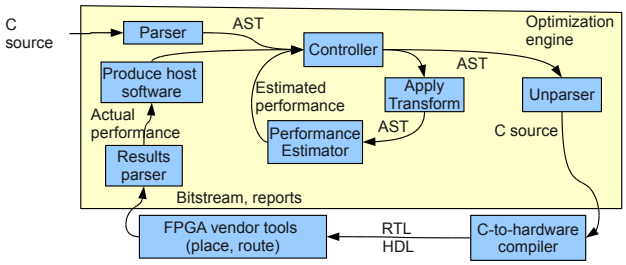

Fig. 1. Overview of our prototype, showing two nested loops: inner loop rapidly explores design space using performance estimator; outer loop uses external tools to get real performance measurements.

I. Introduction

A major challenge for reconfigurable computing is to improve designer productivity while, ideally, raising the quality of the implementations at the same time [1], [10]. This challenge is becoming increasingly acute, since FPGA (Field-programmable Gate Array) vendors continuously produce larger and more complex devices which not only require greater effort from designers to use them efficiently, but also more time for vendor tools to implement designs on them. Both algorithmic techniques [3] and architectural techniques [5] have been reported in addressing this situation.

A well-known method for improving productivity is to increase the abstraction of the design representation [1]. The intention is to allow designers to focus on key properties of the design, ignoring implementation details which can be vendor specific. The benefits are obvious: designs are simpler so design cycles can be shorter; implementation details can be taken into account separately to improve design re-use; there are automatic ways of optimizing designs by transformation; and it is easier to specify functional requirements for high-level descriptions, facilitating development and verification.

One way to automate high-level design exploration and optimization is to apply successive correctness-preserving transformations, turning an obvious description into a less obvious, more efficient one.

A transformation-based approach is attractive, because: (a) the relationship between designs is made explicit; (b) transformations allow parameterizing and formalizing optimizations, facilitating their reuse, automatic application and verification; (c) it enables designers to work at a higher level, selecting applicable transformations and sequences of transformations.

There is, however, one key issue. High-level descriptions are, by their nature, abstract and may not contain sufficient information to enable accurate performance estimation. This inaccuracy in performance estimation could affect the selection of appropriate transformations to produce a design that meets the desired performance requirements.

This paper introduces a novel approach, which combines high-level design exploration and optimization with feedback of accurate performance information, while making use of estimation techniques to avoid spending excessive amounts of time on running FPGA tools.

The contributions of this paper include:

- a transformation-based approach to high-level design exploration and optimization that uses both estimated and accurate implementation information to guide the development process;
- a prototype tool that supports this approach, representing designs as C programs, optimized by a transformation engine;
- an evaluation of the proposed approach with an analysis of the results.

Whilst our prototype tool deals with designs represented as C programs, the approach is not limited to C and could adapt to other high-level descriptions such as behavioural VHDL.

The rest of the paper is organized as follows. Section II reviews related work. Section III provides a high-level overview of our approach to high-level exploration; Section IV describes a prototype implementation for FPGA design optimization. Section V evaluates our prototype system. Finally, section VI concludes and suggests ideas for future work.

II. Background

Related work on improving productivity for reconfigurable computing can be divided into two classes: those at device level and those at application level.

At device level, since placement is one of the most time-consuming stages, there is much research on methods to accelerate FPGA placement. Techniques based on maximum-bipartite matching techniques have been explored [3]. Incremental methods [13] have also been proposed to reduce execution time by only placing the parts that are changed. Another possibility is to parallelize the placement algorithm to target multiple processors [8]. A complementary approach is to examine how FPGA architectures themselves can be developed in a way to reduce execution time of design tools; an investigation into high-capacity logic block architectures has been reported [5]. Such techniques are orthogonal to our work. Unless the implementation time approaches the high-level transformation time, the benefits of our approach remain.

At application level, recent work suggests that high-level synthesis tools, such as those based on the C language, can achieve quality of results similar to manual design [2]. However, designs still need to be written in a way to allow compilers and synthesis tools to carry out effective optimization. To address this issue, techniques and tools have been proposed to automate the transformation of designs at the source level [15]; a variety of transformations can be supported, including syntax-directed transformations and goal-directed transformations. However, current transformation tools are driven by performance estimates which may not be accurate and, as a result, inappropriate transformations can be selected which would lead to suboptimal designs. The proposed approach addresses the accuracy issues with performance estimates. Representing loop transformations using polyhedral models allows some transformations to be described and combined mathematically [14]; this could combine with our approach; we could also work with heuristics for design-space exploration [4].

The need for our approach is reduced if accurate performance estimates are available. However, current research indicates that such estimates are only available for resource usage [12] and power consumption [9] for RTL (Register Transfer Level) descriptions. While there has also been work on performance estimation for FPGA-based processors [16], we are not aware of a general technique capable of rapid performance estimation with high accuracy for reconfigurable computing designs.

There is also much research in software compiler optimization to find an appropriate order of optimizing transformations; examples use virtual execution [6] and machine learning [7]. However, it is non-trivial to adapt such techniques for general-purpose reconfigurable computing, since: (a) there is no fixed instruction-set architecture, and (b) hardware compilation time is significantly longer than software compilation time.

III. Overview of approach

We optimize designs by repeatedly applying *high-level transformations* to them; here *high-level* means that (a) the design is represented in an abstract way that could correspond to many concrete implementations; (b) the transformations similarly operate at high-level, rather than on low-level, device-specific descriptions. Examples of high-level representations include software source code, behavioural hardware descriptions, or dataflow graphs for streaming applications.

Our approach contains the following elements:

- a high-level representation of the design to be optimized;
- a transformation engine operating on the high-level representation;
- a performance model, to rapidly evaluate performance of the high-level design representation;
- a means of making a detailed implementation, from which real performance results can be obtained;
- a means of updating the performance model to reflect the real performance results.

Operating at a high level has important advantages. For example, pattern matching for applying transformations is easier as we do not need to factor out or recognise low-level details. We can also evaluate designs much more rapidly, because there are no low-level details. Design transformations take seconds rather than minutes or hours to apply. Conversely, high-level representations and transformations have disadvantages: they are abstract, so evaluating design performance and the effects of transformations is less accurate. Accurate evaluation requires translating to an implementable representation to measure the real performance. This can be slow; for FPGAs, implementing large designs can take hours, even days.

We combine the benefits of high-level representations and transformations with detailed design performance data obtained by periodically using detailed implementations. If the detailed implementation meets the user requirements, the optimization terminates. If it does not, we update the performance model to match the real results to the model; some transformations may be reversed if the performance worsens.

Our approach makes a tradeoff: more frequent detailed implementations allow more accurate high-level evaluations, because the model is more frequently updated and compared to implementation results, but make the overall process slower.

The proposed approach provides a simple but powerful means of customizing design automation of reconfigurable computing applications: first, while our approach is general-purpose, it also works well for high-level application-specific and domain-specific descriptions.

Second, if there are multiple ways to estimate performance with trade-offs in estimation speed and estimation accuracy, designers can choose the best way depending on their particular need; some would involve a complete implementation, while others could involve resource estimation tools based on RTL descriptions. Similarly, vendor tools offering fast implementation allow more frequent detailed implementation. Early outer-loop iterations can set the vendor tools to use lower effort for faster, more frequent implementation; later iterations can use higher effort as the process homes in on the final design.

Finally, designers can focus on organizing design transformations. Sequences of successful transformations for related designs can be captured and adapted to for re-use. When

designs are ported to different devices, adjusting device-specific transformations may suffice for an acceptable solution.

IV. Framework prototype

We develop a prototype framework implementing our proposed approach. The components implemented are:

- The high-level representation is a program source code represented as an Abstract Syntax Tree (AST);
- The high-level transformation engine transforms ASTs into other ASTs, using the ROSE compiler framework [11];
- The performance model uses a syntax-directed model of a C-to-hardware compiler which, given a C program, estimates area and speed by linear functions of the estimated speed and area;
- a strategy decides when to interleave real implementations using the compiler and FPGA vendor tools with high-level transformations;
- detailed implementation results are obtained by unparsing the AST back to C, compiling to hardware, using FPGA vendor tools to compile to a bitstream and real performance measurements, which are then mapped back to update the linear performance model.

The *input* to our prototype is a C design plus user goals for speed and area, with the set of transformations; the *output* is the optimized C design plus the bit streams from vendor tools.

Figure 1 shows our prototype, with the major components and the nested-loop: the *inner* loop rapidly generates new designs by applying source-to-source transformations, evaluating them using a performance estimator. The *outer* loop uses vendor place and route tools to implement designs on the target hardware.

Our prototype takes a design in the C language, which a *parser* translates into an Abstract Syntax Tree (AST). The optimization process is supervised by a *controller*, which runs the two-loop process to try to meet user design goals. In the inner loop, the controller uses a *transformation engine* to generate many designs from the current design, by choosing from the available set of source-to-source (strictly AST to AST) transformations and applying them.

To evaluate the performance of generated designs without placing and route them, we use a *performance estimator* to evaluate design performance, such as speed and area. The performance estimator can use a straightforward approach, for example a syntax directed depth-first traversal of the AST, summing all register and operator widths, and recording the maximum logic depth. To make a performance estimator for a C to hardware compiler, users write test programs to measure the cycle time of different operator widths and control structures. More sophisticated models encompassing, for example, routing delays can be used, but they would take longer to finish.

Users choose a strategy to decide when to get real implementation results. Example strategies could include: running a fixed number of inner loops per outer loop, or ending the inner loop when the estimated performance improvement crosses a threshold. The controller chooses the most promising design from the inner loop to implement on the target hardware. We use ROSE's *unparser* to translate the AST back to C. Next, a *C to hardware compiler* translates the C code into a suitable form for hardware implementation, such as a RTL Hardware Description Language (HDL) description; suitable compilers include Xilinx's AutoPilot [2]. The RTL can be synthesized by *FPGA vendor tools* into a bitstream, plus reports detailing size and area usage of the design.

Finally, the outer loop uses a *results parser* to read speed and area results from the place and route reports. We use this as the actual performance, though our approach could measure wall-clock execution times, accounting for the data transfer time, and other effects not captured by the reports.

Given estimates of $speed'$ and $area'$, we approximate the real performance as a linear combination of the estimates:

$$speed = k_{s0} + k_{ss} \times speed' + k_{as} \times area'$$
$$area = k_{a0} + k_{sa} \times speed' + k_{aa} \times area'$$

where k_{s0}, k_{ss}, k_{as}, k_{a0}, k_{sa} and k_{aa} are *estimation parameters* which model interdependencies between speed and area.

We assume the real performance is a linear function of the estimates for simplicity, although the actual relation could be nonlinear for tightly packed designs. Note that the assumed performance for speed uses both estimates for speed and area; the rationale is that the performance of large designs will depend not just on the estimated speed (the logic depth), but also on the area, due to more routing congestion. Larger area can alleviate routing, or it can support additional processing, leading to higher speed. Similarly, the assumed area depends on the estimated speed as well as the estimated area. These estimates could extend to other design parameters such as power and energy consumption.

Initial values of the estimation parameters can come from previous designs, set by domain or platform experts or otherwise set to reasonable initial values.

To reconcile the actual performance results with the estimated speed and area, the controller tries to correlate the real results with the estimates, using this feedback to generate better estimates for subsequent estimates. This maps the real results back to the estimates, to calculate better values of the estimation parameters, based on linear regression.

This describes one outer loop iteration. Subsequent outer loops revisit the designs generated by the first outer loop with better performance estimates, homing in on the final design. When the design meets user goals, or when a user-specified time limit is reached, the controller terminates, giving either a design meeting the optimization goals, or feedback to the user: designs tried, the number of iterations, and so on. The user can modify their goals or the inputs to subsequent runs of the same process.

Our approach is also *modular*: our transformations could use other compiler frameworks. We implement our own performance estimator, but could use any performance estimator with C input. Much of our approach is agnostic to the C to hardware compiler, if it matches the performance estimator. Finally, other FPGA vendor tools can be used, if performance results can be extracted.

Metric	S	U_{64}	I	IP_2	IP_4	IP_8	P_4U_{64}
P	1	1	1	2	4	8	4
U	1	64	1	1	1	1	64
LUTs/10^3	6.65	33.1	6.05	7.36	10.2	14.9	118
FFs/10^3	7.52	42.7	7.24	9.37	13.7	22.2	155
BRAMs	11	12	14	22	37	61	27
Speed	77.5	74.1	75.1	70.4	68.2	63.9	71.3
PU	1	64	1	2	4	8	256
Bus txns	1	1	W	W	W	W	1
data/txn	NW	N	N	N	N	N	N
Area	1	4.98	0.909	1.11	1.53	2.24	17.8
Speedup	1	61.2	0.977	3.16	6.13	11.5	235
Speedup/Area	1	12.3	1.07	2.85	4.01	5.13	13.2

TABLE I
MEDIAN FILTER RESULTS; SPEED IN MHZ; I = INNERONLY, U_x = UNROLL BY x, S = STRIPMINE, P_x = PARALLELIZE BY x, TXN = TRANSACTION.

V. RESULTS

Experimental setup: we use Maxeler's MaxJava system to implement designs. MaxJava is a stream compiler, so loop-carried dependencies must be removed before compiling to hardware by the transformations: *loop unrolling* (unrolling the carrying loop completely); *loop tiling* (stripmining and interchange to move dependencies outside the streamed loop) and *Inneronly* (implementing only the innermost loop body; dependencies are carried in software).

Unrolling costs extra hardware, stripmining and innermost cost extra time, due to the need to replicate the inputs. We apply other transformations: loop coalescing, loop interchange, loop unrolling and loop parallelization, represented by C pragmas.

To reduce the size of the design space, we apply the following heuristic: apply parallelization after other transformations, keeping most place-and-route times low; we unroll before parallelization to increase overall parallelization. The strategy is to run two inner loops per outer loop.

We systematically translate from C to MaxJava designs by (a) moving all computation to the innermost loop body using predication and (b) generating counters to implement loop iteration variables. The Maxeler system parses the output of the Xilinx FPGA place and route tools, allowing us to extract the performance information.

Table I shows results for part of the design space, showing speedup compared to the initial stripmined version (speedup for hardware parts only), with the Maxeler (2011.3) and Xilinx (ISE v13.1) tools aiming at a clock speed of 75MHz with high effort, aiming at a Xilinx Virtex 6. Total parallelism PU is the unroll factor U multiplied by the parallelization factor P.

We compare both speed and area of the designs, as well as metrics of speedup, area factor and finally speedup per area, to measure the area cost-effectiveness of our transformations, using the stripmined version as a basis for comparison. We also show the number of bus transactions used and the data per transaction, measuring the use of the bus. Stripmining and unrolling use the bus better than inneronly: (one bus transaction versus W); unrolling at the expense of area.

Stripmining and Inneronly give roughly the same clockspeed, despite stripmining having a more complicated control path. We can see that for median filtering, unrolling and parallelism both give large speedups, but that unrolling is much more cost-effective in terms of area than parallelism. Combining unrolling with speedup gives the fastest design, but it is barely more cost-effective than unrolling alone. Compared to the stripmined version, the largest design is about 235 times faster, and 13.2 times more cost-effective.

VI. CONCLUSION

We present a framework for optimizing a design via high-level design exploration, obtaining accurate performance information by detailed implementation. The proposed approach combines fast optimization at high-level with accurate feedback of performance information to guide the transformation process. Results show that our approach has led to about 235-times speedup compared to the original design.

Current and future work includes: (a) parallel optimization, for example by generating multiple designs as in our approach, but then placing and routing multiple designs in parallel, choosing one or more as the basis for the next generation; (b) more nested loops in the approach (fig 1), running parts of the place and route tools in their own loops.

Acknowledgements: This work was supported by UK EPSRC, the European Union Seventh Framework Programme (grant agreement numbers 248976, 257906 and 287804), Maxeler, and by Xilinx.

REFERENCES

[1] B.E. Nelson et. al. Design productivity for configurable computing. In *Proc. ERSA*, pages 57–66, 2008.
[2] Berkeley Design Technology. An independent evaluation of the AutoESL autopilot high-level synthesis tool. 2010.
[3] H. Bian, A. C. Ling, A. Choong, and J. Zhu. Towards scalable placement for FPGAs. In *Proc. ACM Int. Symp. on FPGAs*, pages 147–156, 2010.
[4] C. Silvano et. al. Multicube: Multi-objective design space exploration of multi-core architectures. In *2010 IEEE Computer Society Annual Symposium on VLSI*, pages 488 –493, July 2010.
[5] S. Chin and S. Wilton. Towards scalable FPGA CAD through architecture. In *Proc. ACM Int. Symp. on FPGAs*, pages 143–152, 2011.
[6] K. Cooper et al. ACME: Adaptive compilation made efficient. In *Proc. LCTES*, pages 69–77, 2005.
[7] Grigori Fursin et. al. Milepost GCC: Machine learning enabled self-tuning compiler. *Int. Jnl. of Parallel Programming*, 39:296–327, 2011.
[8] A. Ludwin, V. Betz, and K. Padalia. High-quality, deterministic parallel placement for FPGAs on commodity hardware. In *Proc. ACM Int. Symp. on FPGAs*, pages 14–23, 2008.
[9] P. Schumacher et. al. Fast RTL power estimation for FPGA designs. In *Proc. FPL*, 2011.
[10] S. Merchant et al. Strategic challenges for application development productivity in reconfigurable computing. In *Proc. National Aerospace and Electronics Conference (NAECON)*, 2008.
[11] M. Schordan and D. Quinlan. A source-to-source architecture for user-defined optimizations. In *Proc. Modular Programming Languages, LNCS 2789*, pages 214–223. Springer, 2003.
[12] P. Schumacher and P. Jha. Fast and accurate resource estimation of RTL-based designs targeting FPGAs. In *Proc. FPL*, pages 59–64, 2008.
[13] D. Singh and S. Brown. Incremental placement for layout driven optimizations on FPGAs. In *Proc. Int. Conf. on CAD*, pages 752–759, 2002.
[14] Steven Derrien et al. High-level synthesis of loops using the polyhedral model. In *High-level synthesis*, pages 215–230. Springer, 2008.
[15] T. Todman, Q. Liu, W. Luk, and G. Constantinides. Customizable composition and parameterization of hardware design transformations. In *Proc. DSD*, pages 595–602, 2010.
[16] L. Yan, S. Lam, and T. Srikanthan. Performance estimation framework for FPGA-based processors. In *Proc. Int. Conf. on Field-Programmable Technology*, 2010.