VISTA: A System for Interactive Code Improvement

Wankang Zhao, David Whalley,
Robert van Engelen, Xin Yuan, Kyle Gallivan
Florida State University

Baosheng Cai
Oracle Corporation

Mark Bailey
Hamilton College

Jason Hiser, Jack Davidson
University of Virginia

Douglas Jones
University of Illinois at Urbana-Champaign

Embedded Systems: a Difficult Target

- unusual architectural features
  - low overhead looping hardware
  - specialized address and arithmetic functions
  - highly irregular instruction sets
- stringent application constraints
  - real-time deadlines
  - absolute memory limitations
- efficient code requires specific user knowledge
  - value ranges
  - memory disambiguation
  - determining loop bounds

Choices for Coding Embedded Systems Applications

- high-level language
  - difficult to exploit special-purpose hardware
  - less control over performance
- assembly language
  - difficult to maintain and retarget
  - coding is slow
  - error prone
- hybrid at the module level
  - too coarse grain

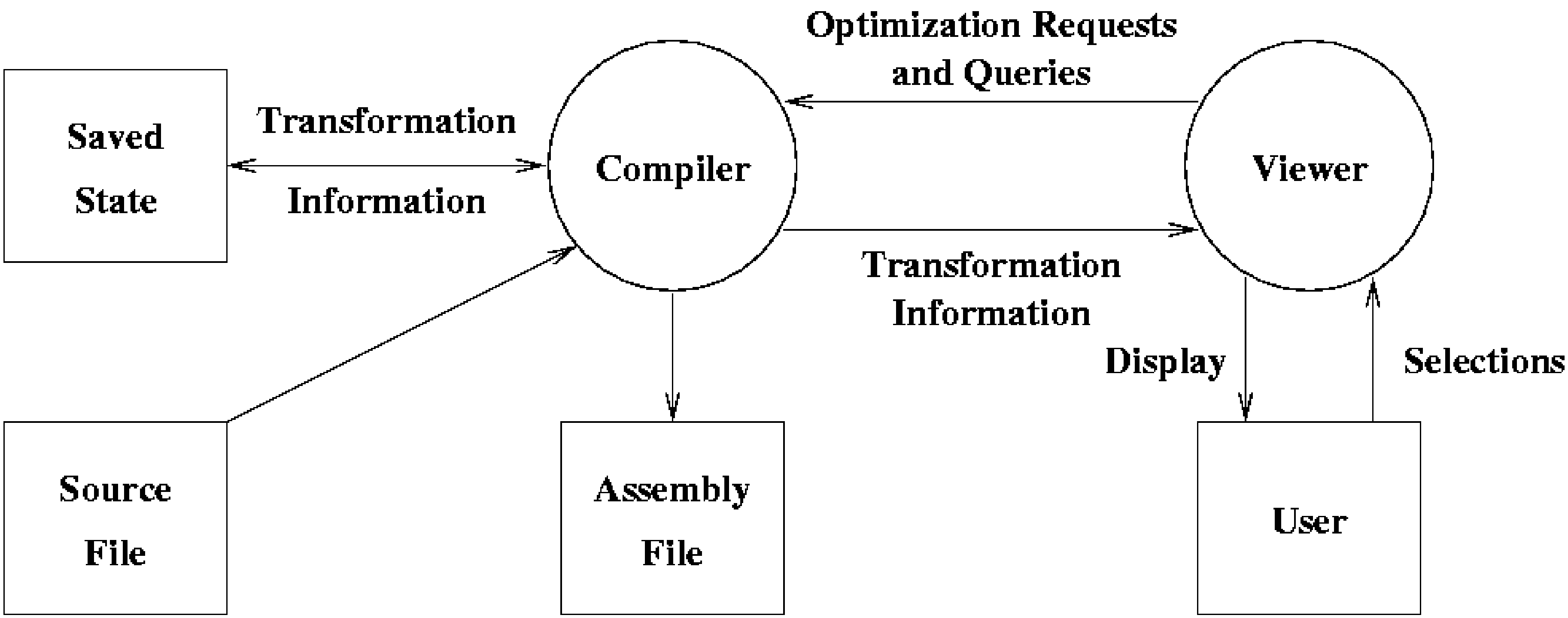
Interactive Code Improvement

- Application development in a high-level language
- Low-level code improvement assisted by developers
  - Selecting the order and scope of traditional optimization phases
  - User-specified code improvements
- User guided code improvement assisted by the compiler

Related Work

- Compiler debugging
  - XVPODB: Boyd, Whalley
- High-level parallelization of programs
  - Pat toolkit: Appelbe, Smith, McDowell
  - Parafrase-2: Polychronopoulos, Girkar, et al.
  - Pittsburgh system: Dow, Chang, Soffa
  - SUIF Explorer: Liao, Diwan, Bosch, et al.

VISTA: Vpo Interactive System for Tuning Applications



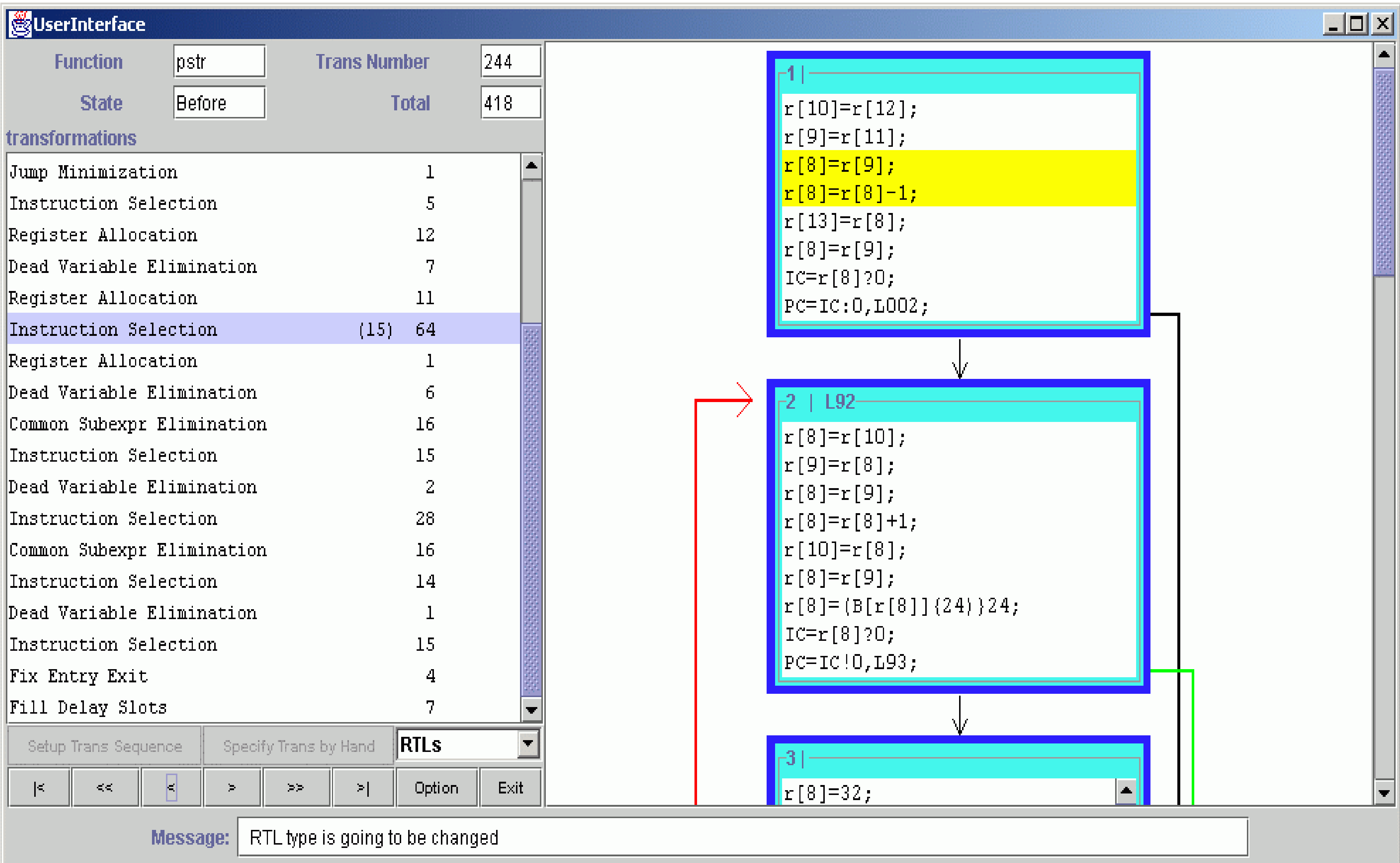
Features of the Environment

- View the representation of a function at any optimization point.
- Specify the order and scope of optimization phases.
- Specify code-improving transformations manually.
- Visualize performance of the application.
- Reverse previously applied transformations.
- Obtain information from the compiler.
- Specify improvements over multiple sessions.

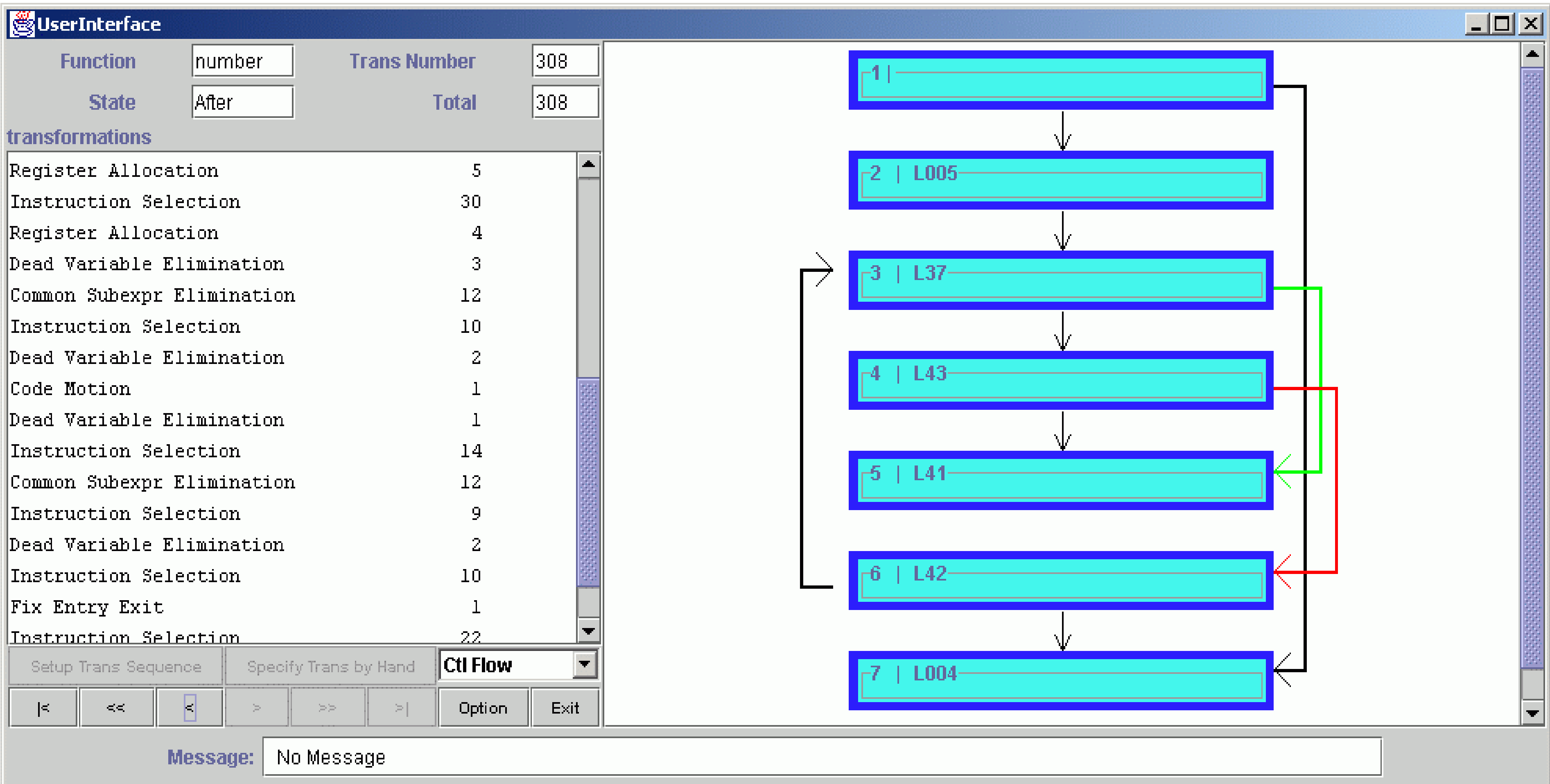
Viewing the Low-Level Representation

- Natural level for embedded systems performance tuning.
- Supports a variety of display options.
  - RTLs
  - assembly
  - control flow
- Eases debugging of compiler errors.
- Provides a better understanding of the code improvement process to a user.

History of Compilation Phases



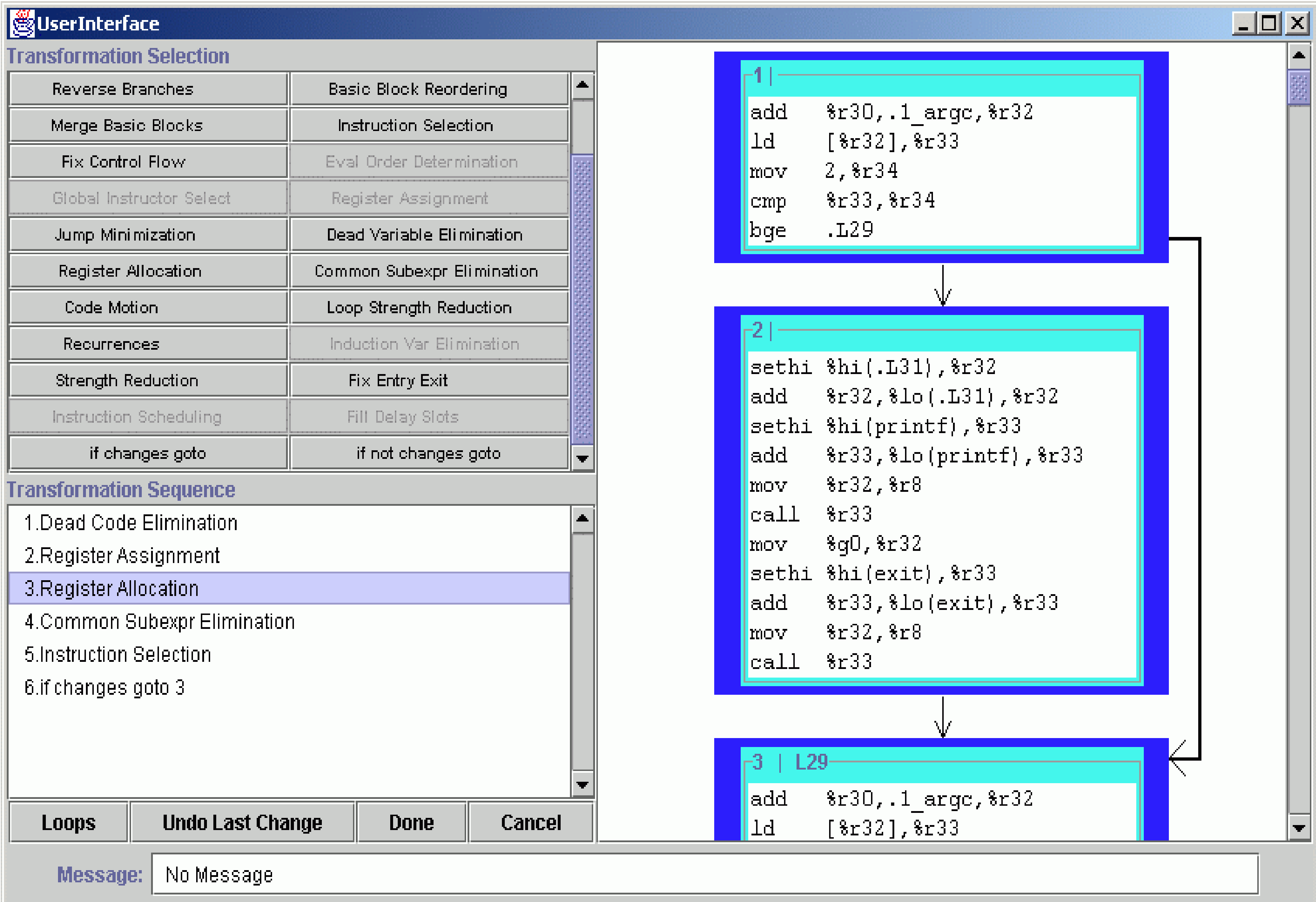
Control Flow: A Bird's Eye View



Specifying Compilation Phases

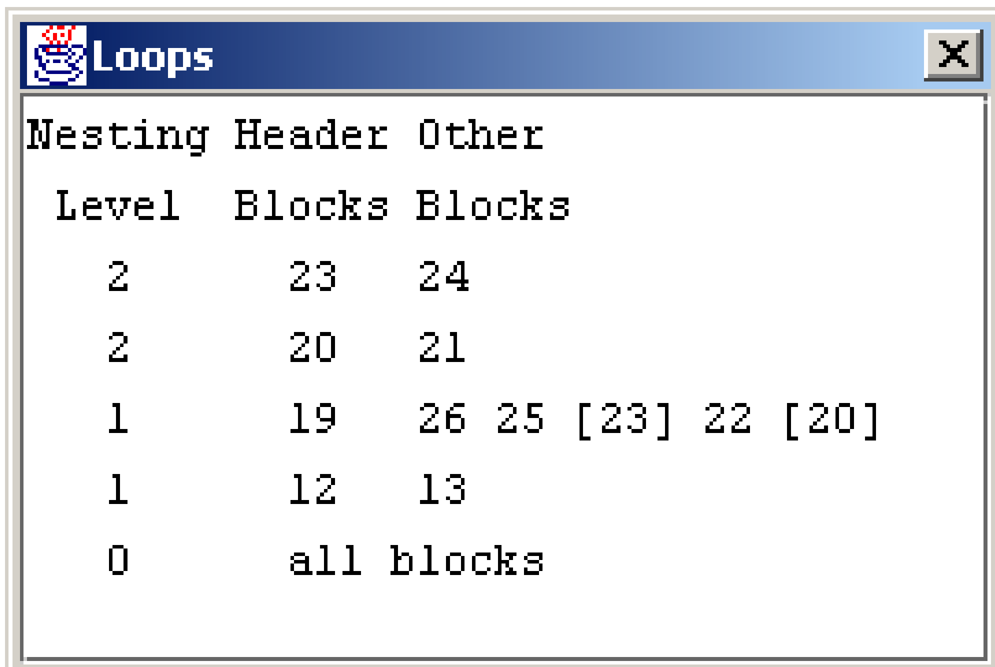
- Gives the user control over the code improvement process.
- Helps to address the phase ordering problem.
- Phases can be specified to be performed repeatedly until no more changes are made.
- Can limit the scope of the program representation where a phase is applied.
- Certain restrictions still have to be enforced.

Phase Order Control



Restricting the Scope of Phases

- set of basic blocks by clicking on each block
- set of loops by clicking on loops in the loop report

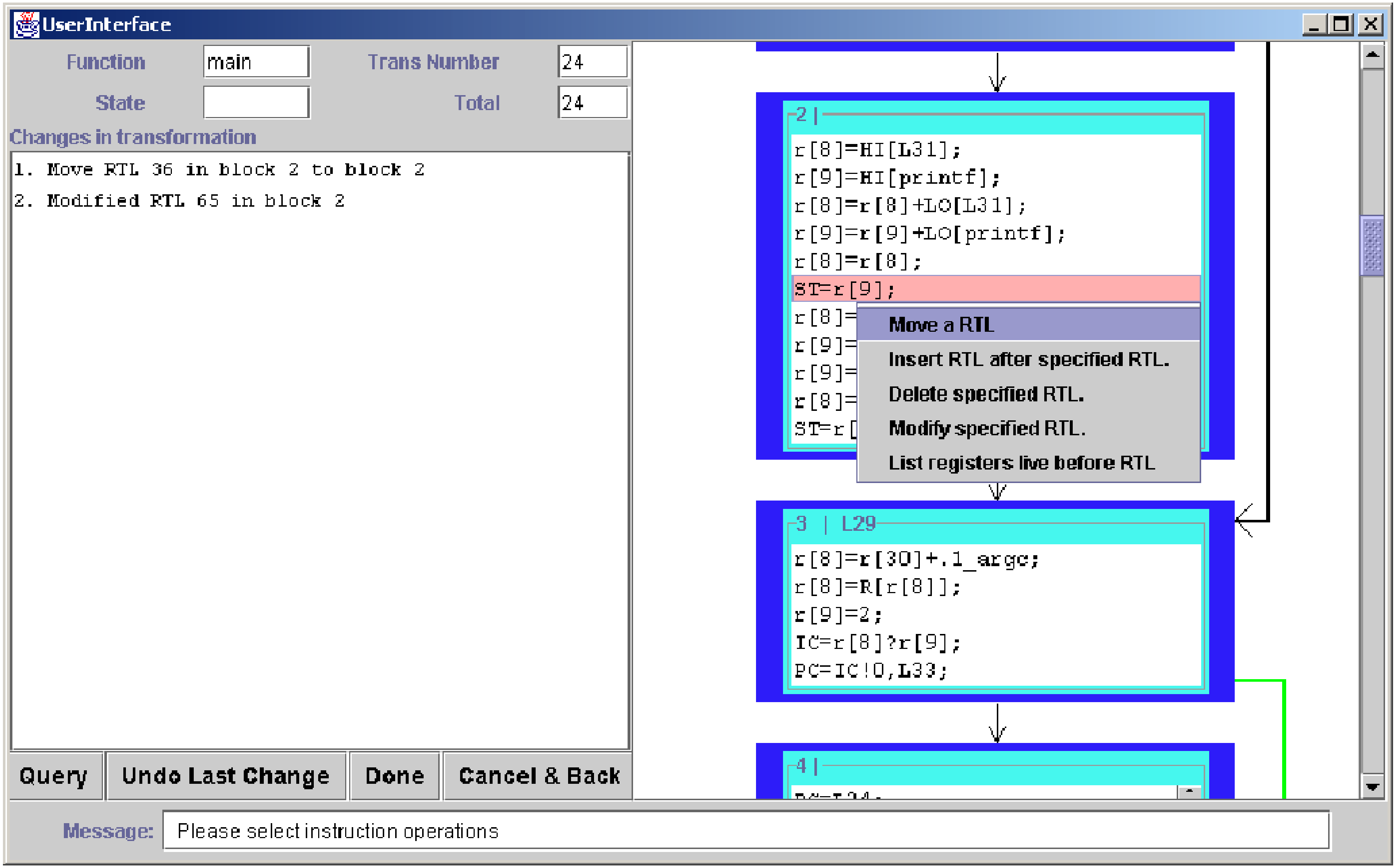

Loops

Nesting Level	Header Blocks	Other Blocks
2	23	24
2	20	21
1	19	26 25 [23] 22 [20]
1	12	13
0	all blocks	

User Specified Improvements

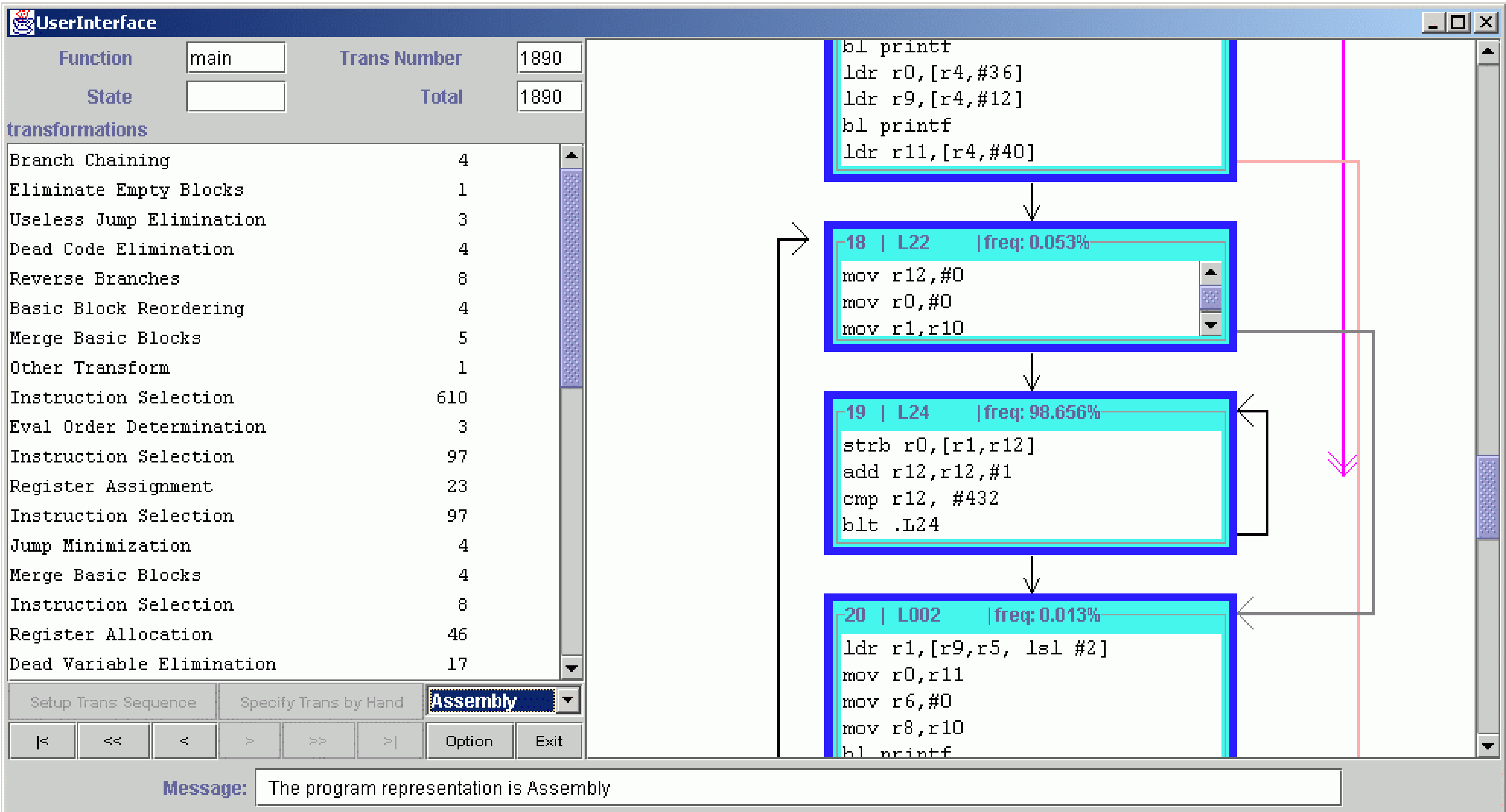
- Often difficult to exploit embedded features.
- User can tune compiler generated code.
- User can make queries to the compiler.
  - What registers are live at a given point?
  - Which blocks dominate a specified block?
  - What loops exist in the function?
  - ...
- Useful for prototyping code improvements.

Manually Specifying a Transformation

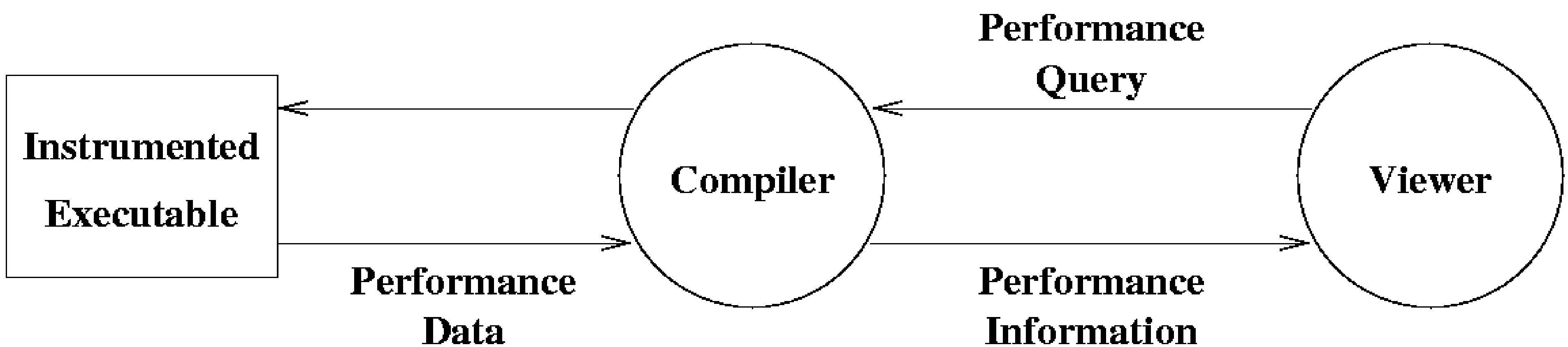


Visualizing Performance

- Can obtain performance measurements and can view them on blocks or loops.



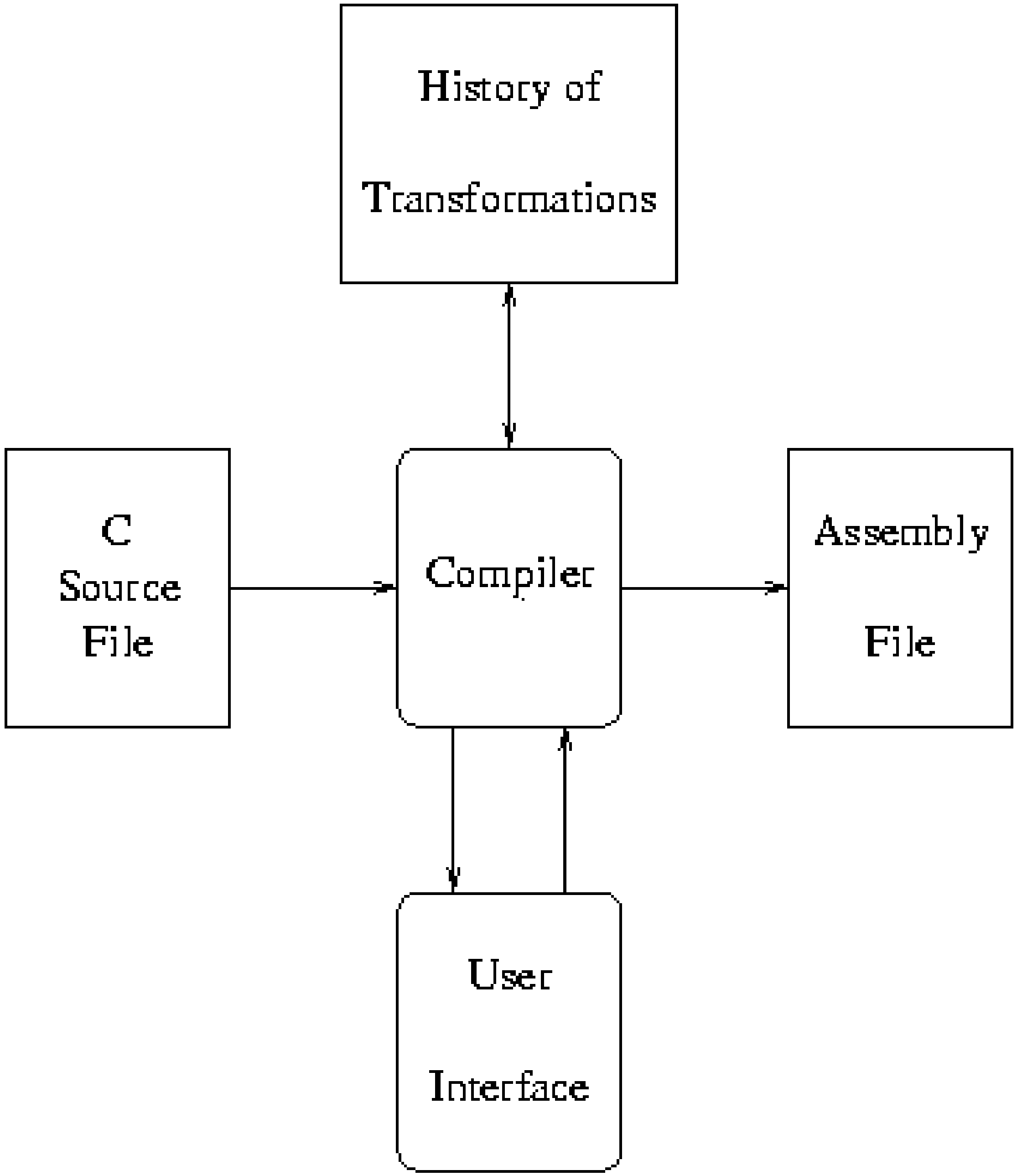
Performance Information Collection



Traversing Applied Transformations

- Can apply or undo transformations.
- Allows a user to experiment with different compilation phase orderings.
- All changes are stored.
- Changes, both compiler and user specified, are saved to a file.

Transformation History Is Saved



Implementation Issues

- Used Java for the user interface to enhance its portability.
- Communication between the compiler and user interface was accomplished using UNIX sockets.
- Analysis needed for or invalidated by each optimization phase had to be identified.
- Translators were required to convert a human specified RTL or assembly instruction into an encoded RTL.

Future Work

- Patterns for detecting code improvement opportunities.
- Show performance improvement.
- Support iterative compilation to meet specified constraints on speed, size, and power.
- Include a mapping between source and assembly.

Conclusions

- Useful for effective embedded systems development.
  - Benefits of coding in a high-level language.
  - Flexibility of coding in assembly.
  - Compiler can exploit user knowledge.
  - User can use compiler supplied information.
- Useful for debugging compiler errors.
- Useful for prototyping.