YARDstick: Automation tool for custom processor development

Nikolaos Kavvadias and Spiridon Nikolaidis

nkavv@physics.auth.gr

Electronics and Computers Lab – Dept. of Physics – Aristotle University of Thessaloniki – Greece

http://electronics.physics.auth.gr/

Abstract

YARDstick is a building block for ASIP development, integrating application analysis, custom instruction generation, selection and synthesis with user-defined compiler intermediate representations.

1. Motivation

ASIPs (Application Specific Instruction-set Processors) play a central role in modern embedded SoCs replacing hardwired solutions which offer no programmability for enabling reuse or encompassing late specification changes. ASIPs are tuned for cost-effective execution of targeted application sets. An ASIP design flow involves profiling, architecture exploration, generation/selection of instruction-set extensions (ISEs) and synthesis of the corresponding hardware while enabling the user taking certain decisions.
It is often in ASIP design that certain practical issues arising from seemingly invariant elements of the design flow are not addressed:

a) Assumptions of the IR (intermediate representation) to which the application code is mapped directly affect solution quality as in the case of ISE synthesis.
b) The exploration infrastructure tied up to the conventions of software development tools.
c) Adaptability to different compilers/simulators.
d) Support for low-level entry for application migration within a processor family and reverse engineering.

YARDstick integrates custom instruction generation, selection and synthesis techniques with a flexible IR infrastructure that can reflect certain designer decisions that is cumbersome to apply otherwise. For example using an IR with intrinsic support for partial predication and bit-level operations may yield significantly different ISEs to the case of an unaugmented IR. YARDstick provides profiling facilities for determining static and dynamic application metrics such as data types, memory hierarchy statistics, and execution frequencies. Application entry can be either high-level (e.g. ANSI C) or low-level (assembly code). A number of recent ISE identification and selection techniques have been implemented while hardware estimators (speedup, area) and bindings to hardware synthesis from CDFGs are provided.

2. YARDstick details and usage

The main role of YARDstick is to facilitate design space exploration (DSE) in heterogeneous flows for ASIP design where the development tools (compiler, binary utilities, simulator/ debugger) in many cases, lack DSE capabilities and/or have been designed with different interfaces in mind. Thus, it is often that significant development effort is required in adding features as afterthoughts and dealing with interoperability issues, especially at the *compiler* and *simulator boundaries*.
To overcome these problems, YARDstick employs the **ByoX** (Bring Your Own Compiler and Simulator) kernel providing:

- An IR specification format called **BXIR** covering primitive operation syntax, semantics and other quantifiers (as latency and area of hardware implementations for IR operators).
- A simple file interface for a flat CDFG (with/without SSA) format of application IR called **ISeq**. The results of compiler analyses (e.g. register liveness, natural loops) can be passed to ByoX as defined by their corresponding BNFs.
- An IR manipulation API for writing external analyses and optimizations.

In ISeq, the following application information is recorded:

- The global symbol table
- The procedure list, consisting of data dependence entries and a statement list per procedure. Different facets of the local symbol table (e.g. single-per-direction vs. multiple variable definition points) can be generated.

At the simulation boundary, YARDstick expects information on the dynamic profile of the application (basic block execution frequencies, program trace, cache memory access statistics) on a target machine. From within YARDstick, static and dynamic application metrics can be evaluated and visualized.
Further, a number of instruction generation/selection methods have been implemented for the ByoX API. Instruction generation involves the identification of MIMO (Multiple-Input Multiple-Output) ISeq patterns that may span across basic block (BB) boundaries, under user-defined constraints. When invoked, a custom instruction library is constructed from the ISeq patterns, which can be filtered during the process for removing redundant cases. A subset of the library can be selected by using either a greedy selector (with configurable priority metric) or a 0-1 knapsack-based one. Since custom instructions can be expressed in ISeq, pattern libraries can be imported to YARDstick.
Application CFGs, BBs and patterns can be processed by a number of backends for exporting to:

- ANSI C subset code for incorporation to user tools (simulators, validators etc).

* This work was supported by the General Secretariat of Research and Technology of Greece and the European Union.

- GDL (VCG) and dot (Graphviz) files for visualization.
- An extended CDFG [1] format for scheduling and translation to synthesizable VHDL (BBs and patterns).
- GGX XML [2] files for algebraic graph transformation.

Assembly-level code generated by GCC [3] and COINS [4] can be imported, assuming a working SALTO [5] backend and the associated set of SALTO passes for the translation.

3. Case studies

For proof-of-concept, we have evaluated YARDstick with the specification of the unmodified SUIFvm IR [6] and a set of incremental extensions to it (summarized in Table 1) as well as DLX assembly viewed as a form of machine-level IR.

Table 1: Different IR settings for ISE generation

IR	Operations
SUIFvm	The operation set defined in [6]
Extended SUIFvm (partial list)	type conversion (sxt, zxt), partial predication (select), operations-by-constant (mulc, lslc, asrc, lsrc), bit manipulation (bitinsert, bitextract, concat)
iDLX	The DLX integer instruction set

Preliminary test results regard analysis and exploration of embedded processing kernels such as motion estimation/ compensation, DSP transforms, rc5, aes, raiden and applications as adpcm, crc, and jpeg. Fig. 2 illustrates backend generated sample files, more specifically an expanded CFG view for raiden (2a), and a pattern (2b) identified in a BB of Fig. 2a.

4. Environment

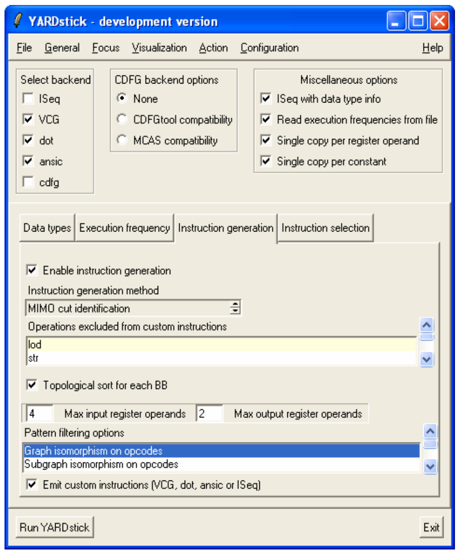
YARDstick has been used along with the SUIF/Machine-SUIF [6], GCC [3], and COINS [4] compilers and (currently) the ArchC [7] simulation framework. YARDstick functionality is accessible through a cross-platform GUI compatible to recent Tcl/Tk versions (8.4.x). A screenshot of the GUI can be seen in Fig. 1.
Supported platforms include GNU/Linux (RedHat 9.0), Cygwin and Win32 (Windows/XP SP2) on x86-compatible processors.

4. Future work and conclusion

Planned additions to YARDstick include support for PDG (program dependence graph) IRs, automatic translation between BXIR and SALTO/compiler backends, and interprocedural analyses and optimizations.
Overall, YARDstick aims in enabling more options for the ASIP designer by allowing crucial parts of development frameworks to be independent of compiler and simulator idiosyncrasies.

5. References

[1] CDFG toolset. http://poppy.snu.ac.kr/CDFG/cdfg.html
[2] The AGG homepage. http://tfs.cs.tu-berlin.de/agg/
[3] GCC. http://www.gcc.org
[4] The COINS project. http://www.coins-project.org
[5] SALTO. http://www.irisa.fr/caps/projects/Salto/
[6] Machine-SUIF. http://www.eecs.harvard.edu/hube/research/machsuif.html
[7] The ArchC resource center. http://www.archc.org



Select backend
- [ ] ISeq
- [✓] VCG
- [✓] dot
- [✓] ansic
- [ ] cdfg

CDFG backend options
- [✓ None]
- CDFGtool compatibility
- MCAS compatibility

Miscellaneous options
- [✓] ISeq with data type info
- [✓] Read execution frequencies from file
- [✓] Single copy per register operand
- [✓] Single copy per constant

- [✓] Enable instruction generation

Instruction generation method: MIMO cut identification

Operations excluded from custom instructions:
lod
str

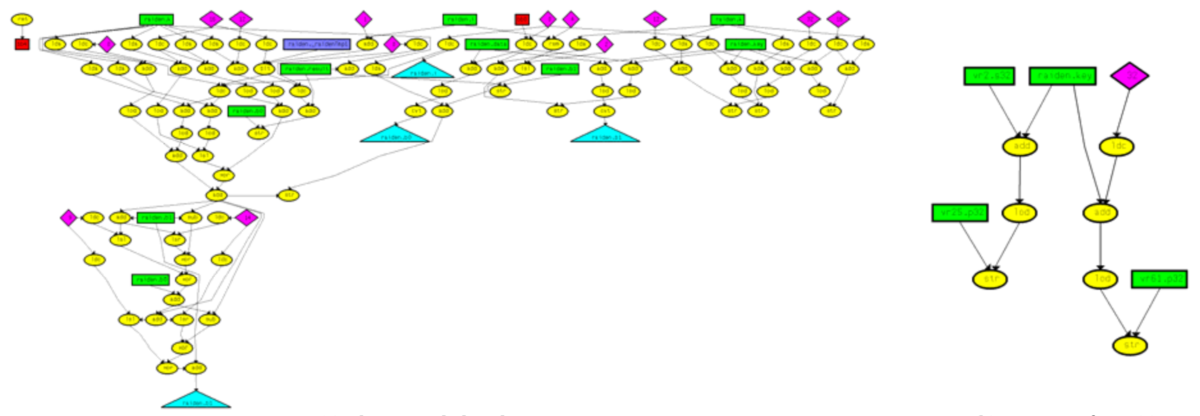
- [✓] Topological sort for each BB

4 Max input register operands 2 Max output register operands

Pattern filtering options:
[✓ Graph isomorphism on opcodes]
Subgraph isomorphism on opcodes

- [✓] Emit custom instructions (VCG, dot, ansic or ISeq)

Figure 1: Screenshot of the YARDstick configuration GUI



(a) The expanded raiden CFG (b) A pattern from (a)

Figure 2: Sample backend generated files.