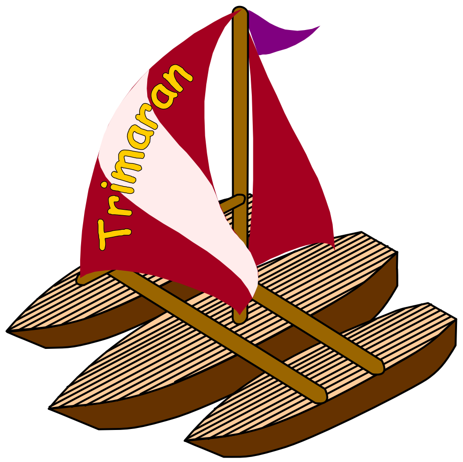
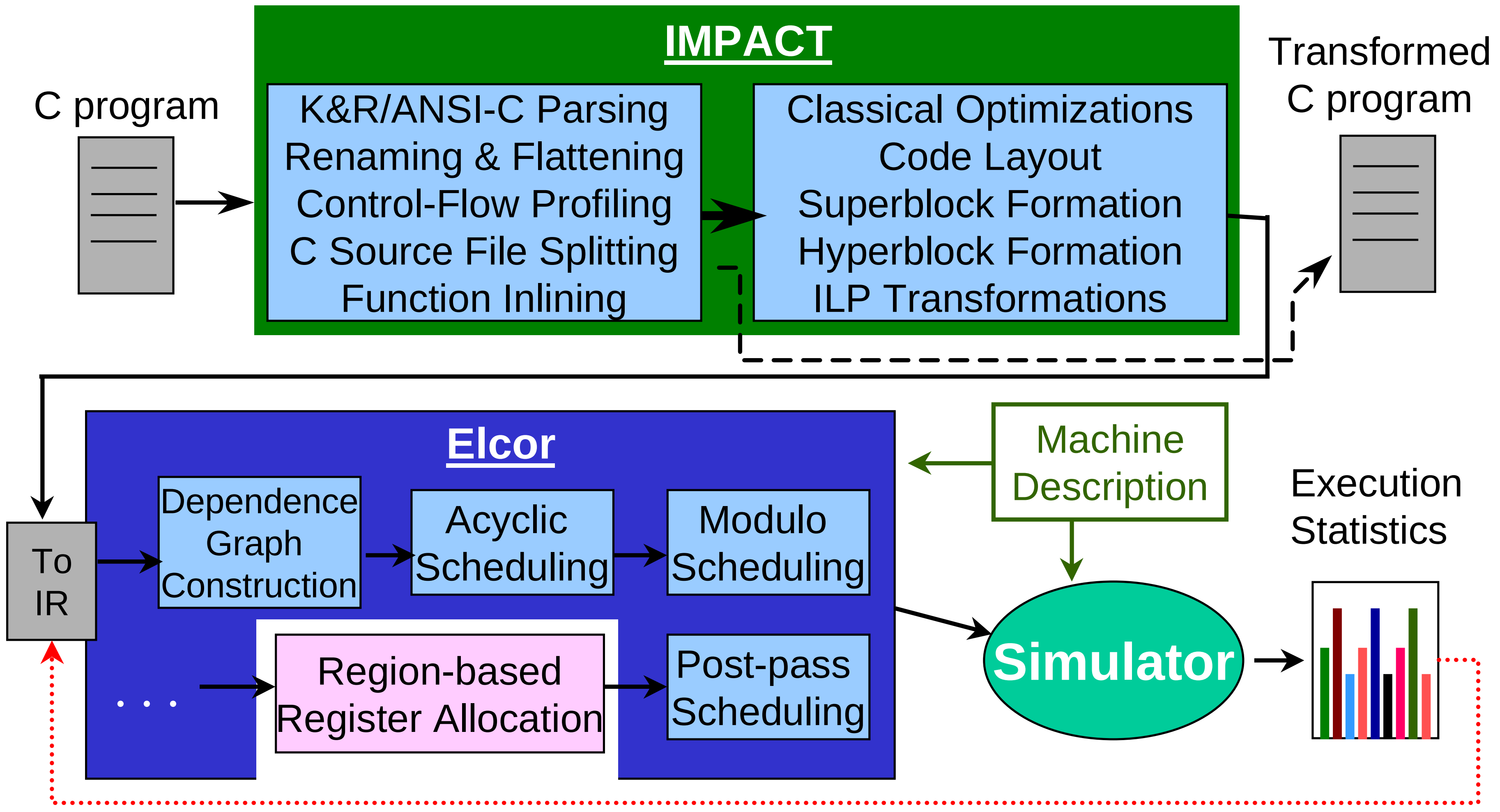


An Overview of the IMPACT Module and Its Optimization Suite

System Organization

A compiler researcher's view of the infrastructure:



New Benchmark Framework

- “bench_info” framework features
  - how to compile and run the benchmark
  - copy of the benchmark’s source (optional)
  - robust, complete, and user-friendly set of tools
    - test_bench_info script validates info provided
  - plug-in-play support
    - user configurable search paths for bench info
    - no script modifications needed to add benchmark
  - multiple input support
  - tutorial with walk-through of usage
    - …/impact/tutorials/bench_info_tutorial

K&R/ANSI-C Parser

- Built upon EDG C parser
  - Solid but persnickety about C language spec
    - May need to modify benchmark source to match spec
  - Utilizes native compiler's header files (in most cases), and libraries
  - We may only distribute binaries and source diffs
    - Unmodified source available via free educational license from EDG (see web site for source diffs and instructions)
    - Modified to generate our source-level intermediate rep.
- Compile **all** the available source **together**
  - Don't link in libraries if have source for libraries!
  - Profiler and source analysis tools need everything

Renaming and Flattening

- Renaming affects all global static variable and function names
  - Changes to allow global non-static scope
- Flattening transforms all complex expressions into simple expressions
  - Adds temporary variables when necessary
- Operates at the C source-file level

Control-Flow Profiling

- Straightforward control-arc profiler
  - Generates execution and branch weights
  - 2-3 times slower than uninstrumented executable
- Reverse generates instrumented C code
  - May also use rest of IMPACT/Elcor path instead
- Currently annotates in only one run's data
  - Multiple-input support will be released soon
- Required step when using IMPACT module!
  - 1.5X to 4.1X faster code with profile info

C Source File Splitting

- Breaks source into one function per file
  - Preparation step for function inlining
  - Renames structure/union tags when necessary
  - Moves all global variables into one file data.pcs
  - Moves all structure definitions into struct.pch
  - Generates every function prototype into extern.pch
    - All functions are now explicitly prototyped
- Operates on the entire program

Function Inlining

- Profile-based global function inlining
  - Currently inlines most important call sites until reach 20% static code growth (configurable)
  - Currently does not inline calls via function pointers
- Significantly improves benchmark performance
  - Expands optimization and scheduling scope
  - 1.5X to 2.2X faster code with inlining

Reverse Generation of C

- Typically only used by control-flow profiler
  - May reverse generate C after any point in frontend processing (including after inlining)
  - Profiler generates instrumented C code
    - Builds profiling executable with native compiler

Classical Optimizations

- Based on “Red Dragon Book” optimizations
  - Applied both at basic-block and function level
  - Applied iteratively to maximize performance
- Will generate non-trapping operations
  - Typically invariant code removal causes generation
  - Controlled by Lglobal param. non_excepting_ops
- Currently utilizes “unsafe flags” for memory disambiguation
  - For example: different data types independent
  - Some benchmark-specific tuning required (e.g. go)
  - Good, but source-level analysis better (Fall’ 99)

Code Layout

- Intra-function layout based on profile info
  - Arrange code so branches usually fall-thru
  - Most-likely trace placed at beginning of function
  - Then second most-likely trace, and so on
  - Unexercised code placed at end of function
- Potential benefits
  - Reduces jumps in frequently executed code
  - Executed code usually fits better in Icache
  - May result in fewer branch entries in BTB

Superblock Formation

- Creates single-entry/multiple-exit blocks
  - Based on profile information
  - Increases scheduling and optimization scope
  - Tail-duplication utilized to avoid bookkeeping code required by trace scheduling techniques
    - Also can expose more optimization opportunities
- After superblock formation, profile information will no longer be completely accurate

Hyperblock Formation

- Creates single-entry/multiple-exit blocks from multiple paths using predication
  - Increases scheduling/optimization opportunities
  - Converts control-flow into data-flow optimizations
  - Can remove branch mispredictions, but not the only benefit
  - Superblocks still used when appropriate
- Predicted code first-class citizen in IMPACT
  - Dataflow, predicate analysis, classical optimizations, ILP optimizations, etc.

ILP Transformations

- Enhance and expose ILP
  - Loop unrolling, register renaming, renaming with copy, induction and accumulator variable expansion, operation migration, predicate promotion, predicate-based branch combining, …
- Usually increases dynamic # of operations
  - Raw IPC can be misleading, use speedup, etc.
- Accuracy of profile information further reduced
- Can significantly improve performance
  - 1.2X to 3.2X faster code with superblock/ hyperblock formation and ILP transformations