Compiler/architecture interaction in a tree-based VLIW processor

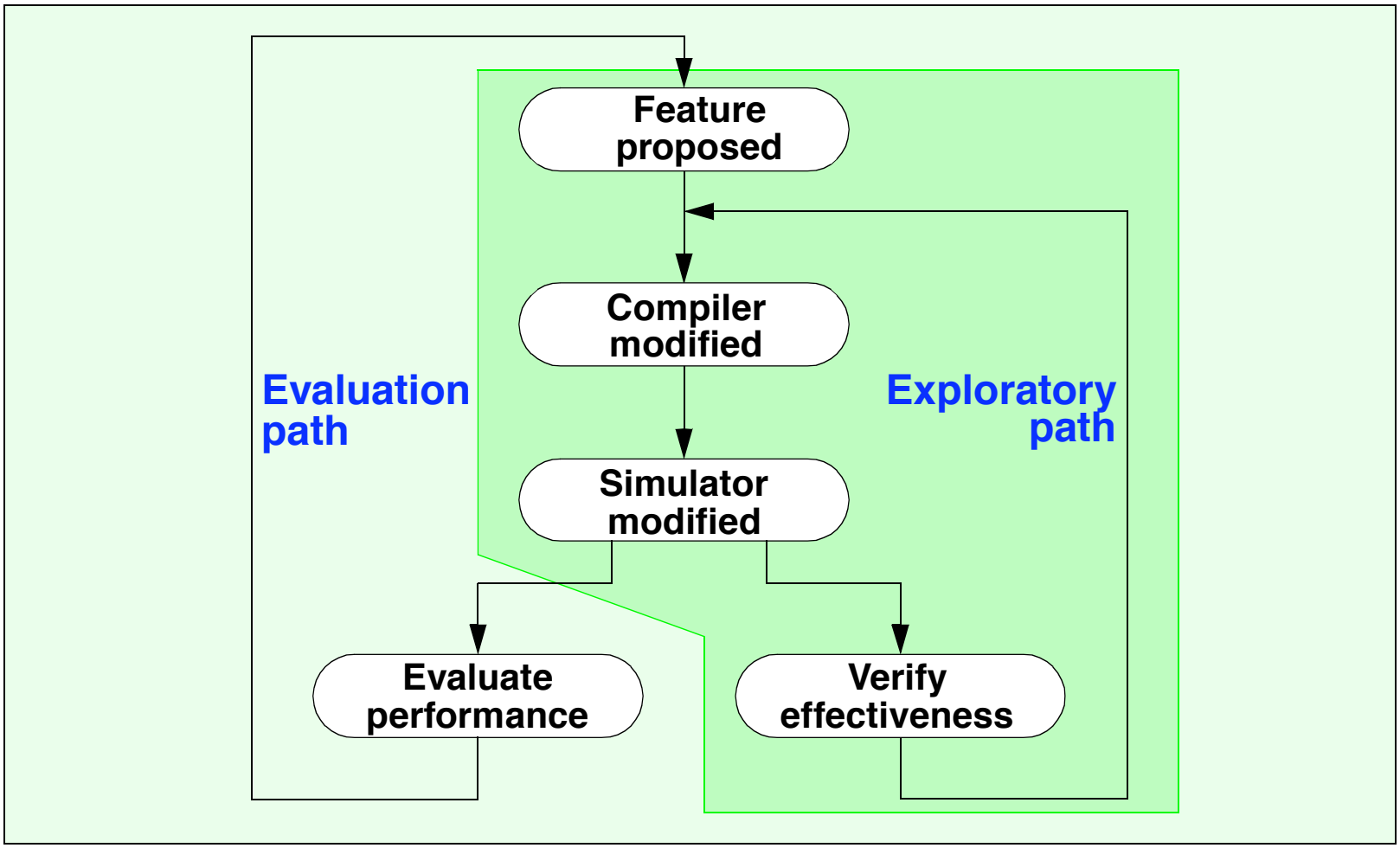
M. Moudgill, J. Moreno, K. Ebcioglu,
E. Altman, S.K. Chen, A. Polyak
IBM T.J. Watson Research Center

Scenes from ...

"If we had a instruction KK, then we can reduce the execution time of workload XYZ by n%"	Are you crazy? How are you going to implement that instruction? What about the benefits in other workloads?
"We were having difficulties implementing instruction JJ, so we have decided to drop it"	Are you crazy? Do you know how much performance will be lost?
"We have noticed that instruction LL is not present in the execution traces"	What did you expect? It was too difficult to incorporate it in the compiler, so we never did.
"We must have a larger set of registers"	How much does it buy you? How are you going to encode them?

Iterative simulation/evaluation process

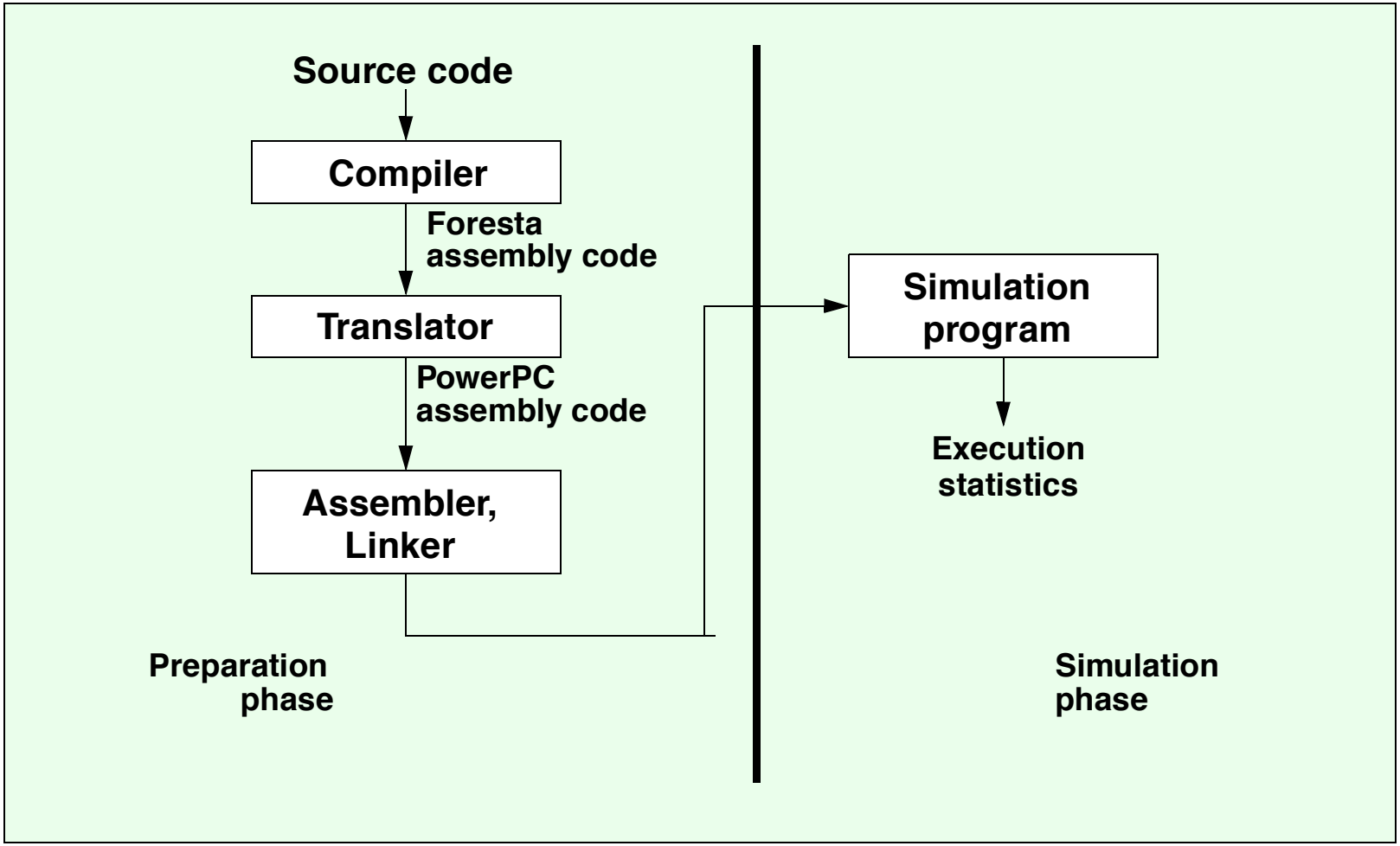
Built *around* the architecture/compiler interaction



Overview of environment

Tool to perform trade-offs in many dimensions

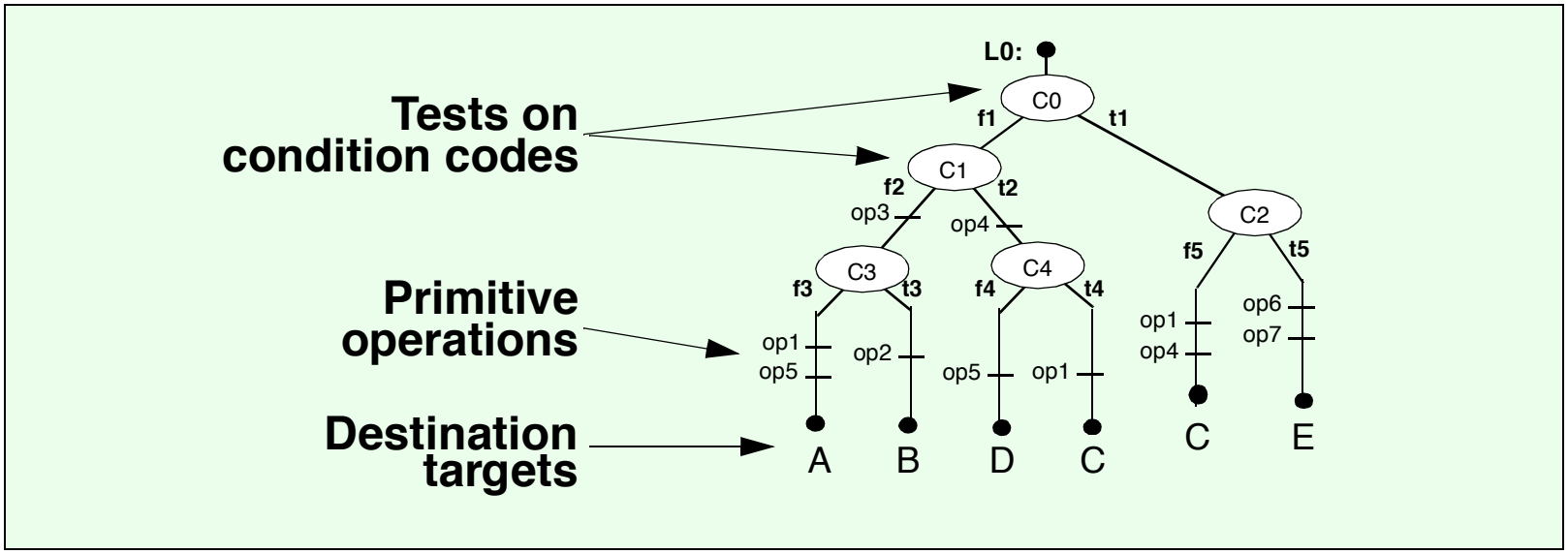
Rather conventional approach but innovative components



FORESTA, a VLIW architecture based on tree-instructions

Program representation based on "tree-instructions" [Ebcioglu 88]

- multiway branch tree
- primitive operations in tree-paths
- all operations and multiway branch executable concurrently
- sequential semantics in each path of the tree
- only instructions in taken path complete execution



Primitive operations

Based on PowerPC architecture

Deviations include

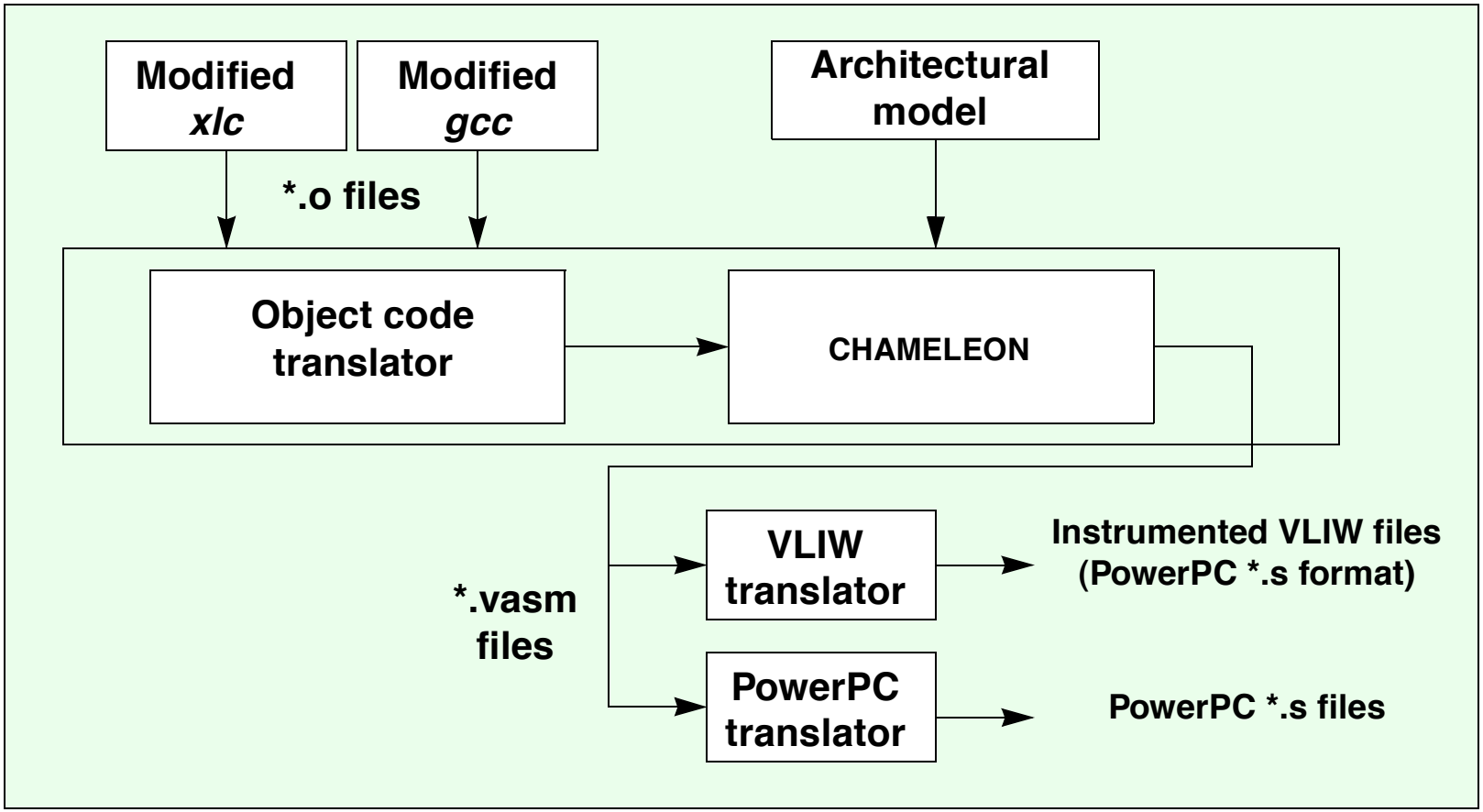
- larger register set
- support for speculative, non-trapping load operations
- some complex operations deleted
- “record” form of instructions can specify any Condition Register
- shorter displacement field in memory operations
- some 64-bit encoding
- support for 32-bit immediate values
- some three-input operations added: add&shift, and&or, ...
- some support for conditional execution

The CHAMELEON compiler

Designed to support ILP research and evaluate trade-offs

Extensively parameterized

Designed to support mutability

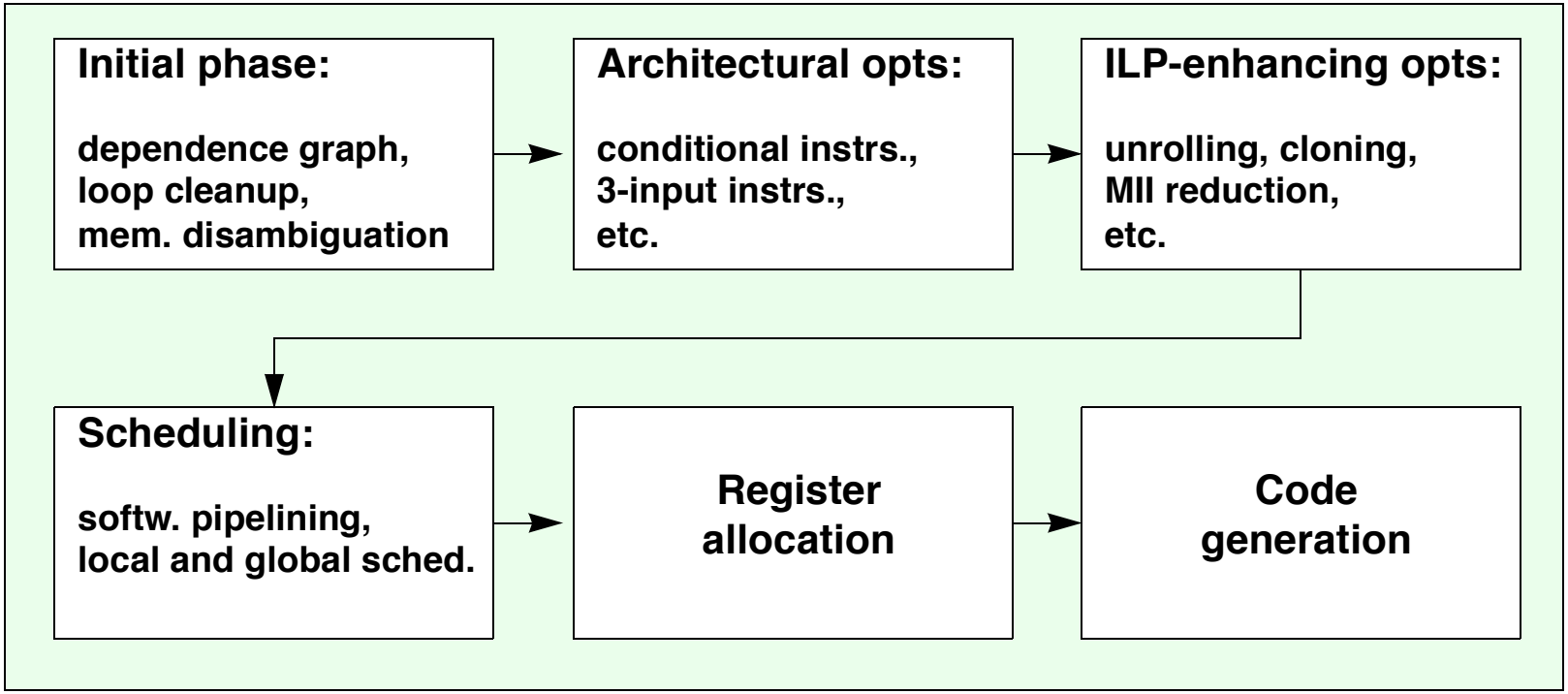


Optimizations in CHAMELEON

Traditional (applied throughout the compilation process)

ILP-increasing

Architectural



Variant of Ball-Larus heuristics (no profile-directed feedback)

No interprocedural analysis

Implementation of CHAMELEON

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

- integrated data/control flow information

Table-driven, extensively parameterized

- adding a new primitive: add one entry into a table

Stand-alone transformations

- transformations on the DFG

Algorithms are global

Extensive debugging support

Compilation time

- conscious decision

Compiler/architecture interactions

Wide range can be explored

number and type of operations per VLIW

size of register set

latency of operations

availability/unavailability of specific instructions

3-input instructions

conditional move and conditional store instructions

“record” form of instructions

length of displacement and immediate fields

static reordering of ambiguous memory references

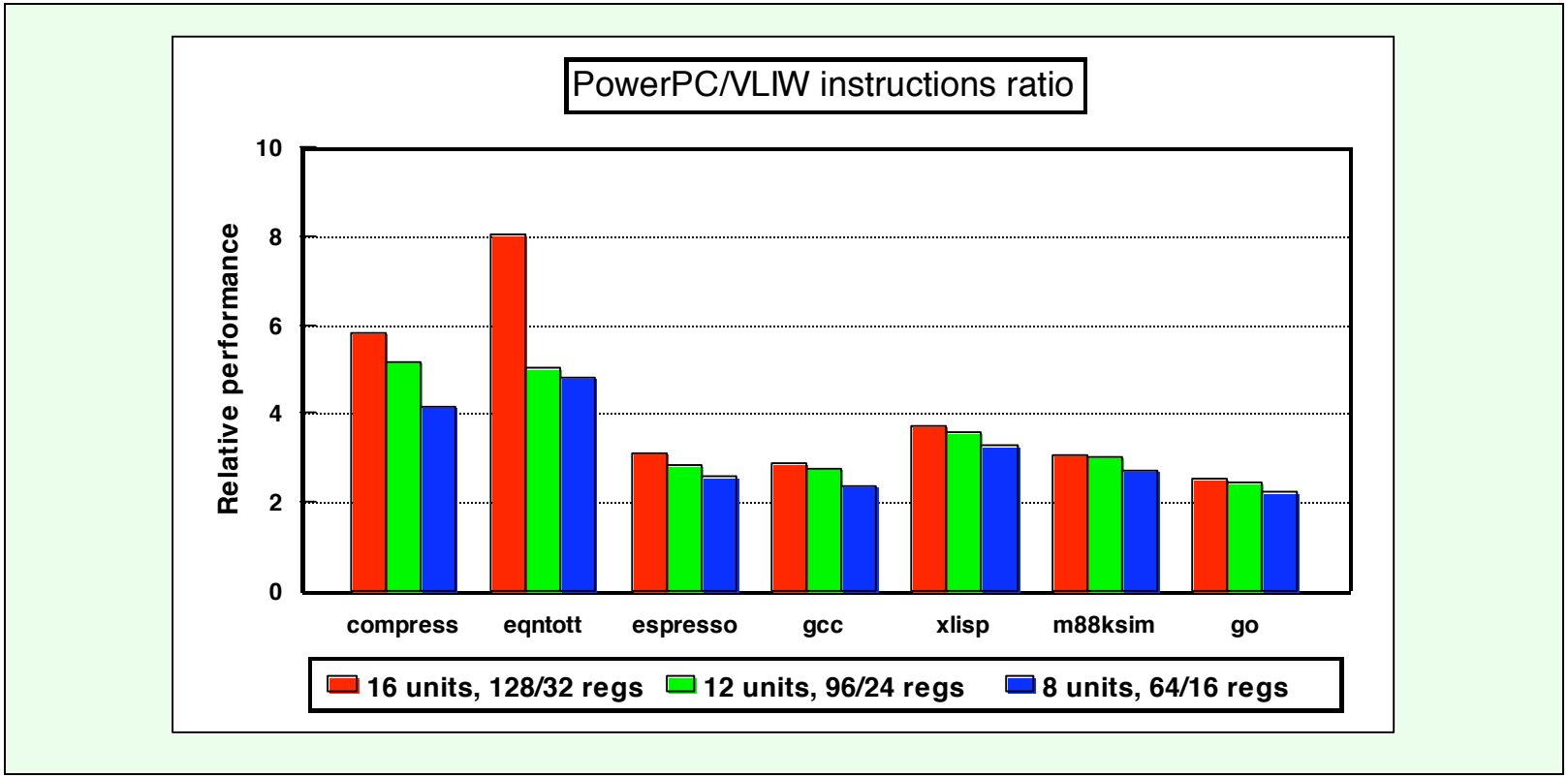
cache prefetch instructions

.....

Examples of compiler/architecture interactions

Instruction-level parallelism, larger register set

Compiler invoked with parameters file describing target architecture



ILP computed with respect to sequential code optimized with *xlc*

Three-input operations

Three classes of instructions

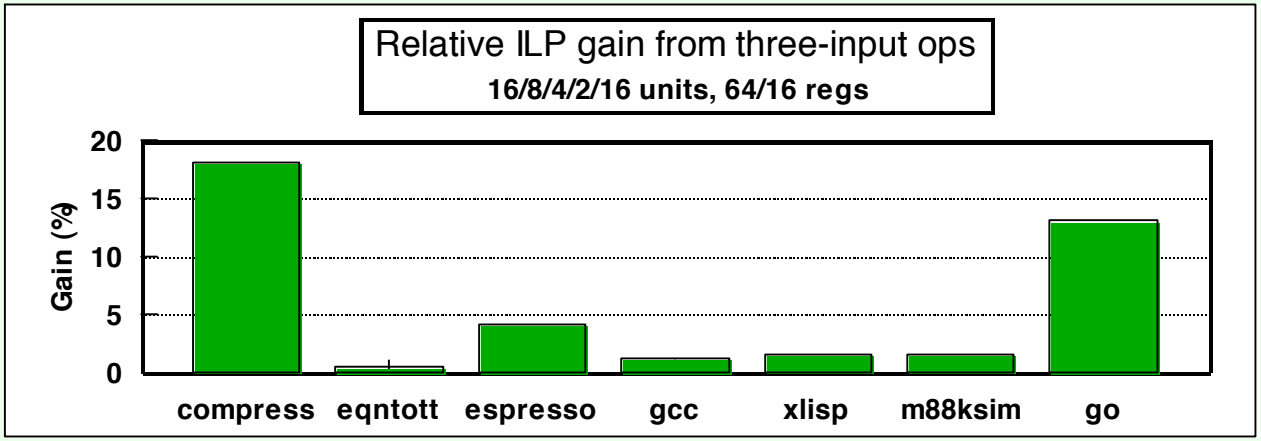
A: any combination of add/subtract with add/subtract
S: any combination of add/subtract with shift, or vice-versa
L: any combination of logical with logical

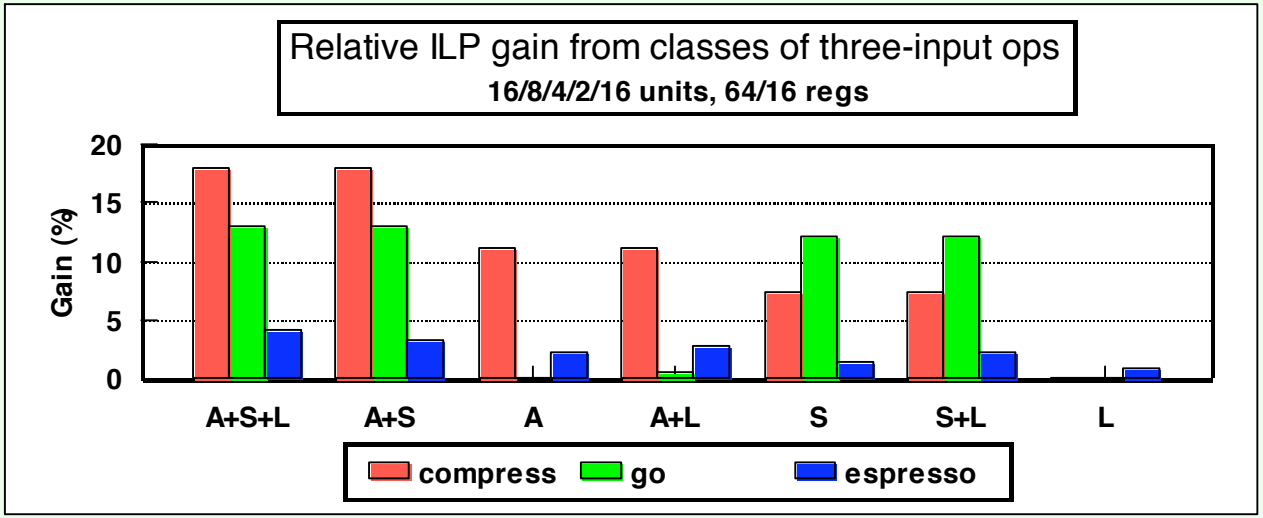
New instructions added to opcode table

Optimizations added and modifications

- **phase combining two operations from same basic block into single one**
- **scheduling heuristics**
- **final peephole compaction**

Three-input operations





Concluding remarks

Environment for studying compiler/architecture interactions

- large variety of interactions possible
- many dimensions

Designed for mutability

State-of-the-art compiler

Fast simulation/evaluation environment

- 10 to 15 times slower than optimized native program

Further work

Adaptation to dynamically scheduled processors