Simulation/Evaluation Approach for a VLIW Processor

K. Ebcioglu, J. Moreno, M. Moudgill

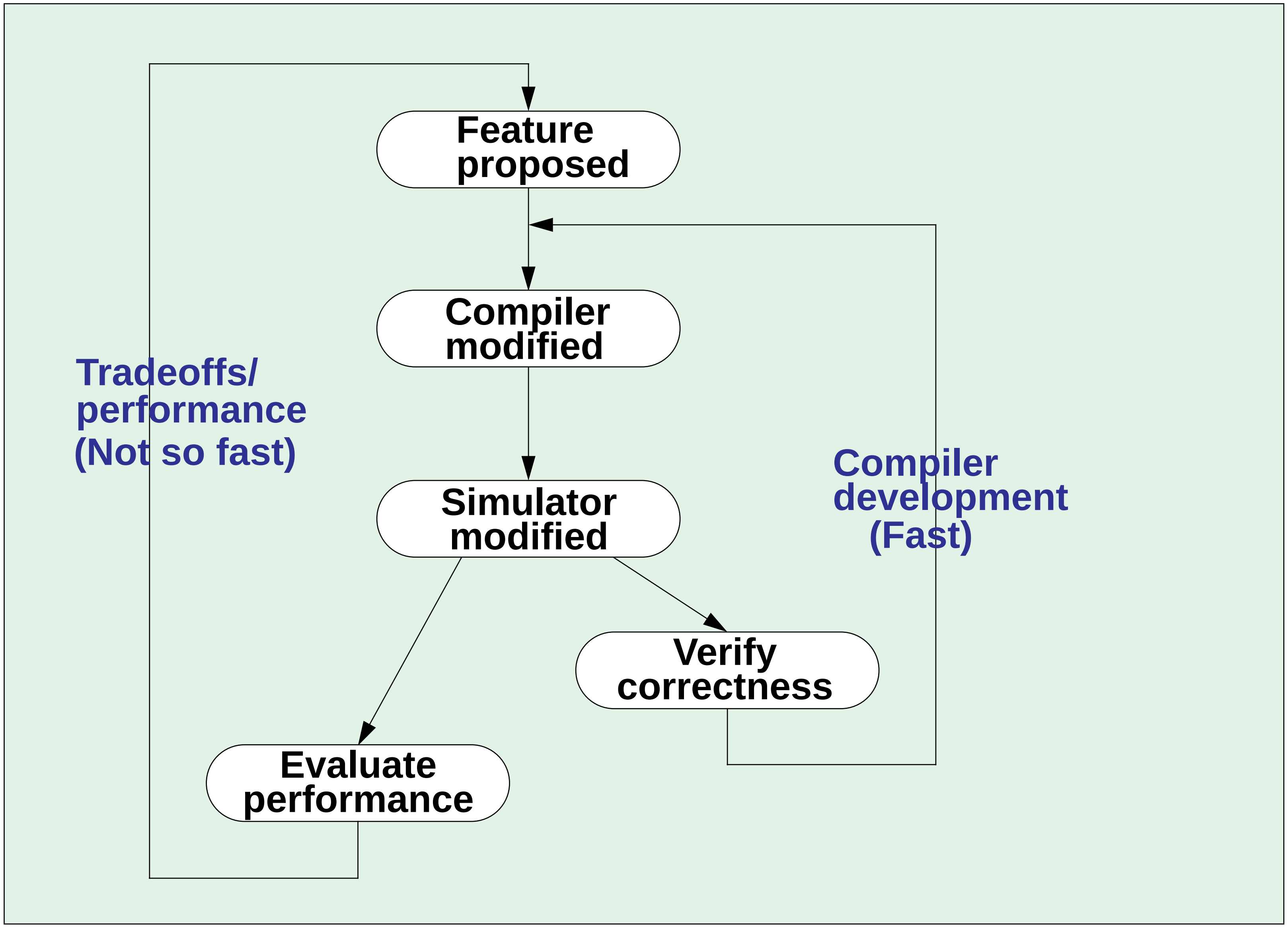
High-Performance VLSI Architectures

IBM T.J. Watson Research Center

Objectives of the environment

- Compiler for VLIW architecture
- Tool to perform tradeoffs
  - Compiler, architecture, and implementations
- Early evaluation of proposed architecture
  - Based on RS/6000 (PowerPC) architecture
- Rather conventional approach but innovative components

Iterative simulation/evaluation process



Important issues

Interaction among compiler and architecture

- Even more than for modern superscalar processors
- Simulation/evaluation environment *built around* such interaction

Predictability of a VLIW architecture

- Fully-scheduled code
  - Number of VLIWs executed vs. number of cycles (infinite cache)
- Compiler-speculated instructions
  - Fewer inaccuracies in count from cycle timer

Object-code compatible VLIW architecture

Allow for different implementations of same architecture

- **different number and type of functional units**
- **includes scalar, superscalar, and VLIW**

Basic aspect

- ***implementation-independent* representation of program in main memory**
  - **suitable for execution in any implementation**

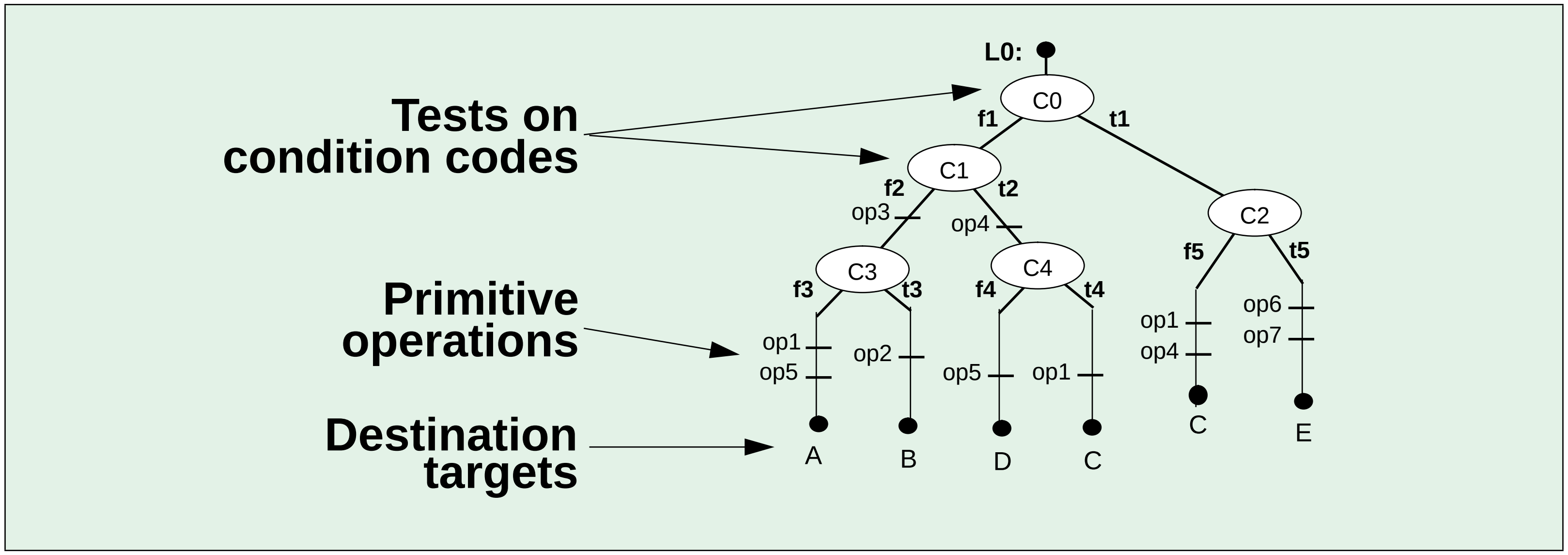
VLIW program model

Parallelized program represented as set of "tree-instructions" [Ebcioglu 88]

Tree-instructions contain

- **Multiway branch tree, with tests on condition codes**
- **Multiple primitive operations**
- **Multiple branch targets**

Subtree is also a tree-instruction



Semantics of a tree-instruction

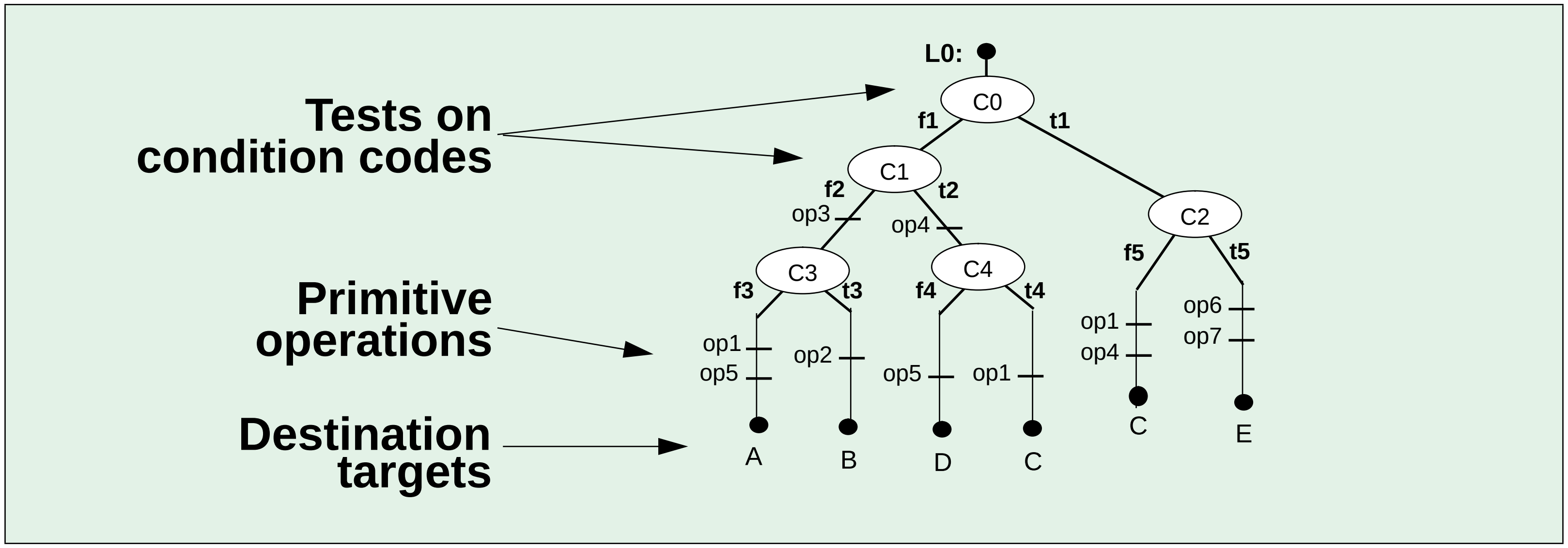
All operations are independent and executable concurrently

***Sequential* semantics for operations in each path of the tree**

All execution paths are active simultaneously

Only instructions in taken path complete execution

- **concurrent execution of instructions in non-taken paths, but no effects**



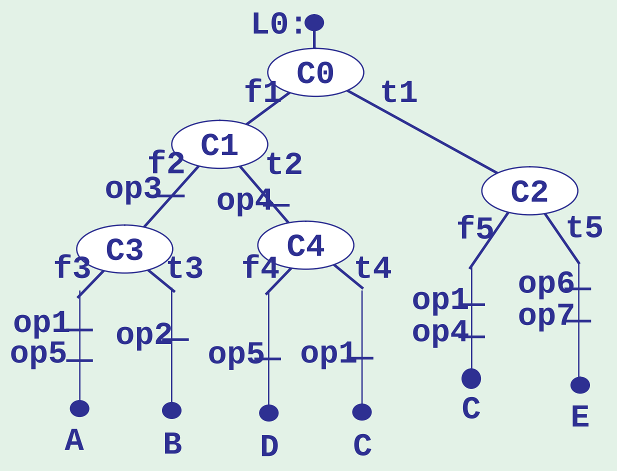
Representation in main memory

Sequential program from depth-first traversal of tree

New instruction: Conditional Skip

- control flow within a tree-instruction

Representation directly executable by scalar/superscalar processor



```
L0:  skip C0,t1          t4:  op1
f1:  skip C1,t2               branch C
f2:  op3                 t1:  skip C2,t5
     skip C3,t3          f5:  op1
f3:  op1                      op4
     op5                      branch C
     branch A            t5:  op6
t3:  op2                      op7
     branch B                 branch E
t2:  op4
     skip C4,t4
f4:  op5
     branch D
```

End of tree-instruction

Unconditional branch

Next primitive instruction unreachable from any skip

```
L0: skip C0,t1
f1: skip C1,t2
f2: op3
    skip C3,t3
f3: op1
    op5
    branch A
t3: op2
    branch B
t2: op4
    skip C4,t4
f4: op5
    branch D
```

```
t4: op1
    branch C
t1: skip C2,t5
f5: op1
    op4
    branch C
t5: op6
    op7
    branch E

L1: ....
```

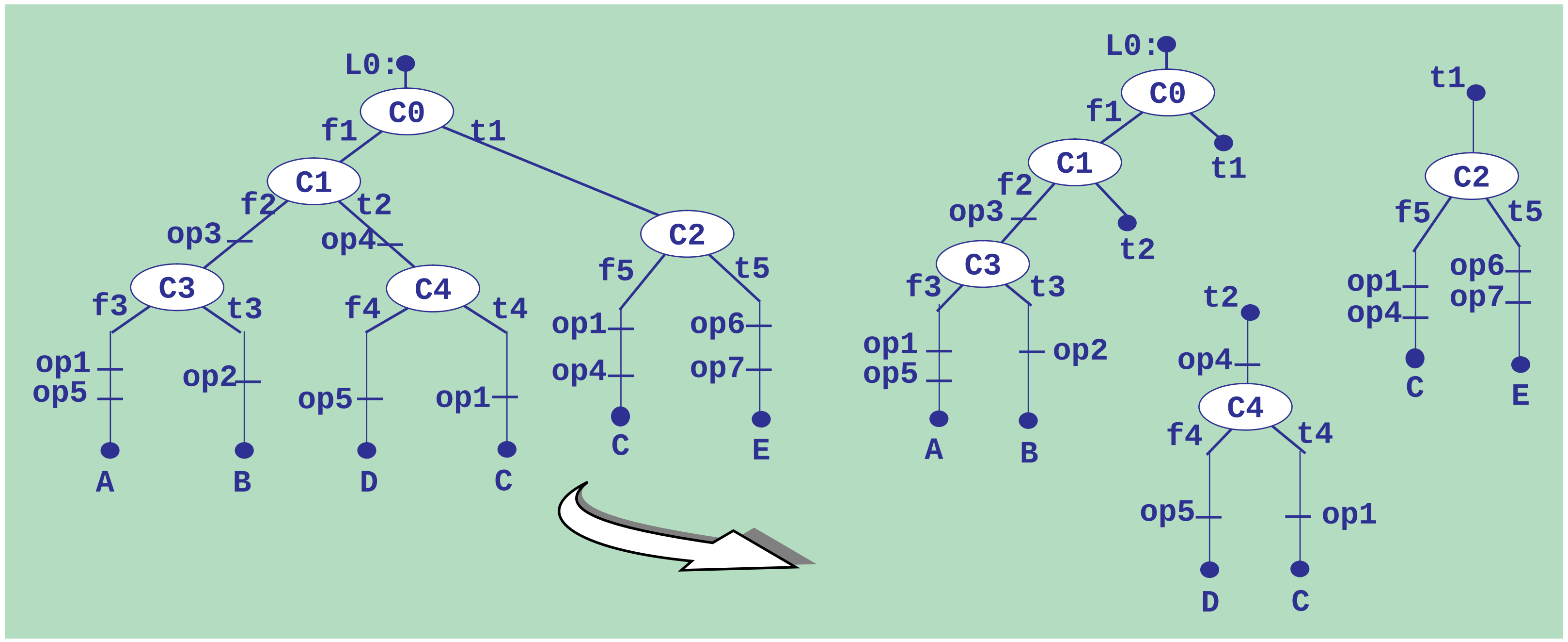
Execution of tree-instruction in processor with limited resources

Basic concept: “Pruning” the tree

- **semantics and number of operations remain unchanged**
- **implicit “unspeculation”**

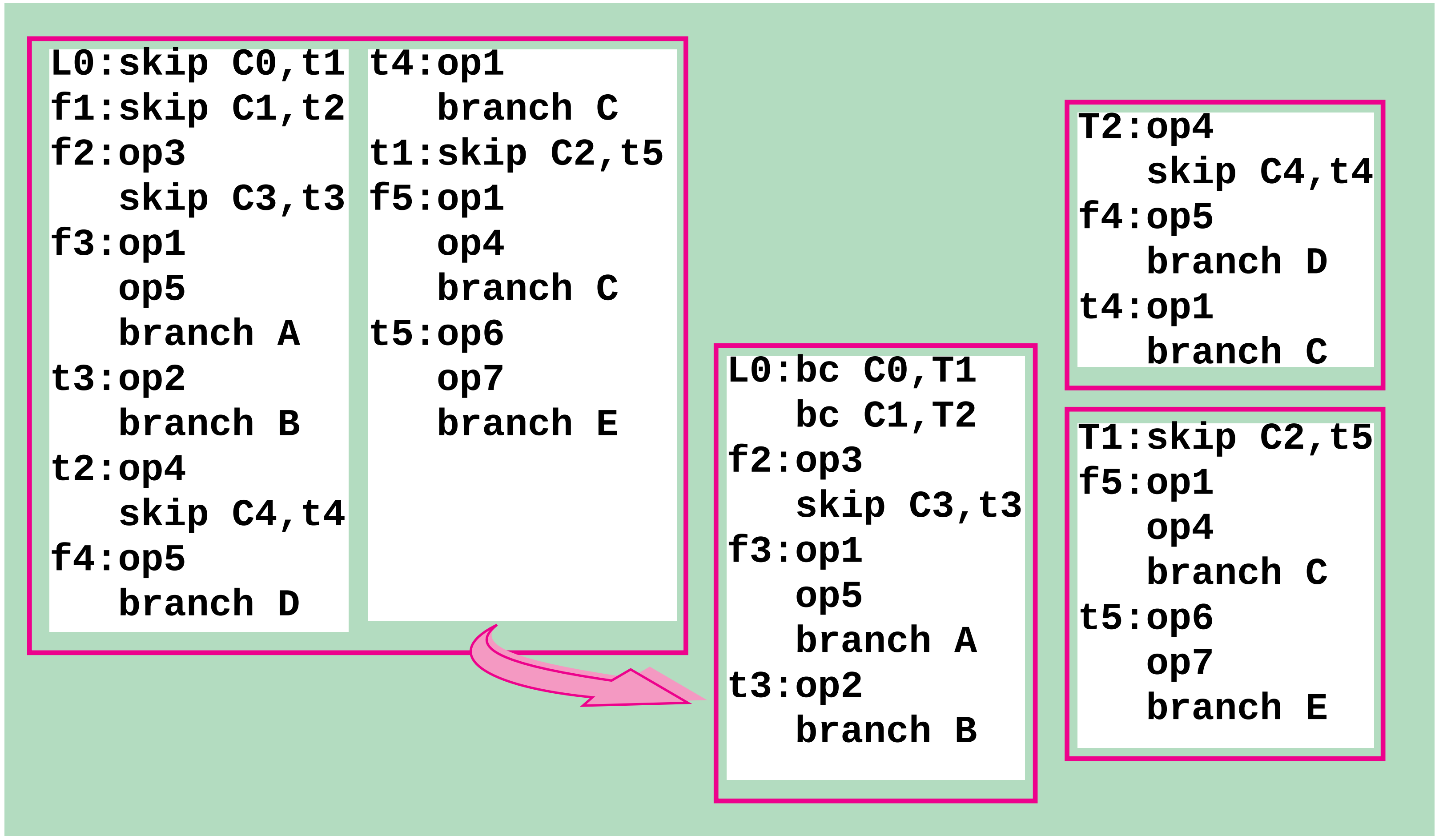
Pruning points determined by resources in the implementation

- **pruning performed by hardware**



Pruning example: insufficient branching capabilities

- Replace skip instructions by *conditional branch instructions*



```
L0:skip C0,t1
f1:skip C1,t2
f2:op3
   skip C3,t3
f3:op1
   op5
   branch A
t3:op2
   branch B
t2:op4
   skip C4,t4
f4:op5
   branch D
```

```
t4:op1
   branch C
t1:skip C2,t5
f5:op1
   op4
   branch C
t5:op6
   op7
   branch E
```

```
L0:bc C0,T1
   bc C1,T2
f2:op3
   skip C3,t3
f3:op1
   op5
   branch A
t3:op2
   branch B
```

```
T2:op4
   skip C4,t4
f4:op5
   branch D
t4:op1
   branch C
```

```
T1:skip C2,t5
f5:op1
   op4
   branch C
t5:op6
   op7
   branch E
```

Expected features of VLIW processor

RISC-like instruction set architecture

- RS/6000-based

Large register set (64, 128 registers)

10-20 operations per VLIW, 4-8 way branch

Support for compiler-speculated/out-of-order operations

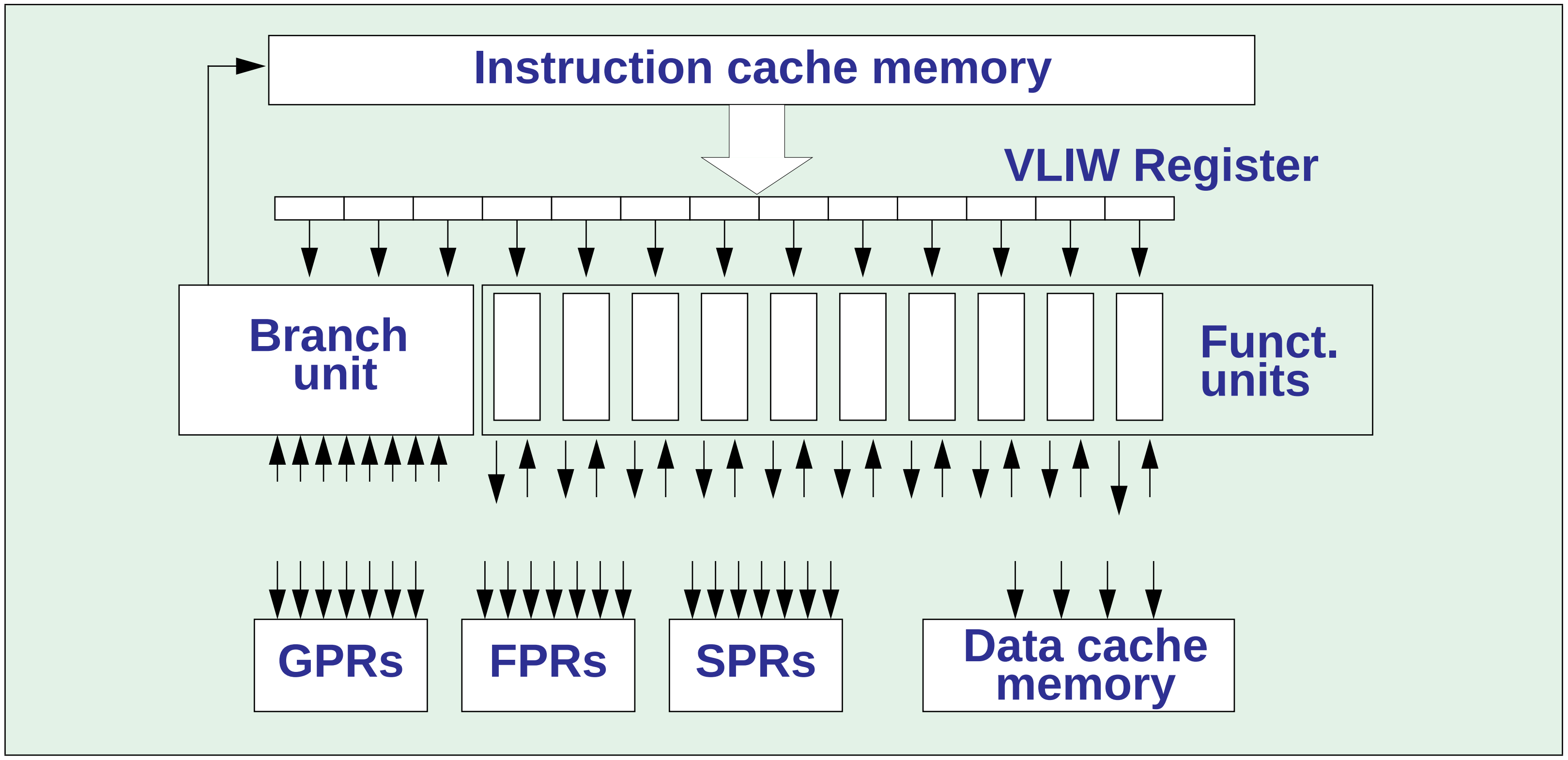
Tree-instructions

Fewer (if any) serializers (e.g., specialized registers)

High memory and register file bandwidth

Model of processor implementation

- **Multiple functional units**
- **Multiport register files**
- **Multiway branching in every cycle**
- **Short execution pipelines**



Evaluation compiler

Targeted (tailored) towards performing tradeoffs

Goals

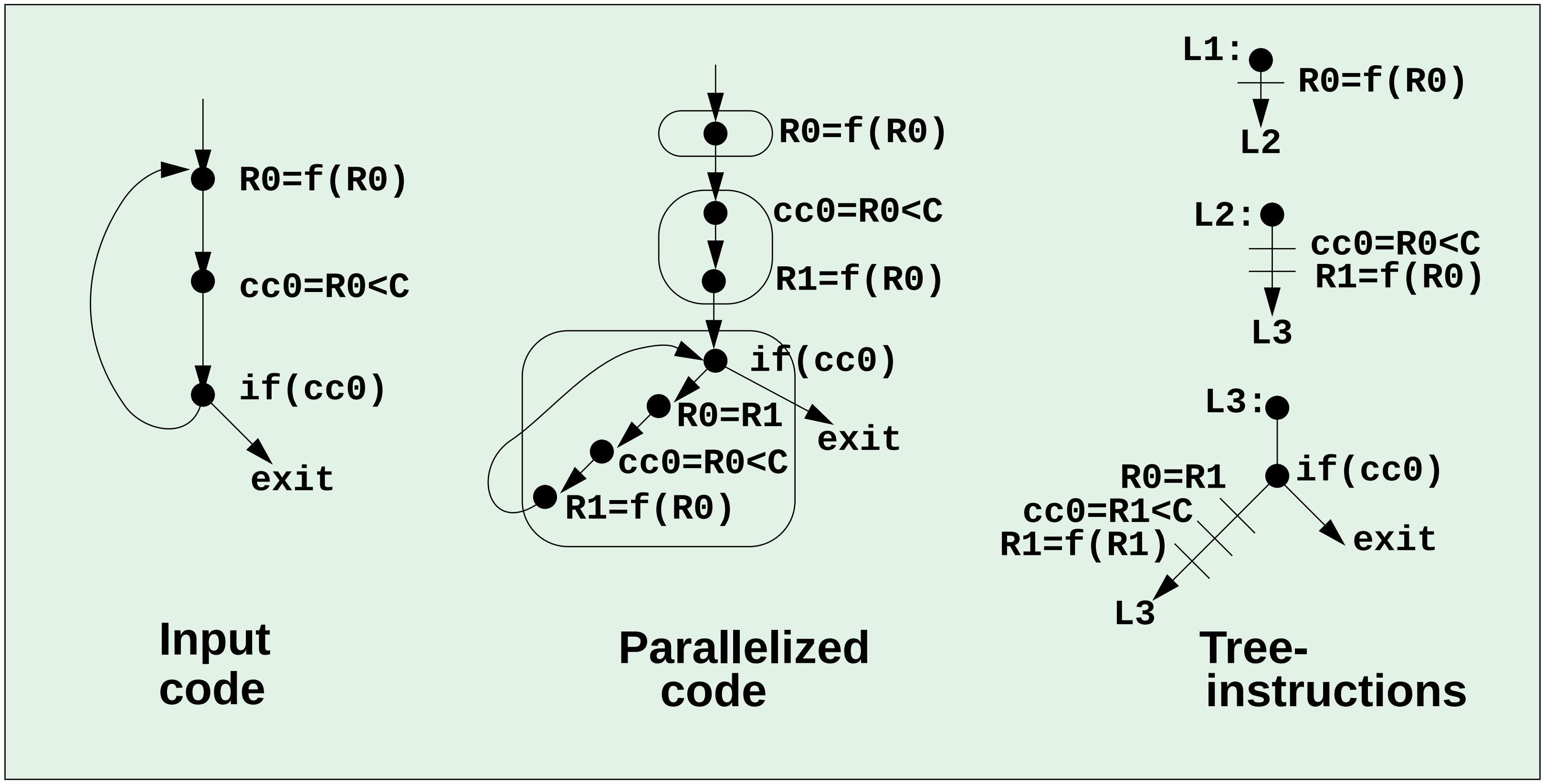
1. **Modifiability**
   - **ability to implement/test new algorithms/architectural features**
   - **emphasize programmers' productivity over space/ time efficiency**
2. **Robustness**
   - **verification/debugging as easy as possible**
   - **detect errors early in the process**
   - **internal self-tests**

Parallelization process [Moon 92]

Sequential code used as input

Determine parallelized code through compiler algorithms

Transform parallel code into tree-instructions



Basic compiler design

Based on dependence-flow graph [Pigali et al. 91, Johnson 94]

Integrated, consistent and persistent representations of

- **dependence flow information**
- **control flow information**
- **interval information**

Targeted to general instruction-level parallelism

- ✓ **architecture applied at end**
  - **parameterized, simple to change**
- ✓ **can also generate code for superscalar processor**
  - **currently, only VLIW back-end**

Compiler practices

Implicit use of traditional optimization techniques [Auslander 82]

Generalized software-pipelining, applied pervasively

- **inner-loops, outer-loops**
- **complex loop bodies, loops with multiple entries**
- **non-structured loops, functions**

Non-iterative register allocation

- **derivation of global graph-coloring [Chaitin 82]**

All algorithms are global: entire interval or function

O(n^2) algorithms acceptable, even for large regions

Extensive aliasing information, represented in easy-to-use form

- **more space**

Compiler practices

Cutting-edge technology

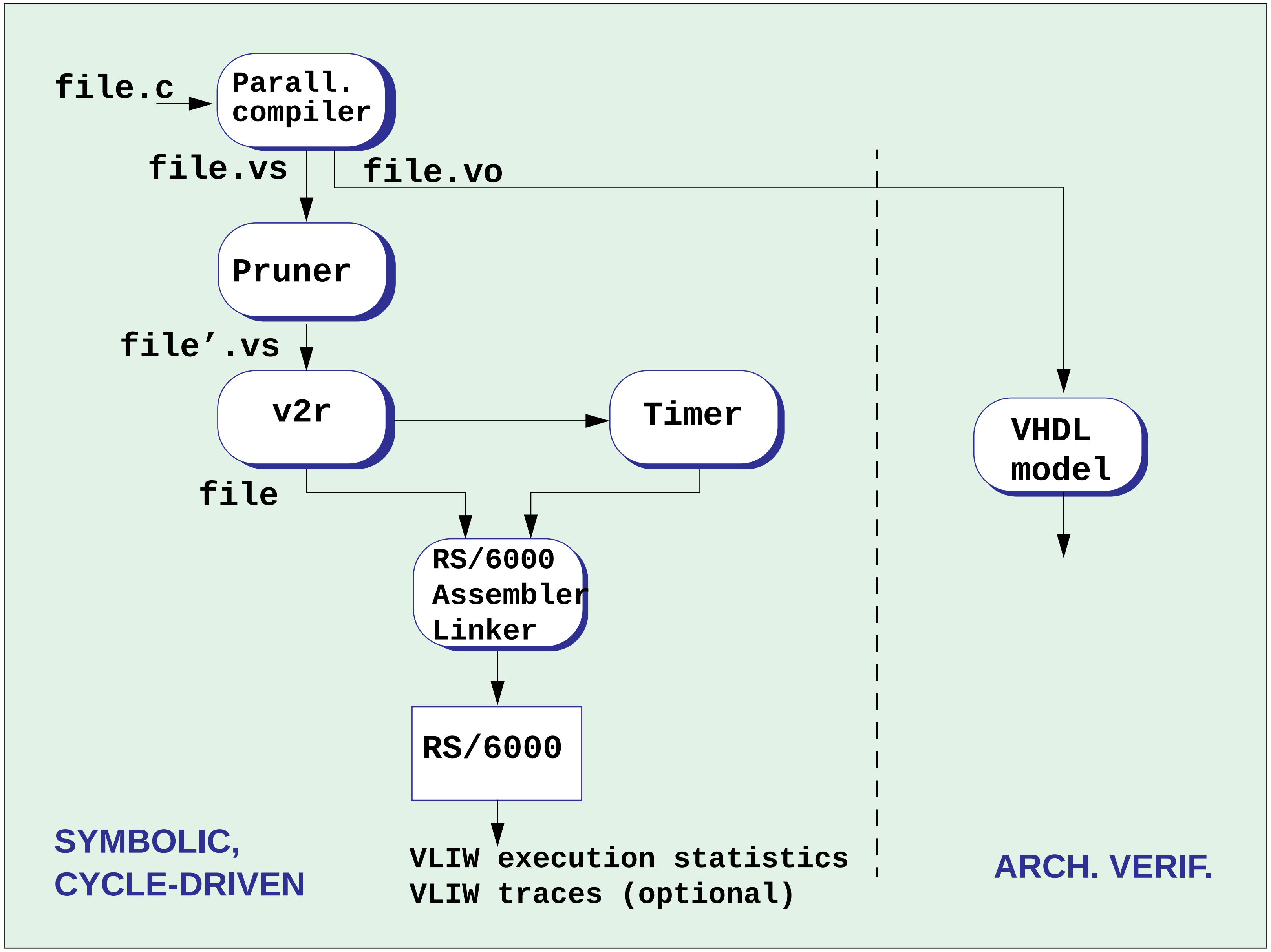
Good intermediate form

- **expensive, but considered “good investment”**

Extensive use of *assert/verify*

- **abort instead of run-time errors**

Simulation environment

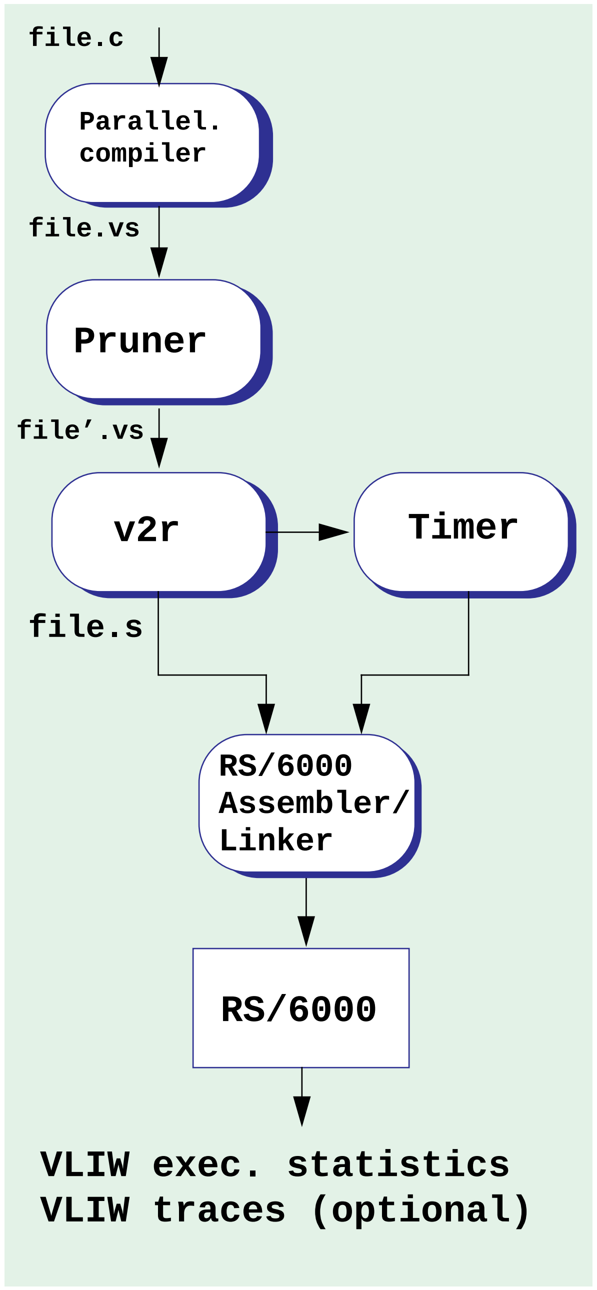


Symbolic, cycle-driven environment

Generates basic performance figures

- **VLIW execution and cycle count**
- **operations per VLIW executed**
- **number and type of stalls**
- **memory effects**
- **.....**

Can generate traces



Symbolic, cycle-driven simulator

Uses assembly output from compiler

Translates tree-instructions into sequential program

- **includes emulation of the different architectural features**

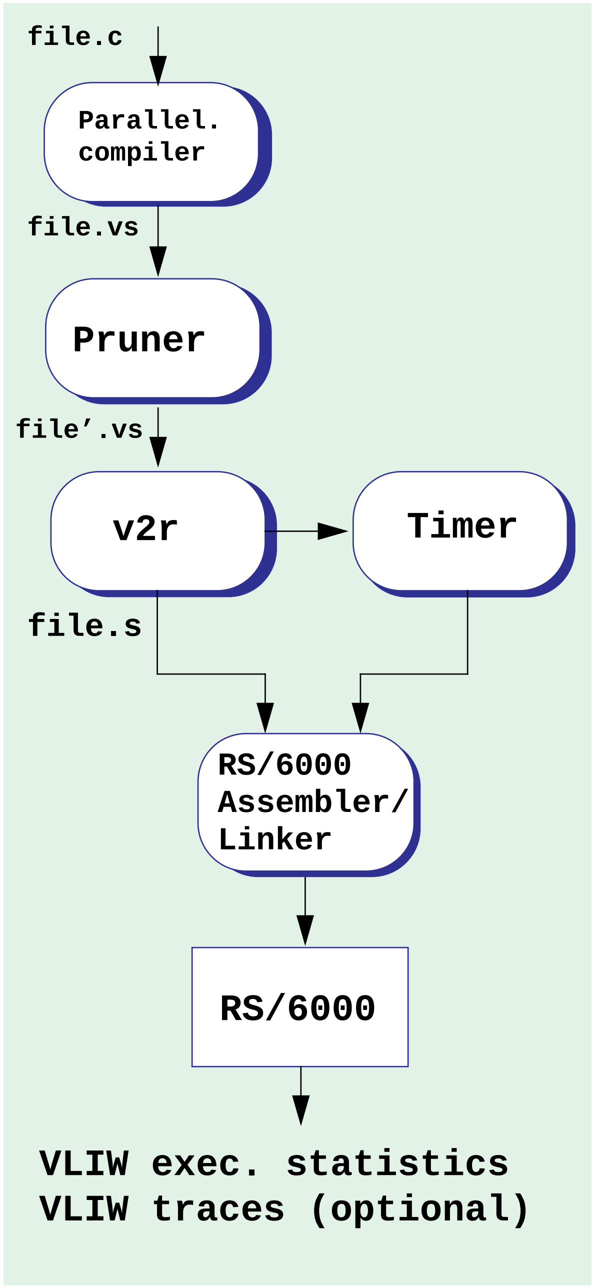
Instrumented

- **VLIW and cycle count, operations per VLIWs executed, number and type of stalls, cache misses, ...**

Mixing of parallelized and non-parallelized modules

Fast execution

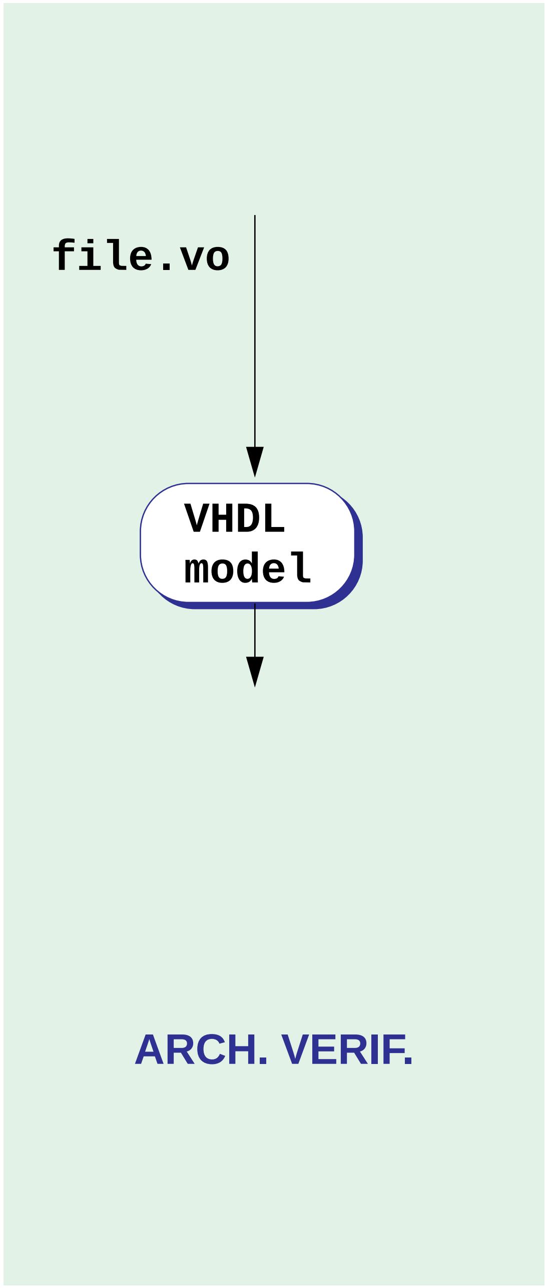
- **5-1000 times slower than native version of same program, depending on instrumentation**



VHDL mode

Bit-level behavioral description of an implementation

Behavioral verification of architecture specification



Examples of trade-offs among compiler, architecture and implementation

Somewhat simplifying ISA

- **decomposition into sequences of simpler instructions**
  - **some complex RS/6000 instructions**
  - **index addressing mode**

Somewhat complicating ISA

- **inclusion of some frequent combined operations: add-shift, ...**

Tighter constraints in instruction encoding

- **larger register fields (more registers)**
- **larger condition register field (remove serialization due to CR0)**

Varying number of resources

Concluding remarks

Suitable environment for early verification/simulation of compiler and architecture

- **whole environment designed for mutability**
  - ✓ **sacrifice performance sometimes**
  - ✓ **extensive use of table-driven techniques**

Reasonable fast turn-around time at symbolic level

- **from compiler output to simulation results**
- **allows testing the compiler**

Good tool for intended purposes

Promising quantitative results