Open64/ORC compilers

Sébastian Pop

Université Louis Pasteur Strasbourg,

Project A3 INRIA

FRANCE

Short history

- 1994: Ragnarok compiler for MIPS R8000

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications
- August 1994: start Mongoose compiler

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications
- August 1994: start Mongoose compiler
  - scientific and non-scientific applications

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications
- August 1994: start Mongoose compiler
  - scientific and non-scientific applications
  - fast and stable for day-to-day development

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications
- August 1994: start Mongoose compiler
  - scientific and non-scientific applications
  - fast and stable for day-to-day development
- 1999: focus on IA-64:
  SGIpro 1.0 (alias osprey1.0)

Short history

- 1994: Ragnarok compiler for MIPS R8000
  - designed for scientific applications
- August 1994: start Mongoose compiler
  - scientific and non-scientific applications
  - fast and stable for day-to-day development
- 1999: focus on IA-64:
  SGIpro 1.0 (alias osprey1.0)
- 2001: Intel and ICT Chinese Academy of Sc.
  ORC (Open Research Compiler)

Compiler's Structure

1. FE (Front-ends)
2. WHIRL (Intermediate Representation)
3. IPA (Inter Procedural Analysis)
4. LNO (Loop Nest Optimizer)
5. WOPT (Global Optimizer)
6. CG (Code Generator)
7. ORC (Open Research Compiler)

Front Ends

- Use GCC's C/C++ and Cray F90 front ends

Front Ends

- Use GCC's C/C++ and Cray F90 front ends
- Each front end has its own specific trees

Front Ends

- Use GCC's C/C++ and Cray F90 front ends
- Each front end has its own specific trees
- Translation to WHIRL

Front Ends

- Use GCC's C/C++ and Cray F90 front ends
- Each front end has its own specific trees
- Translation to WHIRL
- Question: Is this translation valid?

Front Ends

- Use GCC's C/C++ and Cray F90 front ends
- Each front end has its own specific trees
- Translation to WHIRL
- Question: Is this translation valid?
  - Test suites were not GPL-ed,
    could use GCC test suites (inappropriate)

Front Ends

- Use GCC's C/C++ and Cray F90 front ends
- Each front end has its own specific trees
- Translation to WHIRL
- Question: Is this translation valid?
  - Test suites were not GPL-ed,
    could use GCC test suites (inappropriate)
  - Bug data base wasn't GPL-ed.

WHIRL

Winning Hierarchical Intermediate Representation Language

WHIRL

Winning Hierarchical Intermediate Representation Language

- 5 levels: VH, H, M, L, VL
- Lowering happens when needed
- Each optimization performed at the right level

WHIRL

Winning Hierarchical Intermediate Representation Language

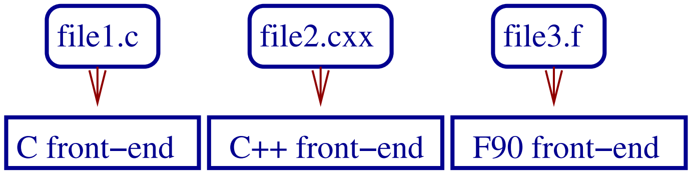
- *whirl2c* and *whirl2f* dump WHIRL in compilable files.
- *whirl2a* dump WHIRL in ASCII.

Inter Procedural Analysis

file1.c file2.cxx file3.f

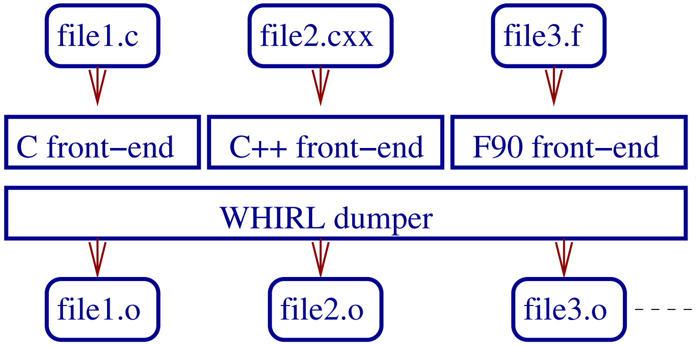
Suppose that we want to build a project containing 3 files and use the IPA for optimizing it.

Inter Procedural Analysis



The first step invokes the right front–end.

Inter Procedural Analysis

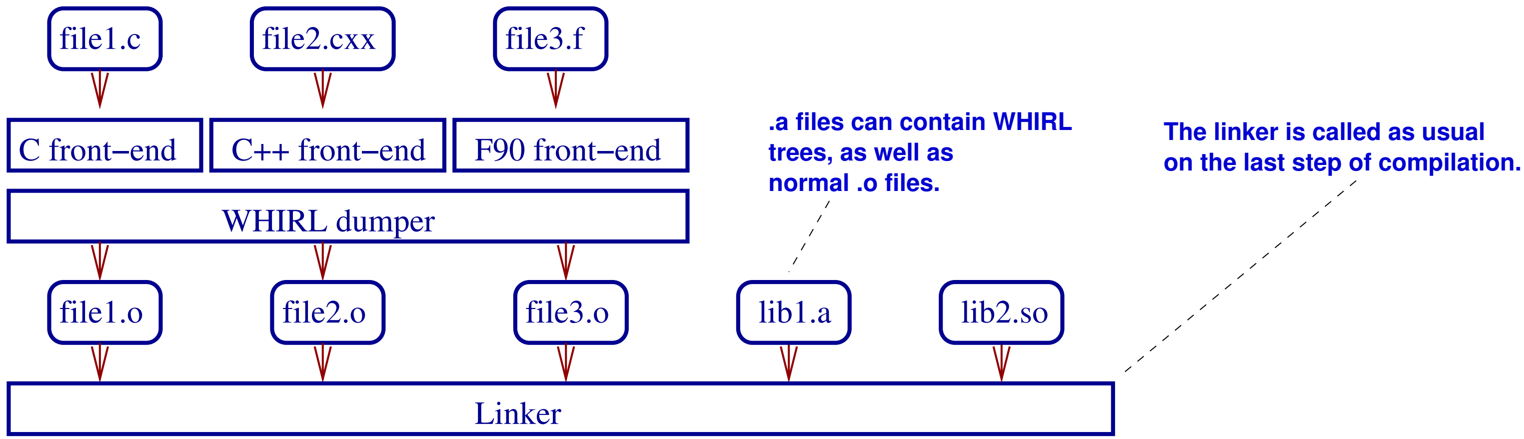


The compiler transforms front–end specific trees into WHIRL trees.

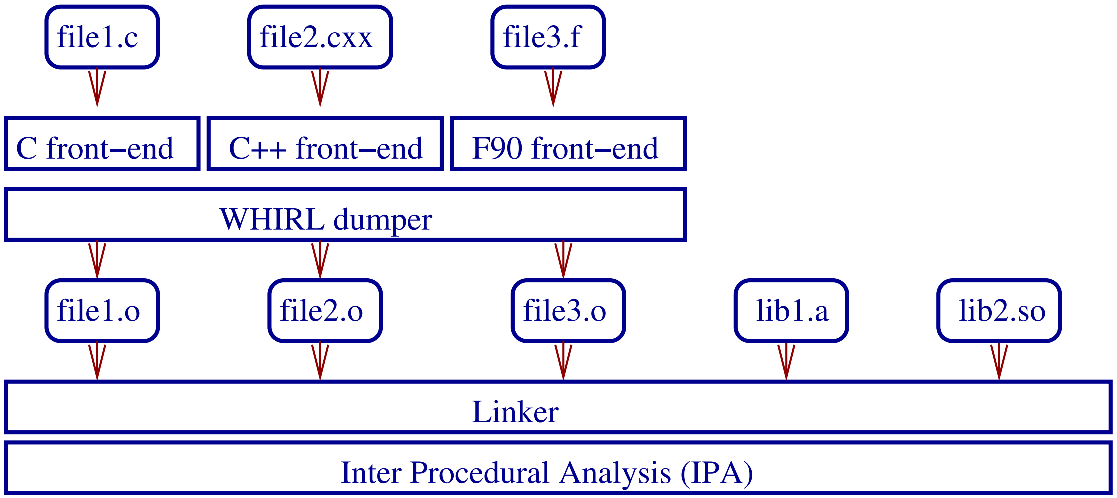
This representation is then dumped into a .o file.

These .o files behave like normal relocatable code (I.e. can be put in archives, etc.)

Inter Procedural Analysis

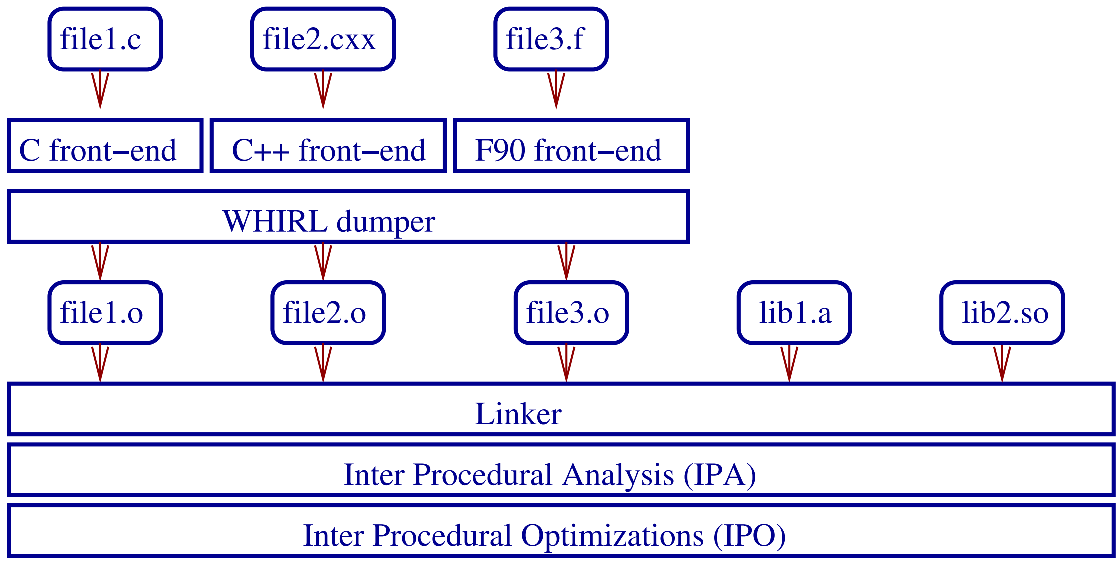


Inter Procedural Analysis

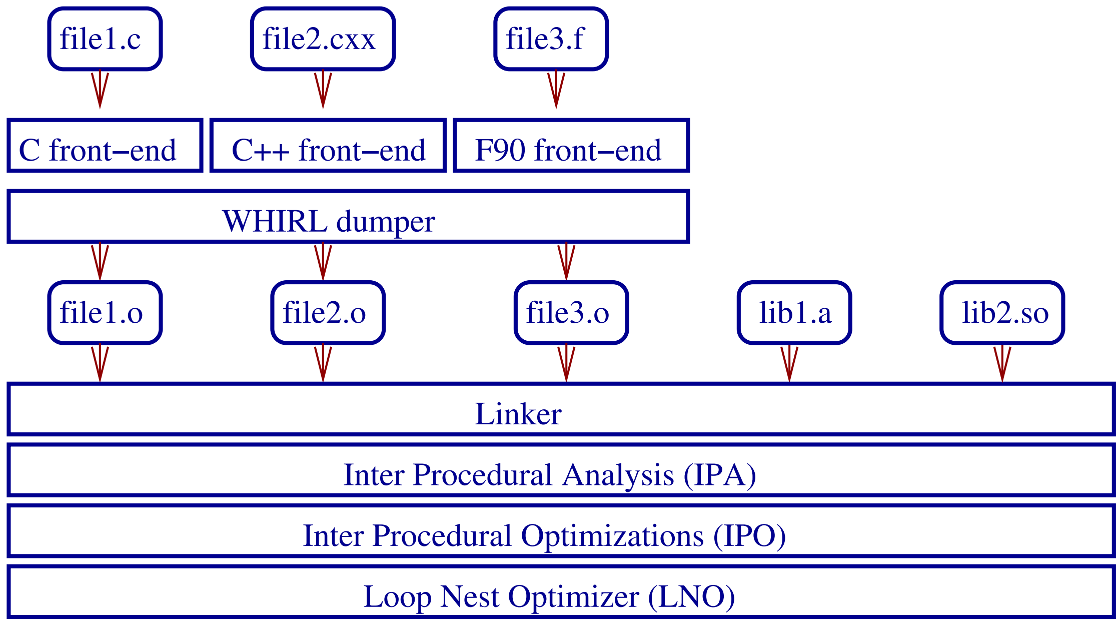


Some files contain WHIRL trees: the compilation is not complete, and the IPA is called.

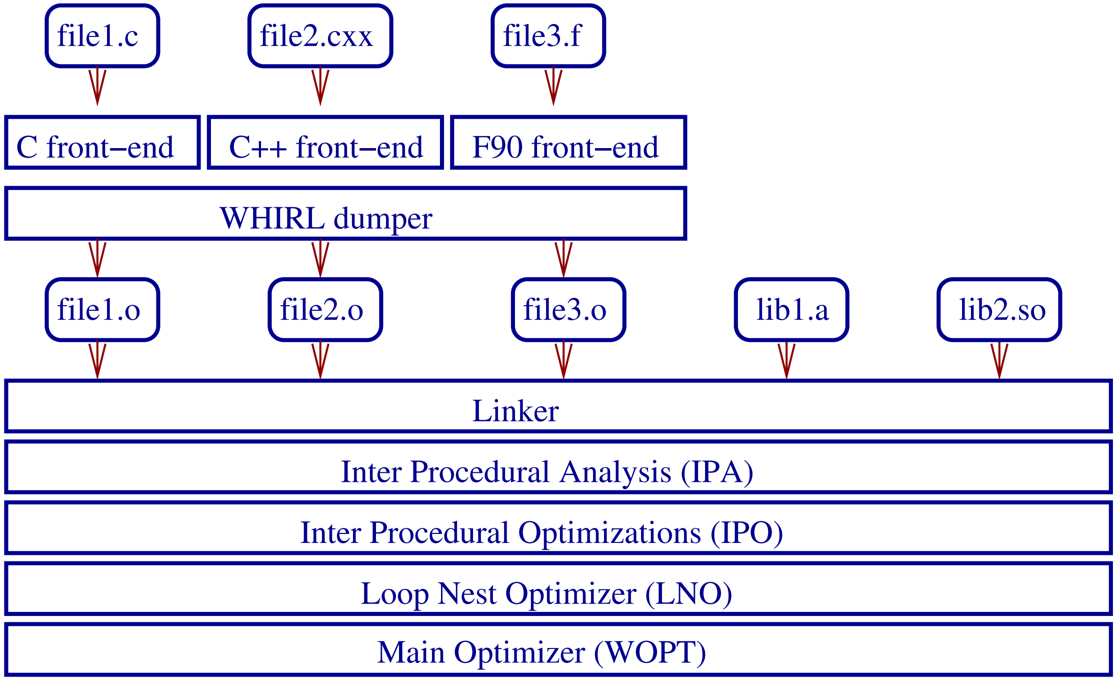
Inter Procedural Analysis



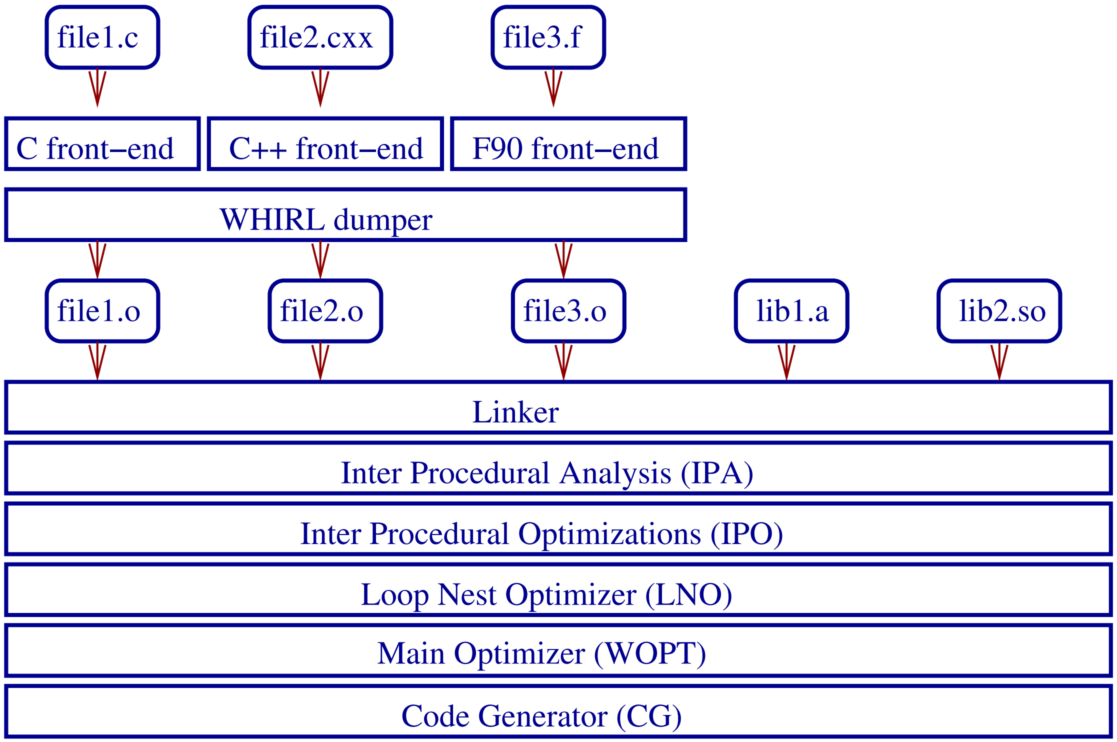
Inter Procedural Analysis



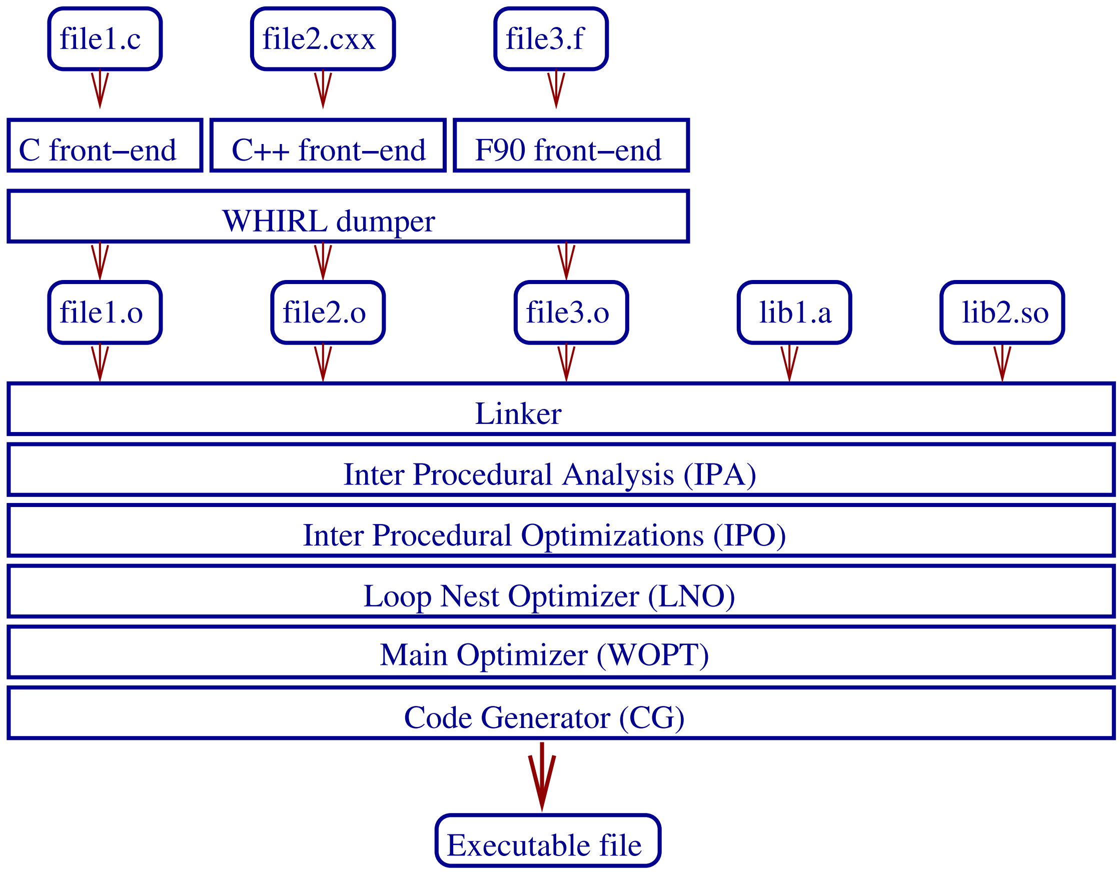
Inter Procedural Analysis



Inter Procedural Analysis



Inter Procedural Analysis



Inter Procedural Analysis

Idea: gather information over a whole project

Inter Procedural Analysis

Idea: gather information over a whole project
Solution:

- save WHIRL trees in .o files
- build a global tree at link time
- perform all optimizations
- generate code

Loop Nest Optimizer

LNO works on High level WHIRL.
Lowering removed unstructured control flow (gotos, switch, ...)

Loop Nest Optimizer

Analyzes extract information from WHIRL and construct specific Intermediate Representations (IRs):

- Array Dependence Graph
- LEGO: for data distributions
- Array and vectors accesses
- Vector space
- Systems of equations
- Polytope

Loop Nest Optimizer

Main optimizers in LNO:

- Loop unrolling
- Hoist conditionals
- Hoist varying lower bounds
- Dead store eliminate arrays
- Loop reversal / fission / fusion / tiling
- Array scalarization
- Prefetch
- Inter iteration Common Subexpression Elimination

Global Optimizer

WOPT works on Medium-level WHIRL
(arrays lowered into load/store + offset, ...)

Global Optimizer

Main intermediate representations:

- CFG (Control Flow Graph)
- SSA (Static Single Assignement)

Main optimizations:

- SSA-PRE (Partial Redundancy Elimination)
- DCE (Dead Code Elimination)
- IVR (Induction Variable Recognition)
- VNFRE (Value Numbering based Full Redundancy Elimination)
- Copy propagation

Code Generator

Code Generator works on CGIR.

- explicit CFG
- each BB contains a list of instructions
- each instruction is under the form
  OP_result OP_code OP_opnd

This representation is close to assembler code.

Code Generator

Main optimizers in CG are:

- EBO: Extended Block Optimizer
- GRA: Global Register Allocation
- LRA: Local Register Allocation
- GCM: Global Code Motion
- SWP: Software Pipelining
- CIO: Cross Iteration loop Optimizations
- FREQ: execution frequencies of BBs and edges

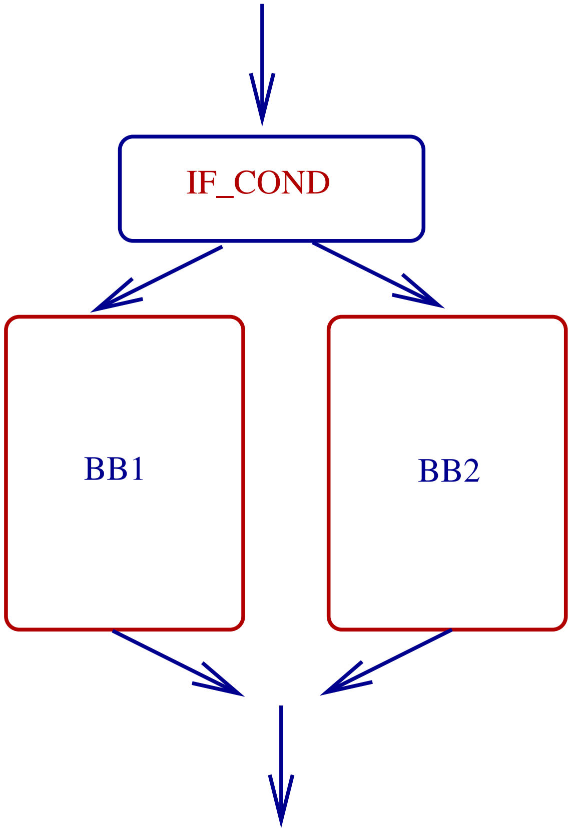
Open Research Compiler

ORC is an extension of the Code Generator.
ORC added the following infrastructure:

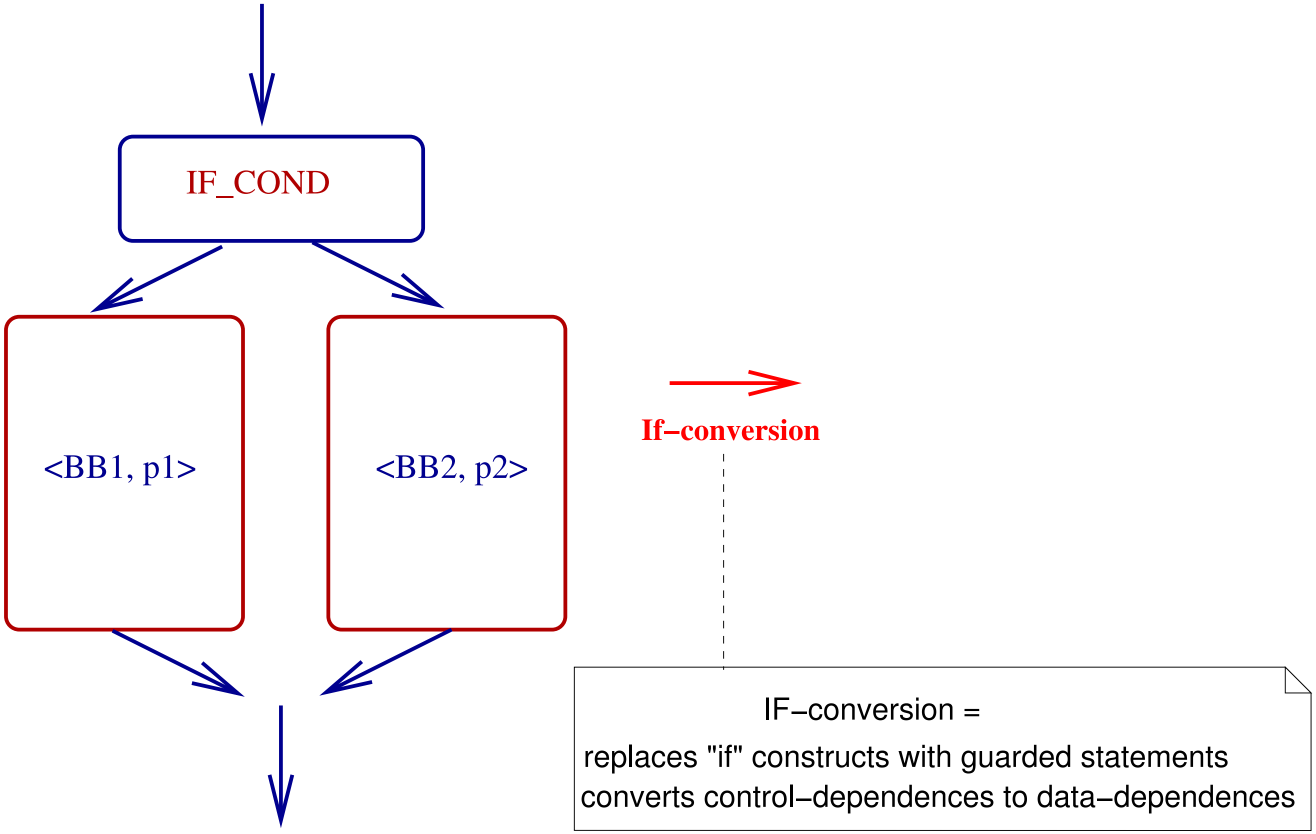
- IPFEC Regions: structures the CFG into a tree
- If-conversion
- PRDB: Predicate Relation DataBase
- Microscheduler
- Local/Global instruction scheduling

Partial Redundancy Elimination

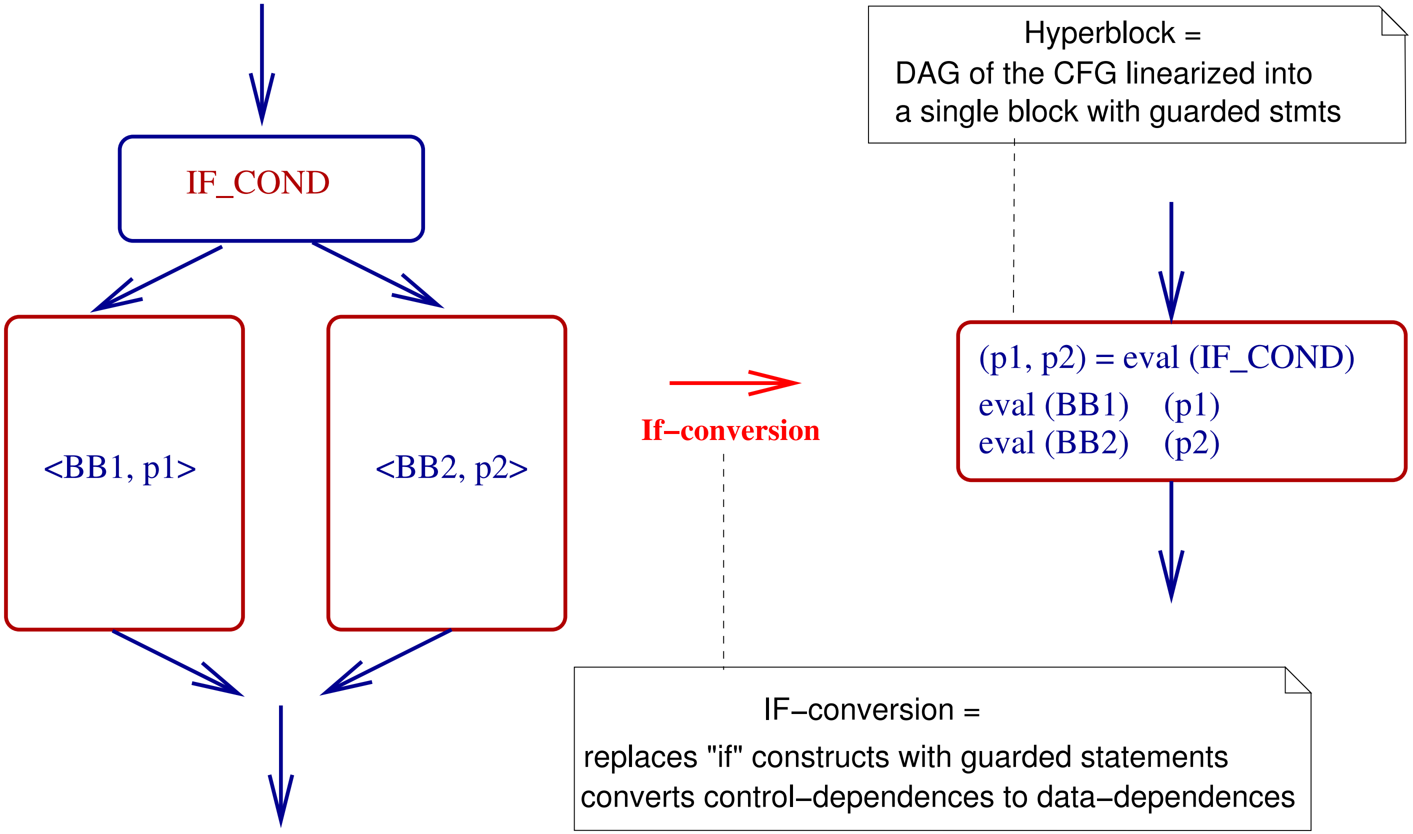
Predicated code



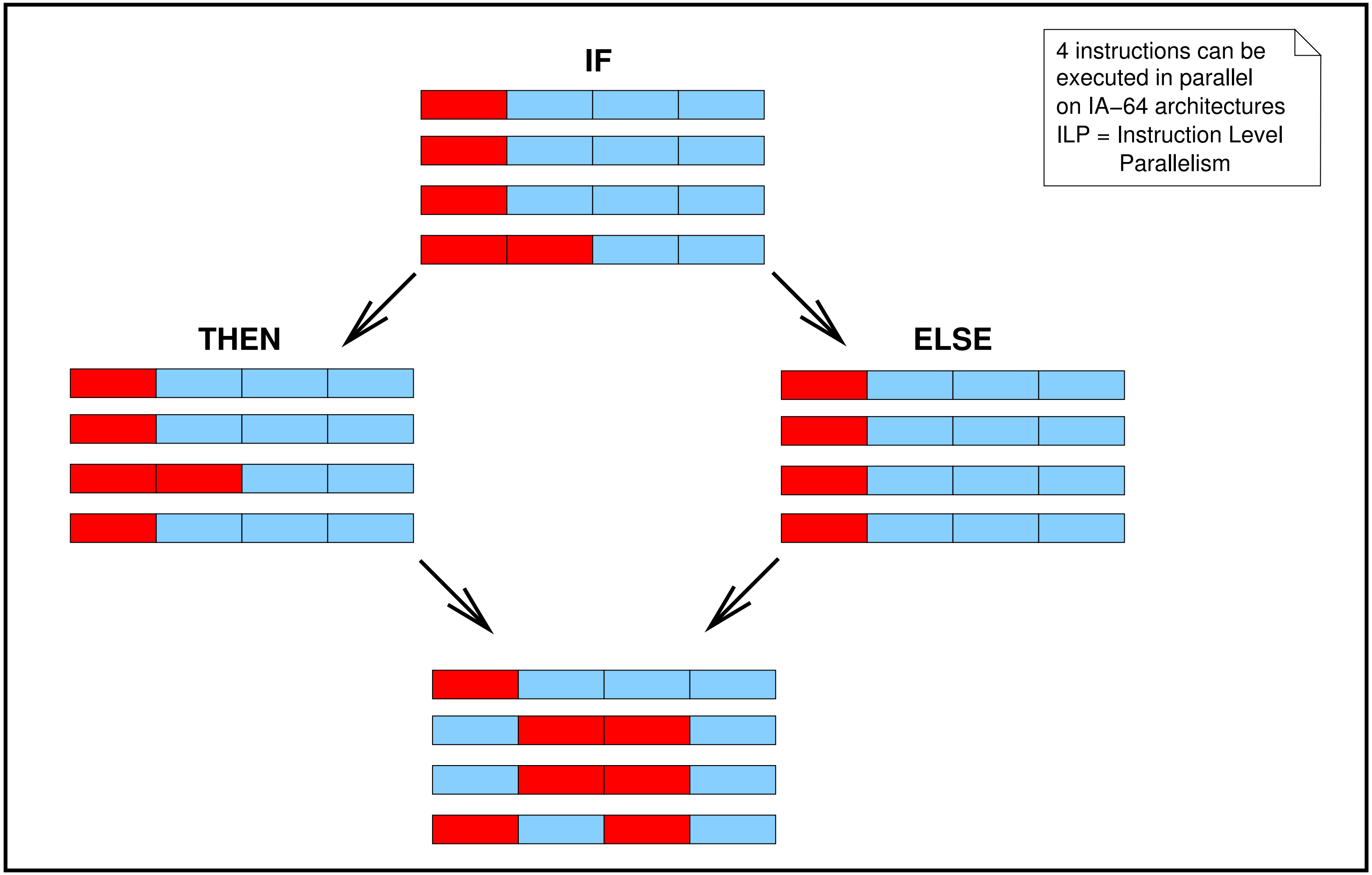
Predicated code



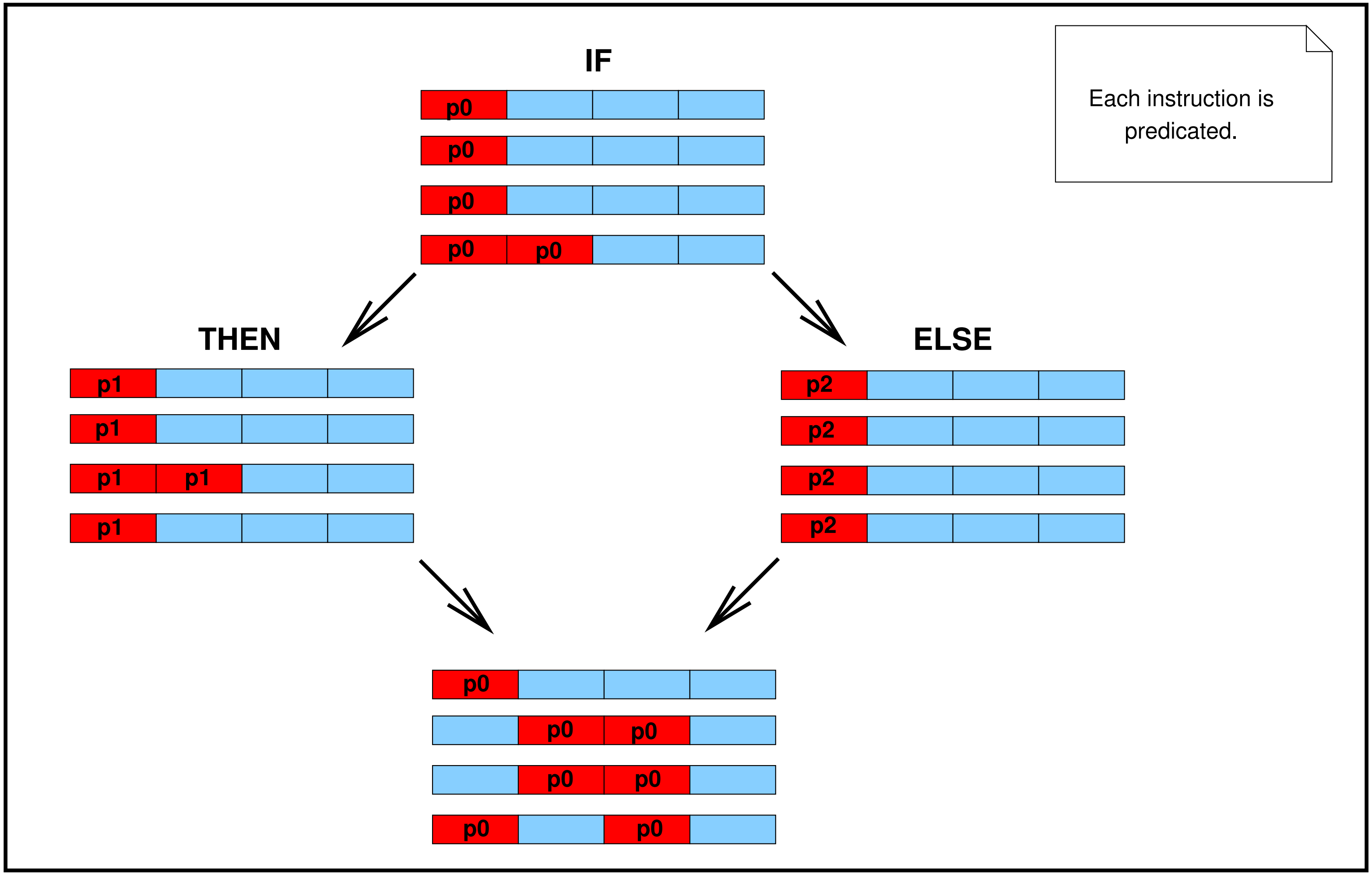
Predicated code



Predicated code



Predicated code



Predicated code