YARDstick
Automation tool for custom processor development

Nikolaos Kavvadias and Spiridon Nikolaidis

Section of Electronics and Computers,

Electronics Lab, Dept. of Physics,

Aristotle Univ. of Thessaloniki, Greece

E-mail: *nkavv@physics.auth.gr*

What is YARDstick?

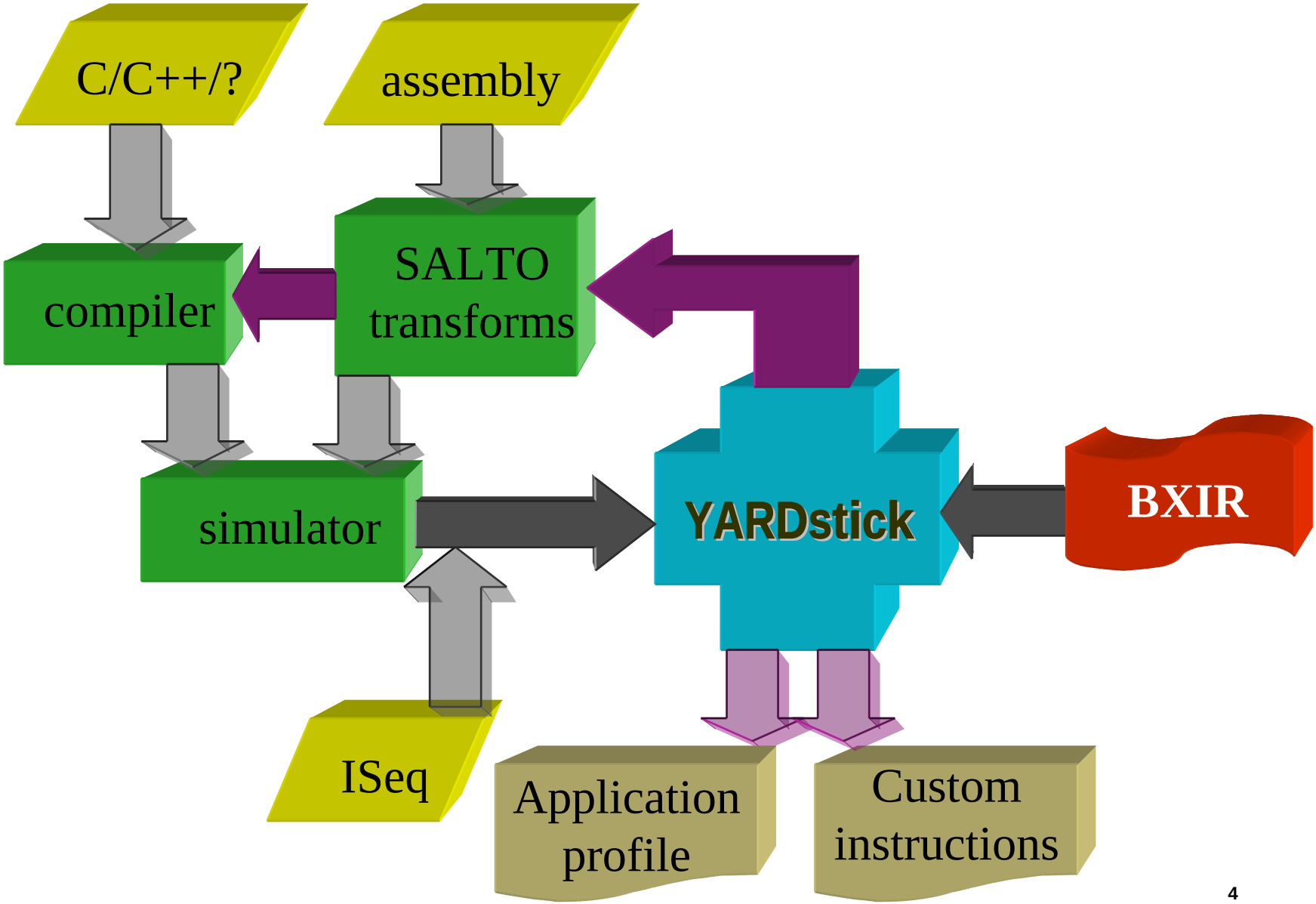
… a building block for ASIP development, integrating *application analysis*, custom *instruction generation*, *selection* and *synthesis* with user-defined (compiler) IRs

- **YARDstick provides a compiler- and simulator-agnostic infrastructure:**
  - **Separates design space exploration from compiler/simulator idiosyncrasies**
  - **Different compilers/simulators can be plugged-in**
  - **Both high- (ANSI C) and low-level (assembly for a machine/VM) input can be analyzed**

The need for YARDstick

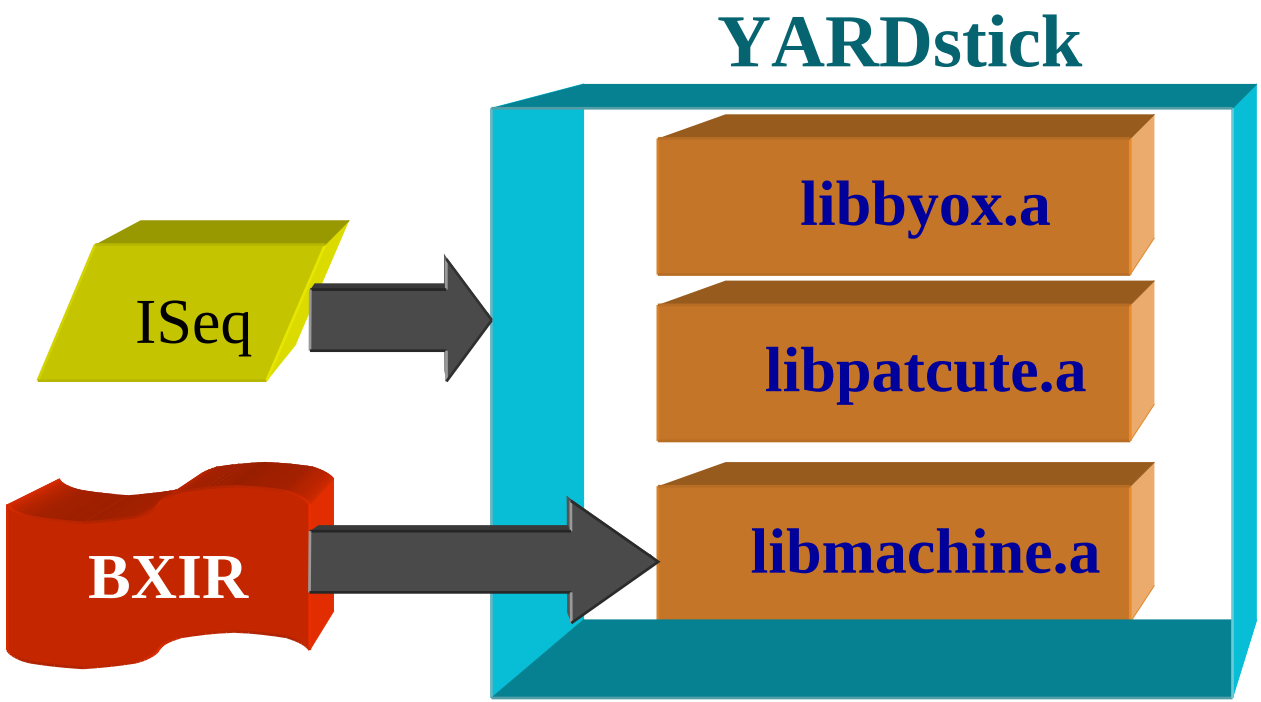
- **ASIP development flows for contemporary embedded SoCs involve**
  - **Architecture specification and DSE**
  - **Profiling**
  - **Exploitation of results: custom instruction (-set extension) generation and synthesis**
- **Certain practical issues in ASIP design are often neglected**
  - **Assumptions of the IR do affect the solution quality**
  - **The exploration infrastructure might be tied up to the conventions of SW development tools**
  - **Difficult to adapt to different compilers/simulators**
  - **Support for low-level entry is some times desirable (reverse engineering, legacy code porting)**
- **YARDstick has been designed with these issues in mind**

A high-level look to YARDstick



A closer look

- **YARDstick components**
  - **libByoX: “Bring Your Own Compiler-and-Simulator” kernel (target-independent)**
  - **libPatCUtE: Pattern-based Custom UniT Exploration (target-independent)**
  - **libmachine: Retargeted by a BXIR (ByoX IR) target architecture specification**



libByoX.a

- libByoX implements the core YARDstick API and provides frontends/manipulators for internal data structures
  - Frontends
    - ISeqinfo: parser for “Instruction Sequence” entries. ISeq records a “flat CDFG” form of the application IR
    - CFGinfo: parser for control-flow-graphs (SSA or non-SSA)
  - API
    - Interpretation of operations (“instructions”) and operation groups
    - Flexible manipulation of input/output/constant/symbol address operands (two facets of local operand list)
    - Parameterization for a template machine context without restrictions to its ISA (e.g. estimator for P. Biswas’ AFU memory model)

libPatCuTE.a

- libPatCuTE implements instruction generation and selection
- Contemporary instruction generation methods are supported (late 90's up to 2007)
  - MaxMISO (maximal subgraphs with a single-output node)
  - MISO exploration under user-defined constraints
  - MIMO (multiple-input, multiple-output) method based on Pozzi-Ienne-Atasu work with a fast heuristic
- Graph/subgraph matching for identification of redundant entire/partial custom instructions (CIs)
- Greedy instruction selection for cycle-gain, cycle-gain-per-area priority metrics
- Custom instruction/functional unit synthesis is supported by external tools (CDFG toolset)
- To extend with user-defined instruction generation methods link to libPatCuTe.a

BXIR format and libmachine.a

- **BXIR (ByoX IR) is a target architecture specification format**
- **Global-scope information**
  - **Data types**
  - **Operation grouping information**
- **Operation-level information**
  - **Operands**
  - **Semantics (not yet fully implemented)**
  - **Area/latency, Cycle timing**
- **Examples: SUIFvm (basic, enhanced), DLX (integer subset)**
- **libmachine.a is the only YARDstick component that needs retargeting for a user-defined target architecture**

```
operator VMselect {
  mnemonic select;
  input src0 src1 src2;
  output dst0;
  area = 302.3;
  latency = 0.097;
  cycles = 1;
}
```

YARDstick capabilities

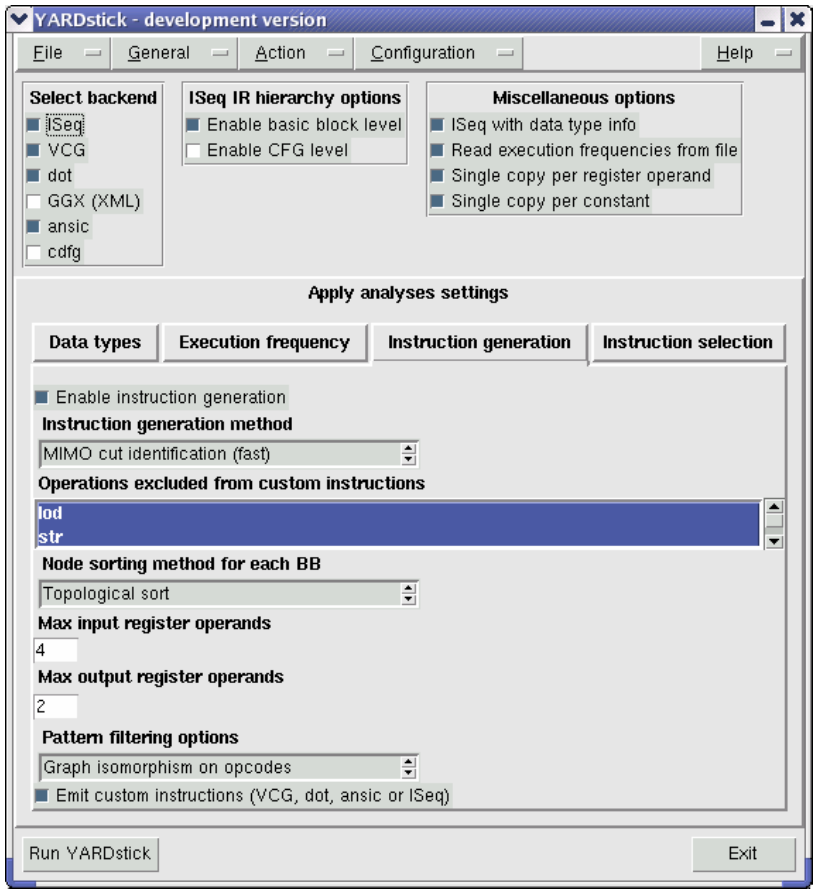
- Program analysis for C/assembly/ISeq (native format) applications
  - Data type and operation-level statistics
  - Basic block execution frequency analysis for identifying hot-spots in the application
- Visualization (VCG, Graphviz) of BBs, CFGs, and CIs
- Export to the GGX XML format which is supported by the AGG attributed graph transformations
- ANSI C output (BBs, CIs) for use in simulators
- Custom instruction analysis
  - Speedups and instruction selection analysis (to identify incremental number of CIs and the associated costs)
  - Visualization and export to the native format
- Interoperability with GCC, ArchC.

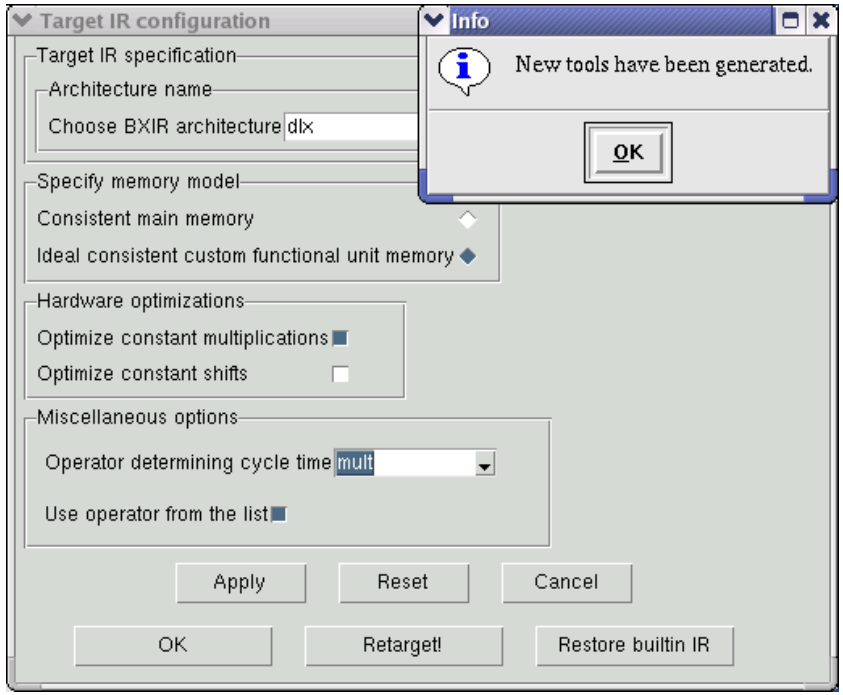
Using YARDstick

- **YARDstick hides complexity**

- **A sample YARDstick session should look like this:**
  1) **Configuration for the target architecture and the application program of interest**
  2) **ISeq/C/"assembly" file entry**
  3) **Run the compiler/simulator (C/asm applications)**
  4) **Load the resulting ISeq file (omit if done on step 2)**
  5) **Select backends of your choice**
     - **ISeq (these files can be input to YARDstick at a subsequent stage, if desired)**
     - **VCG, Graphviz dot for graph visualization**
     - **GGX XML for AGG**
     - **CDFG for the CDFG toolset (builtin architecture only)**
     - **ANSI C (builtin architecture only)**
  6) **View results within the Results Browser.**

YARDstick screenshots (1)

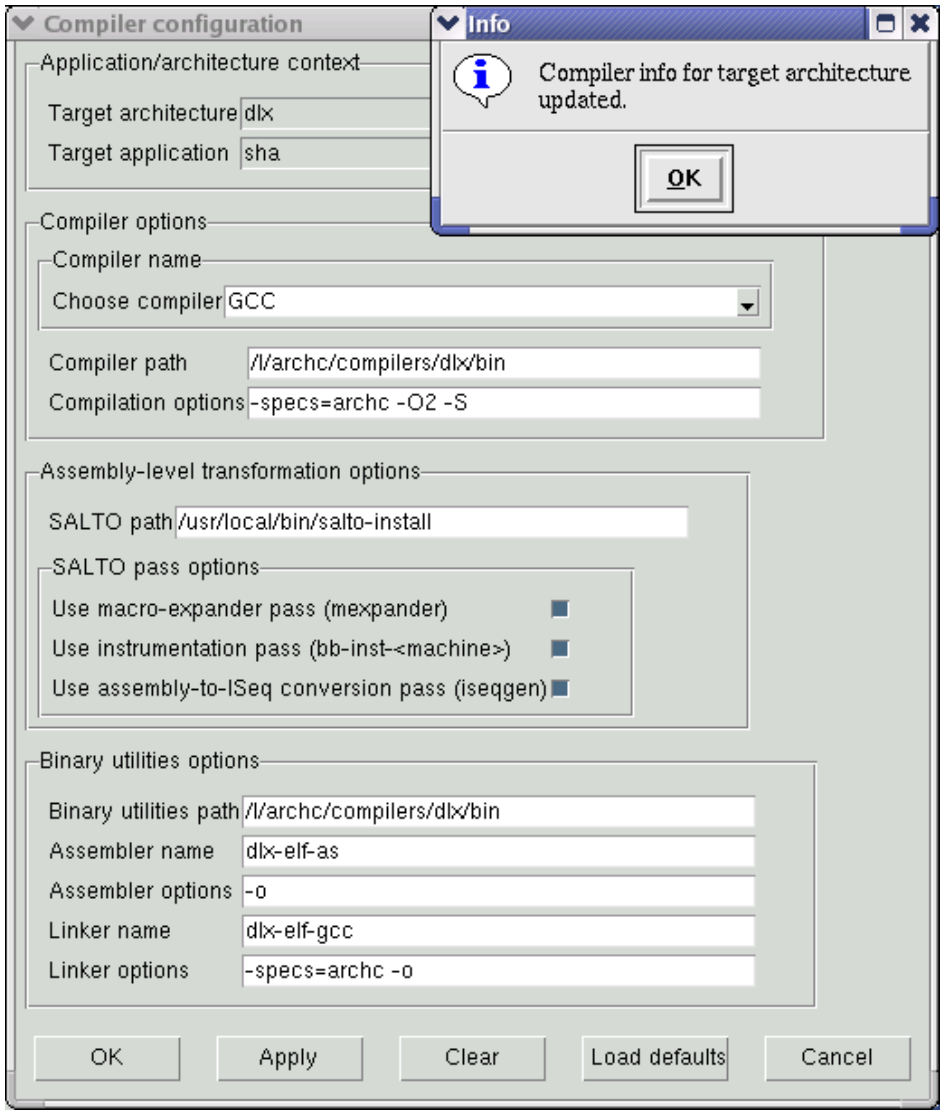




The main YARDstick screen

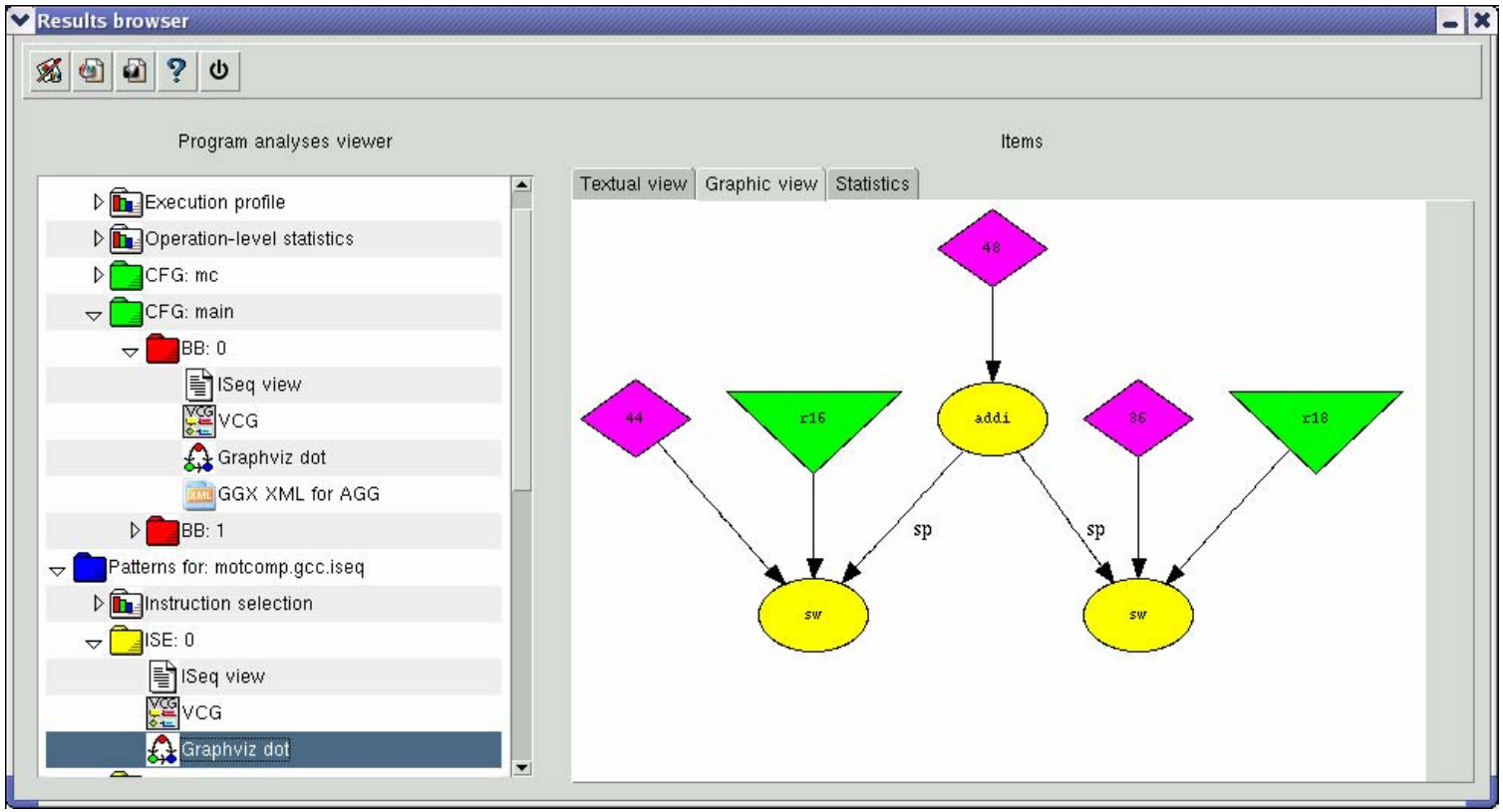
Target architecture configuration

YARDstick screenshots (2)



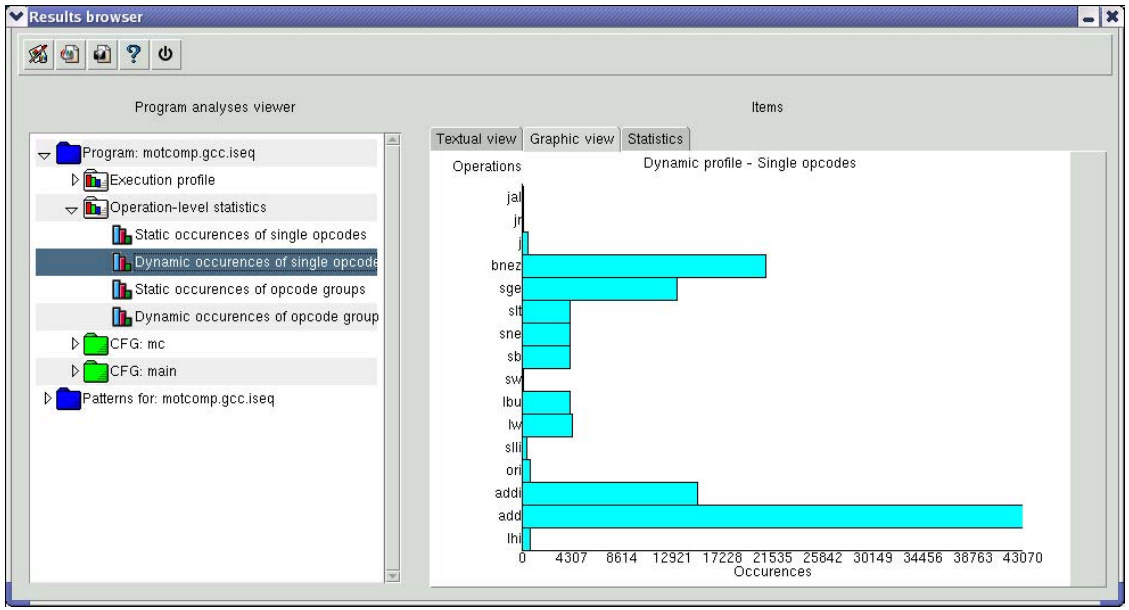
Compiler/binary tools configuration

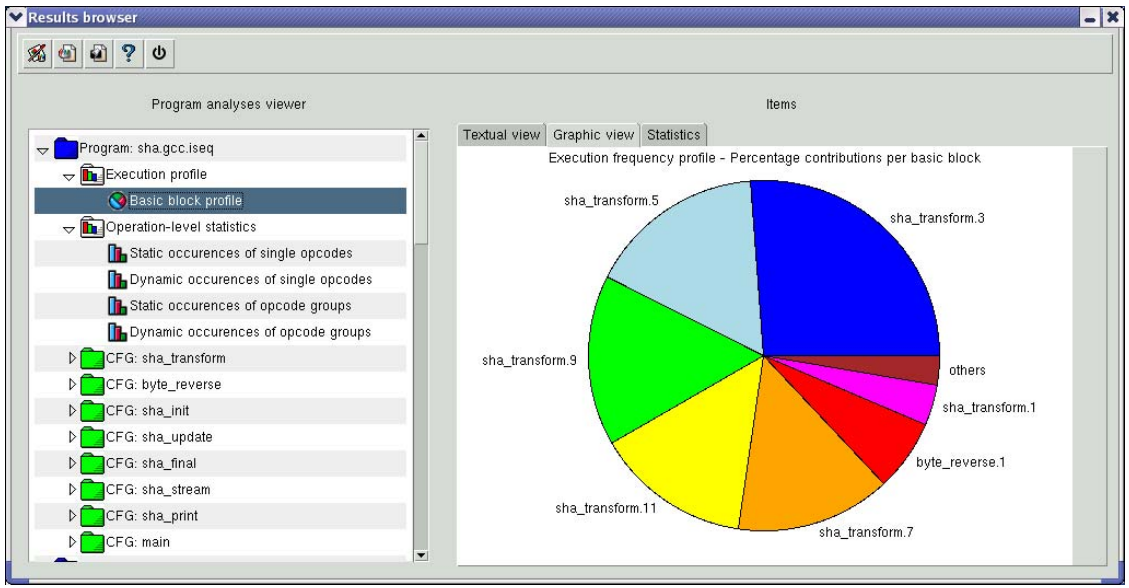
YARDstick screenshots (3)



Graphviz dot layout of a CI, by tcldot on a Tcl/Tk canvas

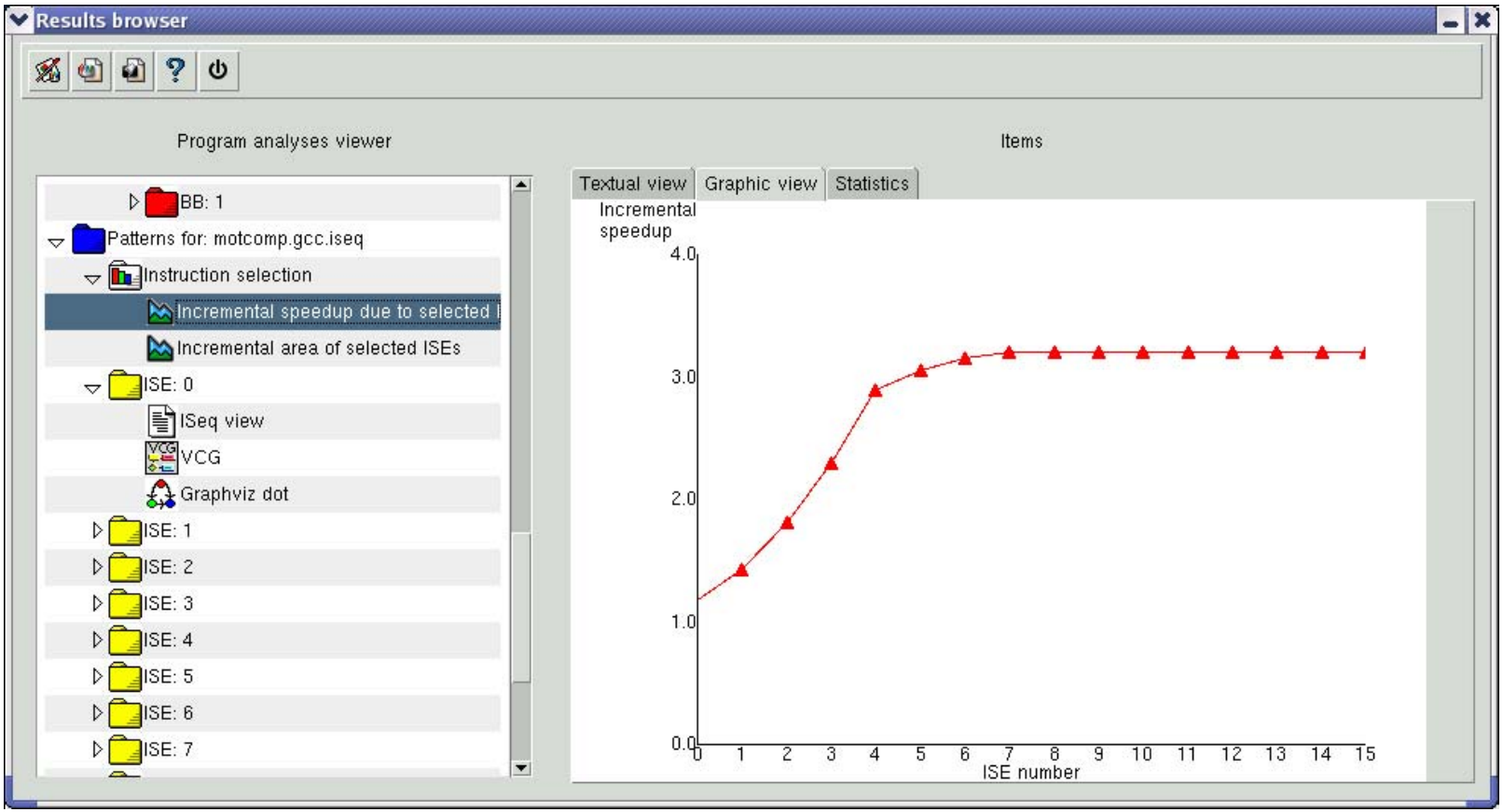
YARDstick screenshots (4)





Samples of dynamic statistics results

YARDstick screenshots (5)



Incremental speedup due to selected custom instruction-set extensions

The development of YARDstick

- Initially only used with SUIF 2.2/Machine-SUIF 2.02.07.15 (semi-complete SUIFrm backend)
- On-and-off work dating around Sept. 2004 (ISeqinfo frontend), heavy coding since Sept. 2006
- Current code base (mainline) around 20K lines (not accounting SALTO passes, GCC DLX backend etc)
- Developed in C, C++, bison/flex, Tcl/Tk
- Some features require external tools/libraries:
  - Mandatory: boost (1.33.0), vflib2
  - Optional: ArchC 1.6.0, SystemC 2.0.1, SALTO 1.4.1beta3, GCC, other compilers/simulators
- GUI (~5K) in Tcl/Tk 8.5.a5 (Tile, GRIDPLUS, other widgets)
- "External" tools as proof-of-concept (GCC/COINS DLX backends, DLX binutils/newlib/gdb port, simulators etc)

The road ahead…

- **A quick sketch of possible future enhancements to YARDstick**
  - **PDG/ whole-program/execution-trace view of the application**
  - **Automatic translation between BXIR, SALTO/compiler backends**
  - **Automatic targeting of compiler backend tools (some ideas are in place)**
  - **A more detailed (micro-architectural) view of target architectures**
  - **Adding analyses, new CI generation techniques, and backends (GAUT for CDFG synthesis)**
  - **Interfaces to more compilers (better COINS support), simulation engines (UNISIM?)**
- **…But what does "YARDstick" mean?**
  - **Well, it's a euphemism meaning "Your ASIP is Ready to Deploy". The "stick" suffix just got caught in the way ☺**