UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Facoltà di Ingegneria

EARLY PREDICTION OF HARDWARE COMPLEXITY IN HLL-TO-HDL TRANSLATION

Alessandro Cilardo, Paolo Durante, Carmelo Lofiego, and Antonino Mazzeo

Dipartimento di Informatica e Sistemistica – via Claudio, 21

Università di Napoli Federico II, Italy

Email: `acilardo@unina.it`

1 September 2010

20th International Conference on Field Programmable Logic and Applications

RC and HW/SW partitioning

- FPGAs and *reconfigurable computing* are increasingly central in digital design flows
- they provide an opportunity for highly tailored hardware/software partitioning
- A non-trivial application development process:
  - both software and hardware design skills are necessary
  - identification of performance bottlenecks and exploration of a wide design space are difficult and may be very costly

HLL-to-HDL translation

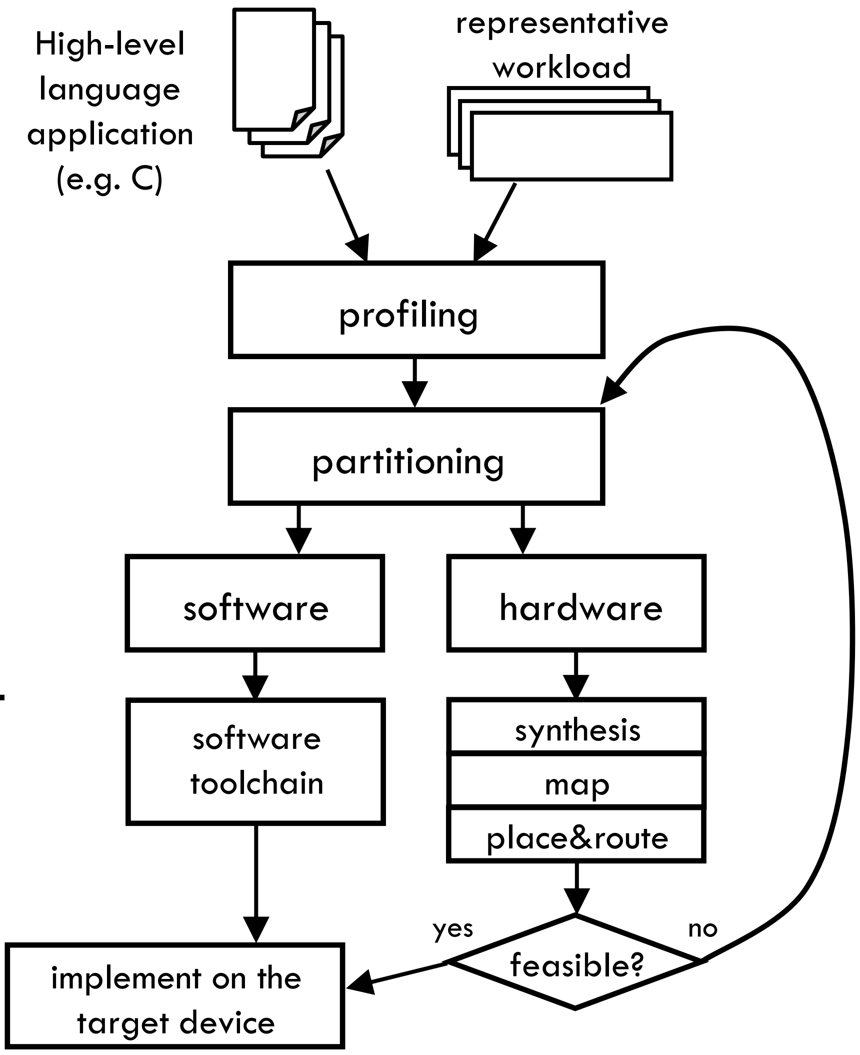
- It is desirable to make FPGA design accessible to developers having pure software skills
- Raise the level of abstraction:
  - adopt a high-level language (HLL) for design entry
  - hide the details of the underlying technology to the application developers
- Automatic translation from HLL to some hardware description language (HDL)

HLL-to-HDL translation

- Today, there is a very large variety of environments for HLL-to-HDL translation
  - both academic: DWARV, ROCCC, SPARK, GAUT, …
  - and commercial: Handle-C, CatapultC, ImpulseC, …
- They all map the program data-flow graph (DFG) to a number of customizable processing units
- All rely on well-established techniques used by software compilers
  - apply straightforward code optimizations
  - guide iteration reordering transformations
  - identify opportunities for parallelization
  - optimize mapping of data to external memories
  - …

HLL-to-HDL translation

- Automatic HLL-to-HDL is an essential tool to explore HW/SW partitioning choices
- Partitioning has an *iterative nature*
  - because of the **physical constraints** imposed by the particular target device
  - based on a *trial-and-error* approach
- Exact results for the HW part of the design are available only **after** actual synthesis and place&route



Early Estimation of HW resources

- Early estimation of hardware requirements:
  - only look at the high-level application code
  - provide the fastest feedback possible to designer/compiler
  - this multiplies the number of partitioning choices that can be explored, possibly in an automated way
- The selection of suitable early prediction metrics is ***inherently*** influenced by the underlying toolchain
  - metrics should be studied and carefully selected for each given toolchain

Existing approaches

- Problem never studied systematically. Most contributions focus on specific HLL-to-HDL tools
  - **[K02]**: consumption formulae for all nodes of the DFG derived from C code by the SA-C HLL-to-HDL compiler. Tightly integrated with the SA-C environment
  - **[S03]**: entirely relies on some tools available in early commercial synthesis environments. Accuracy is relatively low. Used for the DEFACTO HLL-to-HDL framework
  - **[M07]** investigates the adoption of software complexity metrics. Uses metrics borrowed from the software engineering domain. Doesn't capture low-level aspects of HLL-to-HDL translation

[K02] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi, "Fast area estimation to support compiler optimizations in FPGA-based reconfig urable systems", procs. of the 10th Annual *IEEE Symposium on Field-Programmable Custom Computing Machines* 2002

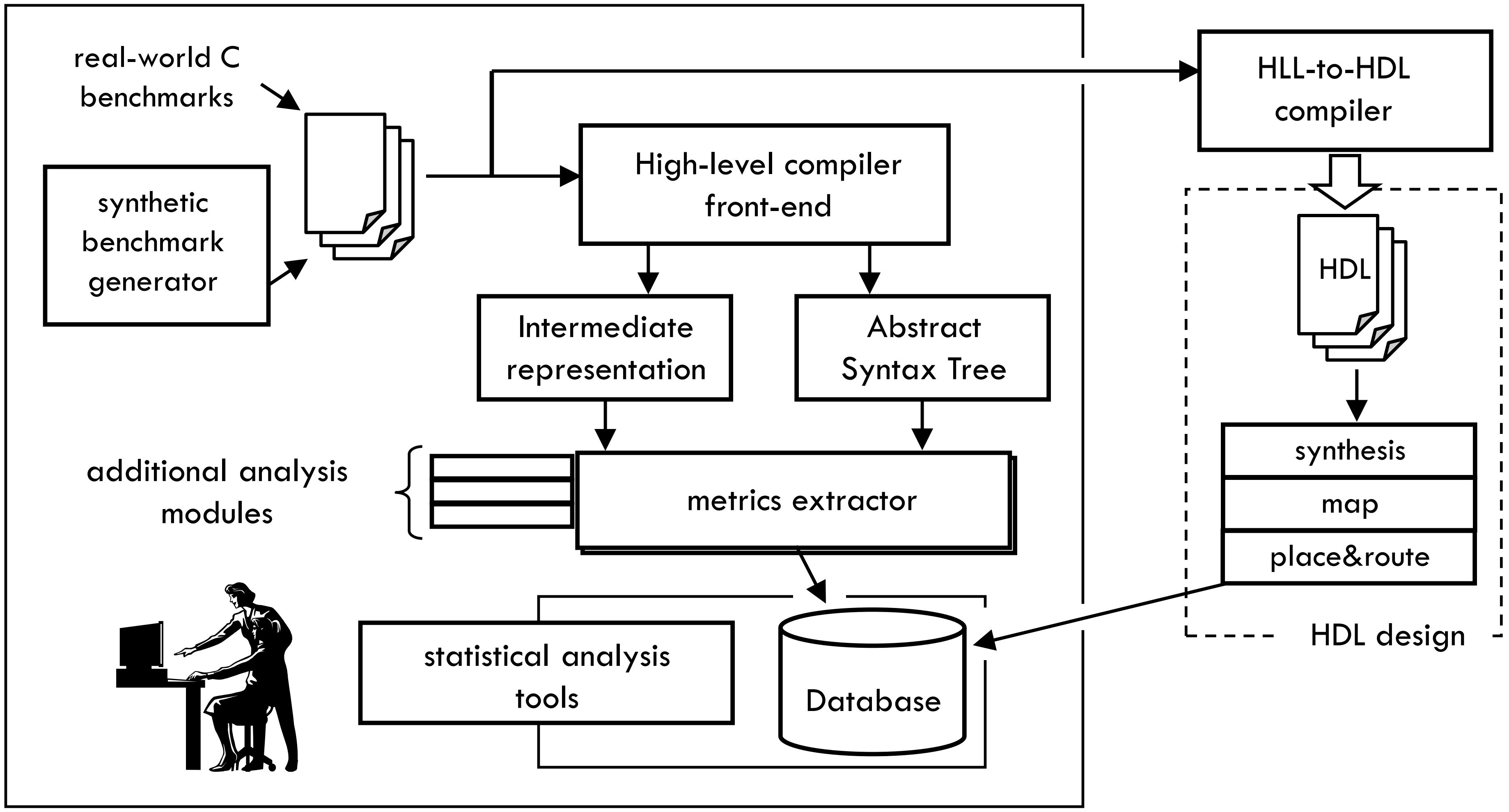
[S03] B. So, P. Diniz, and M. Hall, "Using estimates from behavioral synthesis tools in compiler-directed design space exploration", procs. of the 40th *Design Automation Conference*, 2003.

[M07] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning" procs. of 17th *International Conference on Field Programmable Logic and Applications*, 2007

The proposed framework

- Main objective: proposing a general framework for the systematic study of early prediction metrics
- the framework should be mostly independent of the HLL-to-HDL translation environment
  - but should be extensible and allow additional third-party modules
- should also provide software components for fast complexity estimation during HW/SW co-design
- Relies on a generic *intermediate representation*, together with a library of optimization algorithms, to approximate the behaviour of the hardware compiler we wish to model

Architecture


real-world C benchmarks
synthetic benchmark generator
High-level compiler front-end
HLL-to-HDL compiler
Intermediate representation
Abstract Syntax Tree
HDL
additional analysis modules
metrics extractor
synthesis
map
place&route
statistical analysis tools
Database
HDL design

Architecture

- A representative benchmark of high-level applications
- converted into Static Single Assignment (SSA) form
  - enables a direct manipulation of the C/DFG extracted
  - captures a number of aspects of the program structure, as seen by the HLL-to-HDL compiler, for quantitative evaluations: cyclomatic complexity, number and nature of variables, etc.
  - the metrics extractor processes the intermediate representation and the abstract syntax tree (AST)
- Data are stored in a DB and examined by means of suitable statistical analysis tools
- Highly modular and extensible structure
  - can define an ad-hoc module plugged in the framework, that can access the intermediate representation and the AST through a standardized interface
  - extension modules for matching specific HLL-to-HDL translators
- The framework also includes a module for the automatic generation of synthetic benchmarks

Implementation

- The metrics extractor module is built on top of the *Low-Level Virtual Machine* (LLVM) compiler infrastructure
  - allows the intermediate code to be inspected, and possibly modified, at various levels of granularity
  - provides dataflow information in SSA form
  - structured in self-contained units, or "passes"
  - the pass-based mechanism makes the framework easy to extend as passes can be dynamically loaded

Implementation

- kernel of the metrics extractor module

```
void AnalyzeProcess() {
    ...
    VisitFunction(CP, F->getBody());
    ...
}
void AnalyzeBitcode() {
    ...
    {
         cerr << "Evaluating metrics ...\n";
         PassManager PM;
         PM.add(new CoBitCodePass(this));
         PM.run(BC);
    }
    ...
}
```

Implementation

- The framework is designed to take into account, initially, as many metrics as possible. Some examples:

Name	Meaning
`ops_div_flt`	floating point division operations
`ops_mul_flt`	floating point multiplication operations
`ops_add_flt`	floating point sum and subtraction operations
`ops_div_iN`	integer division (N bits wide) operations
`ops_mul_iN`	integer multiplication (N bits wide) operations
`ops_add_iN`	integer sum (N bits wide) operations
`ops_bra_cond`	conditional branches
`blk_dep_tot`	weighted sum of the nesting level of all basic blocks
`loops`	number of loops in the CFG
`mem_flt_tot`	total bits consumed by floating point variables
`mem_int_tot`	total bits consumed by integer variables

Implementation

- designed to interoperate with commercial synthesis, mapping, and place&route tools
  - includes some ad-hoc utilities to automatically extract post-synthesis results from the internal reports
  - e.g. XML-based reports in the *Xilinx ISE*
- Results stored in an embedded *SQLite* database
- Linear regression performed on the database based on the *R* freeware statistical environment
  - define a linear model characterizing the relationships between post-synthesis results and metrics
  - identifies a subset of the metrics, along with a set of weights
  - provides a linear formula approximating the hardware complexity parameter under study (e.g. #LUTs, #FFs, …)

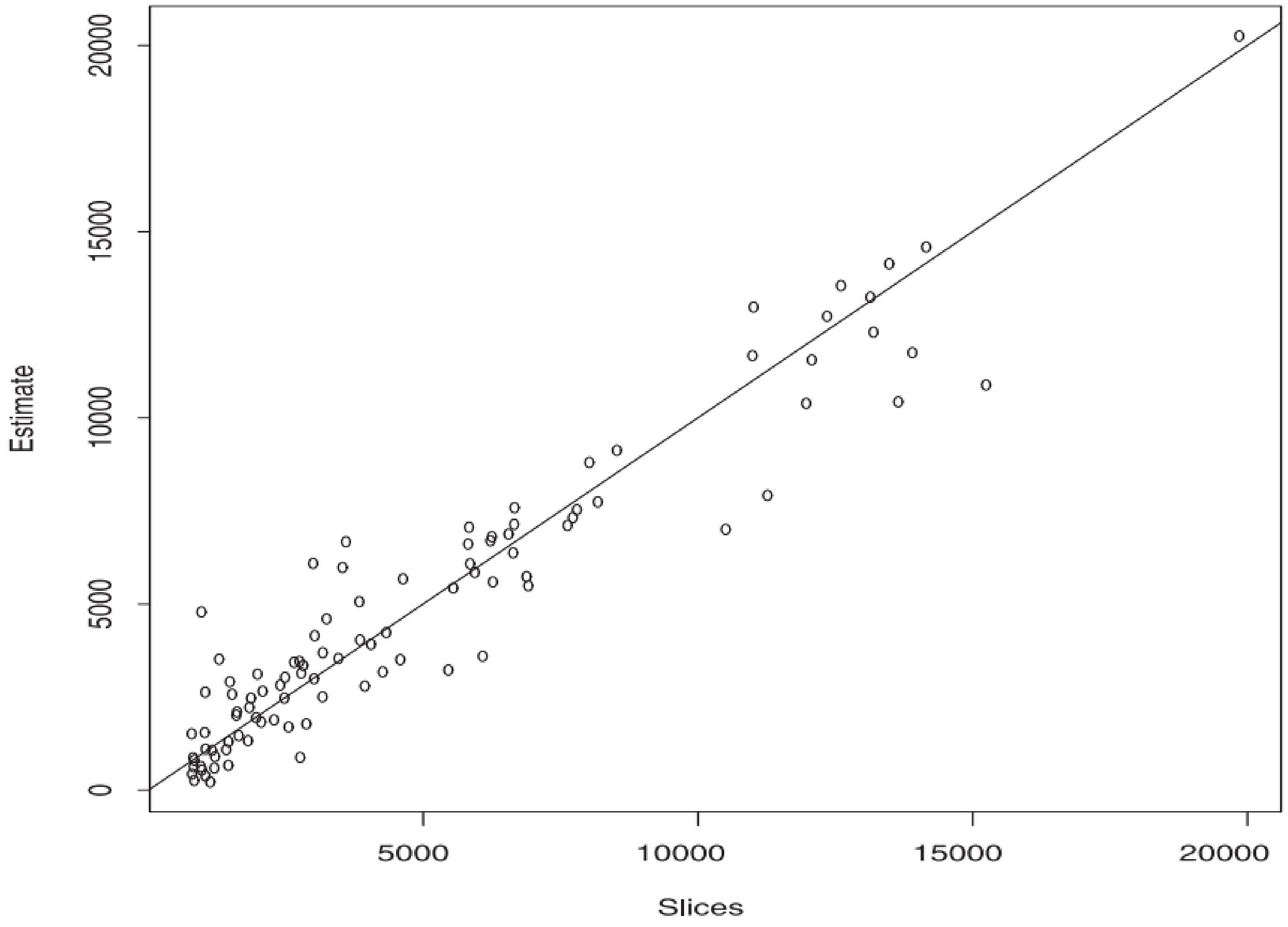
Results

- Targeted *Impulse*C and its associated *CoDeveloper* HLL-to-HDL compiler for tests
- Added *Impulse*C-specific modules to the framework
  - e.g., for stream operations and arbitrary-precision integer types
- *Xilinx XST* synthesis, mapping, the place&route tools
- *Virtex-4 Xilinx* FPGA family as the target technology, namely a *Virtex4FX12* device
- A large set of synthetic benchmarks (around 200) and several *Impulse*C reference designs

Results

- statistically significant dependences are more evident for larger designs
  - due to overhead and fine-grain optimizations
  - We only focused our attention on designs larger than 750 slices

Results: FPGA slices

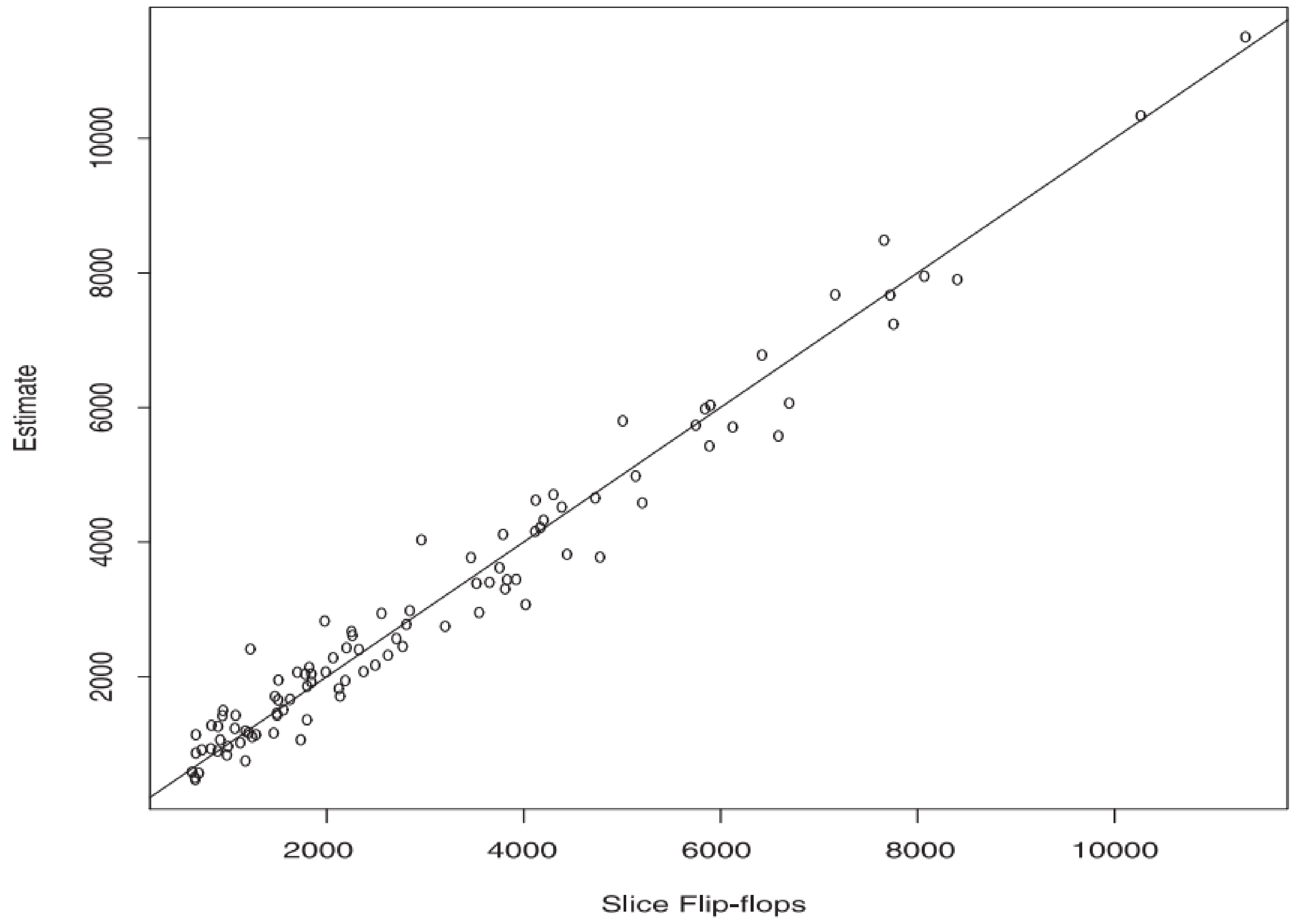


Results: FPGA slices

- linear regression indicates that the most relevant metrics are:

Metrics name
ops_mul_flt
ops_div_flt
ops_mul_i64
ops_mul_i32
ops_add_flt

Results: flip-flops

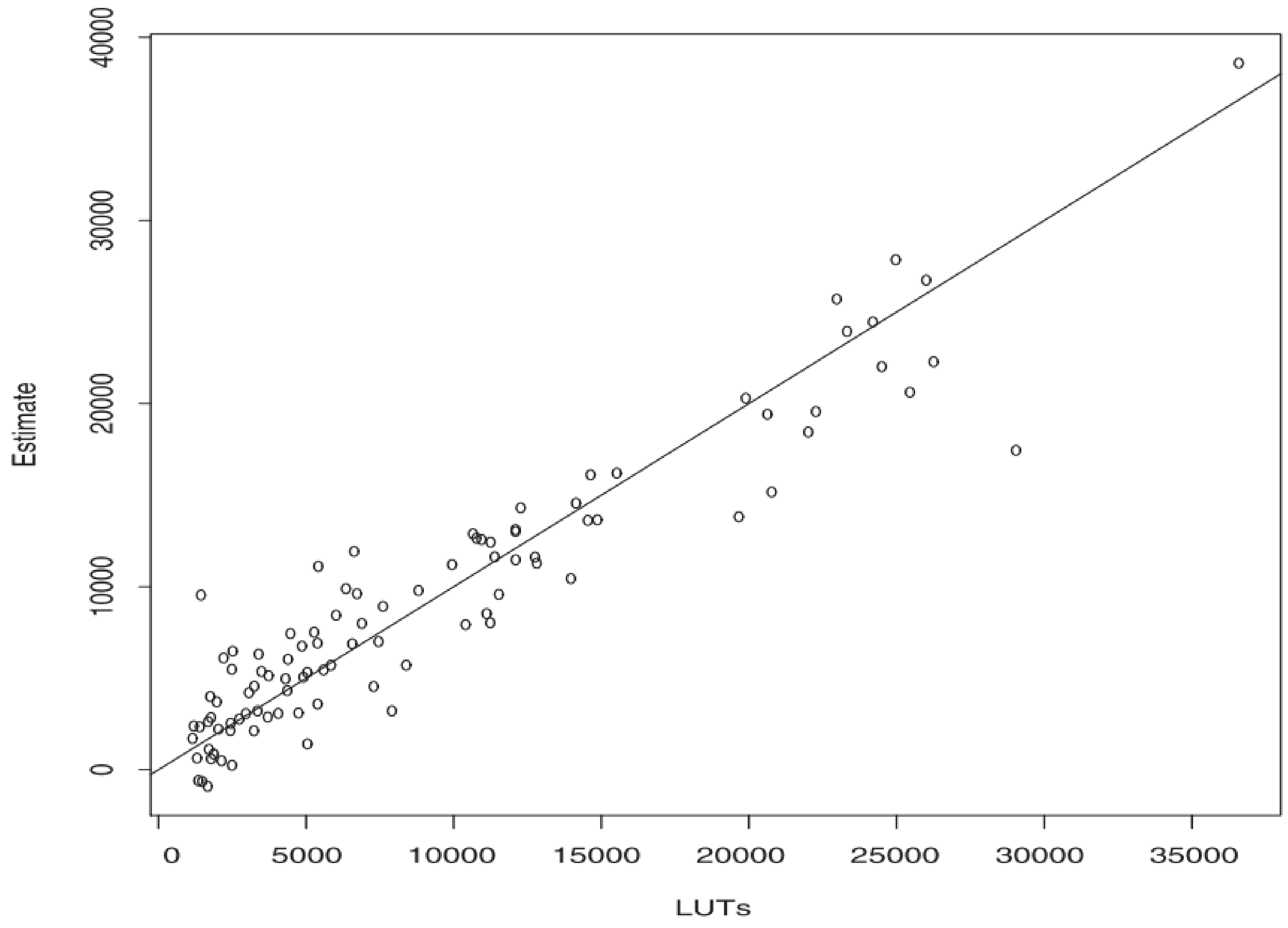


Results: flip-flops

- linear regression indicates that the most relevant metrics are:

Metrics name
mem_flt_tot
mem_int_tot
ops_div_flt + ops_mul_flt
ops_div_i64 + ops_mul_i64
ops_add_flt

Results: LUTs



Results: LUTs

- linear regression indicates that the most relevant metrics are:

Metrics name
ops_mul_flt
ops_div_flt
ops_mul_i64
ops_mul_i32

Comparisons

- first proposal investigating early prediction metrics for hardware complexity based on source-level analysis, along with [M07]
  - works in [K02], [S03] are tightly integrated with a specific hardware compiler and are <u>not</u> general
- Results are more accurate than [M07]
  - their analysis is only based on metrics borrowed from the software engineering domain (code size, structure of branches, etc.)
  - in [M07], average errors on fitted data are equal to 85%, 47%, and 102% for slices, flip-flops, and LUTs, respectively
  - average errors are equal to 31%, 15%, and 42% , respectively, in our work

Conclusions and future work

- Early estimation of hardware complexity essential in design exploration for HW/SW partitioning
- We proposed a general framework to build complexity predictors for third-party HLL-to-HDL translators
- We showed that it is possible to analyze the applications at the source level achieving estimates of reasonable quality
  - work on an *intermediate*, language-independent representation
- Results show that the framework is accurate and very fast:
  - For the benchmark considered, 20 hours for synthesis vs. 62 seconds for source-level estimates
- Future work:
  - extend the metrics extractor module with more sophisticated analysis passes
  - e.g. a data dependence analysis pass to anticipate specific optimizations performed by the map tool

Thank you for your attention!