Interface et extension de Open Research Compiler

Sébastian Pop

Université Louis Pasteur Strasbourg,

Projet A3 INRIA,

FRANCE

Présentation générale

L'entreprise et l'encadrement:

- INRIA Rocquencourt
- Projet A3
- Encadrement: Albert Cohen
- Durée: 13 semaines (du 3 juin au 30 août)

Présentation du stage

Buts du stage:

- Découvrir le compilateur Open64/ORC
- Documenter le compilateur
- Implémenter une passe dans le générateur de code.

Structure du compilateur

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

- Front-ends C et C++ de GCC
- Fortran 90 de Cray
- Chaque front-end a ses propres AST
- Traduction des AST vers WHIRL

WHIRL

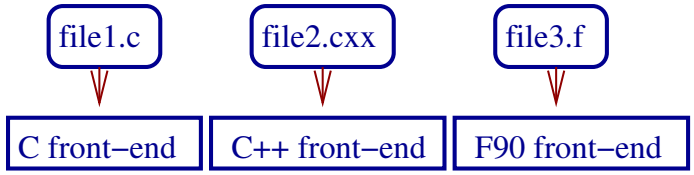
Winning Hierarchical Intermediate Representation Language

- 5 niveaux: VH, H, M, L, VL
- *lowering* entre niveaux.
- Chaque optimization au bon niveau.

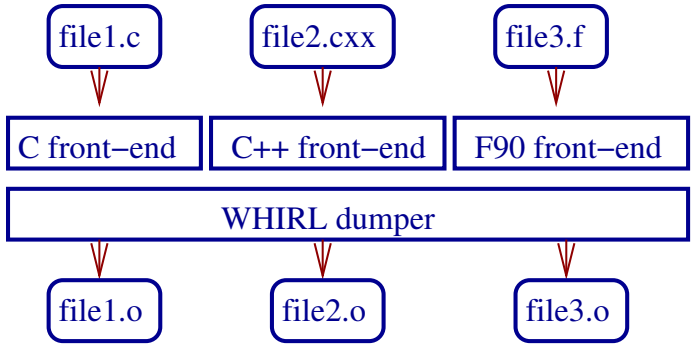
Inter Procedural Analysis

file1.c file2.cxx file3.f

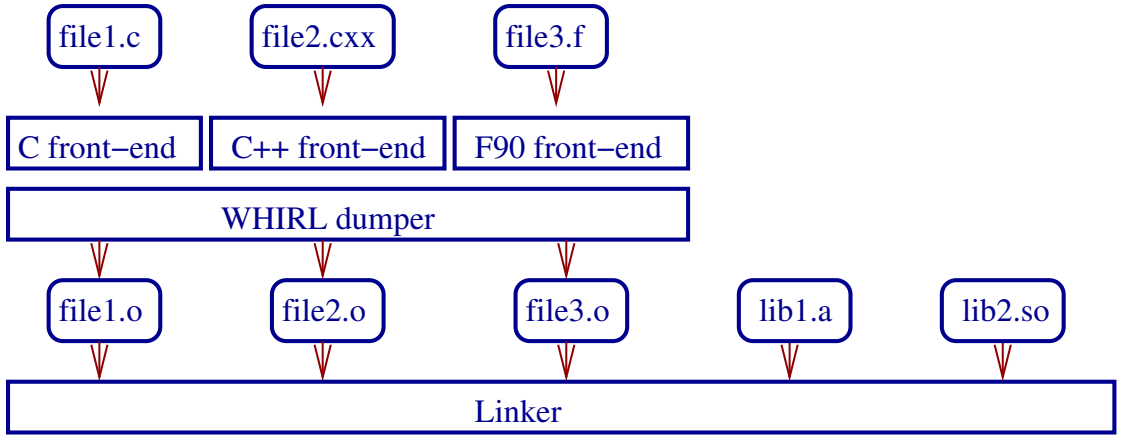
Inter Procedural Analysis



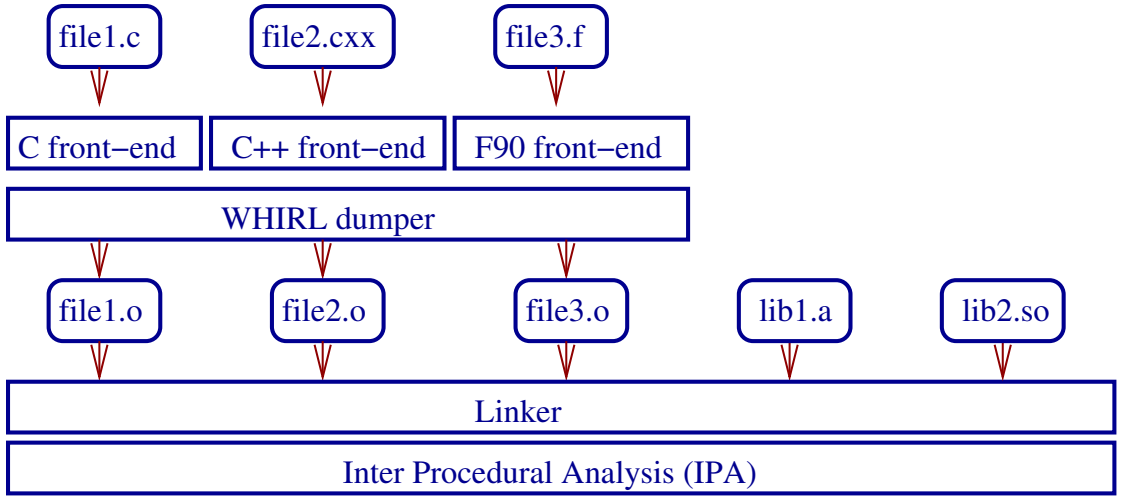
Inter Procedural Analysis



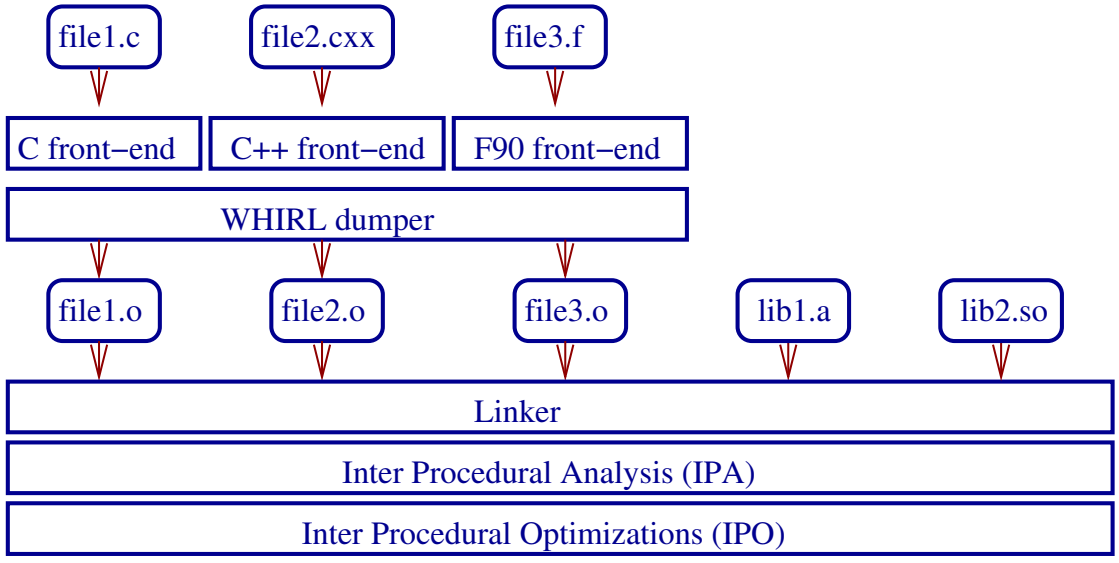
Inter Procedural Analysis



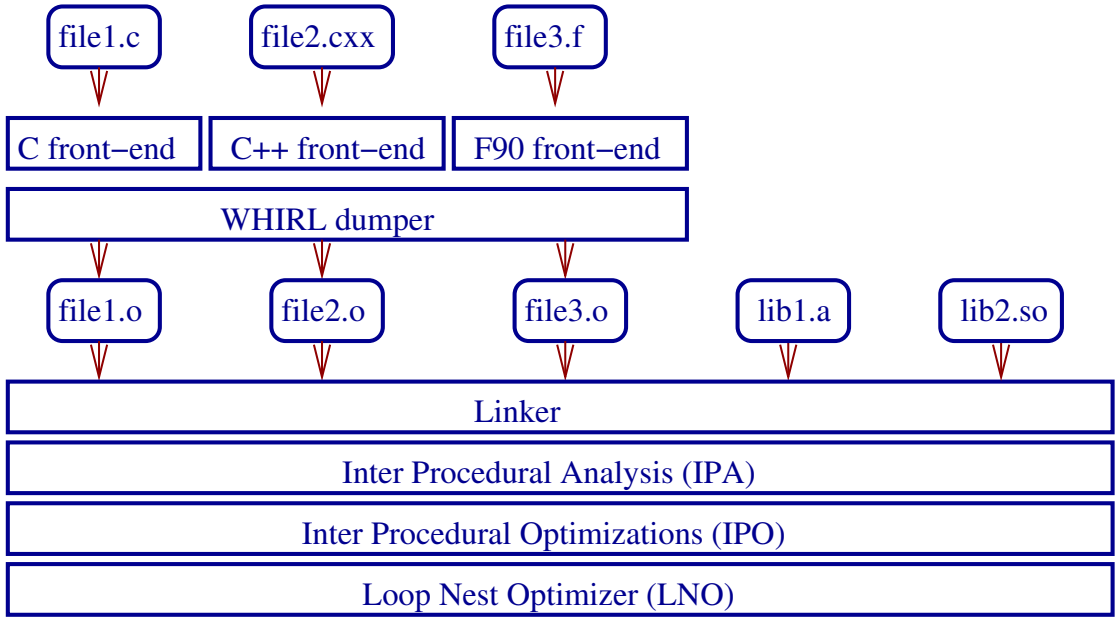
Inter Procedural Analysis



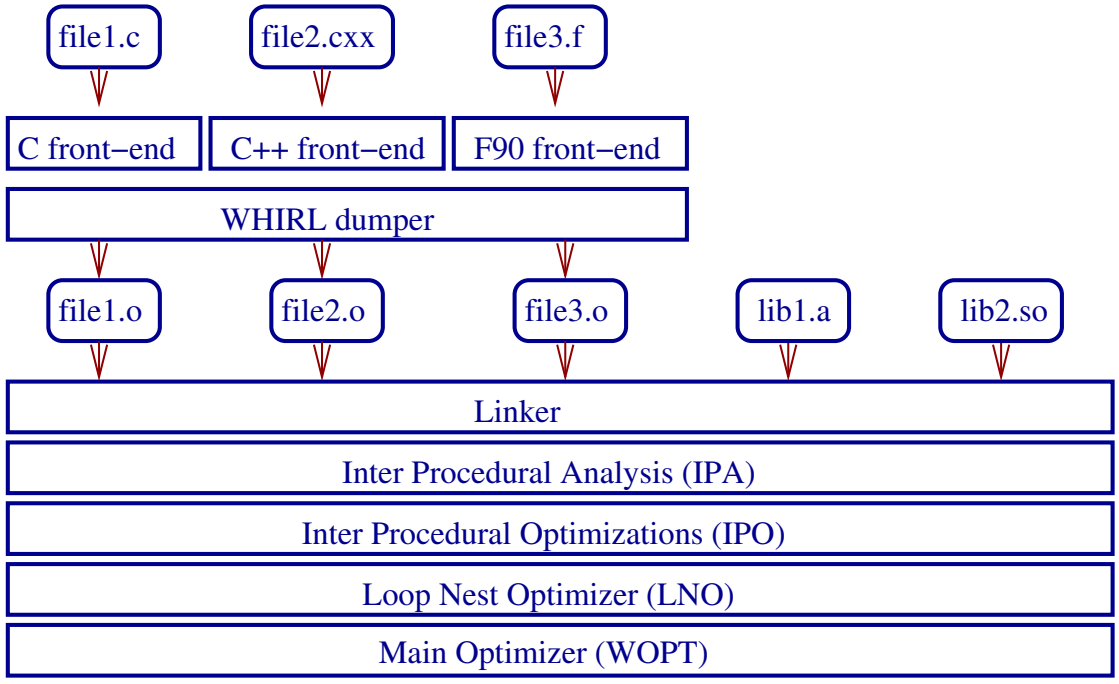
Inter Procedural Analysis



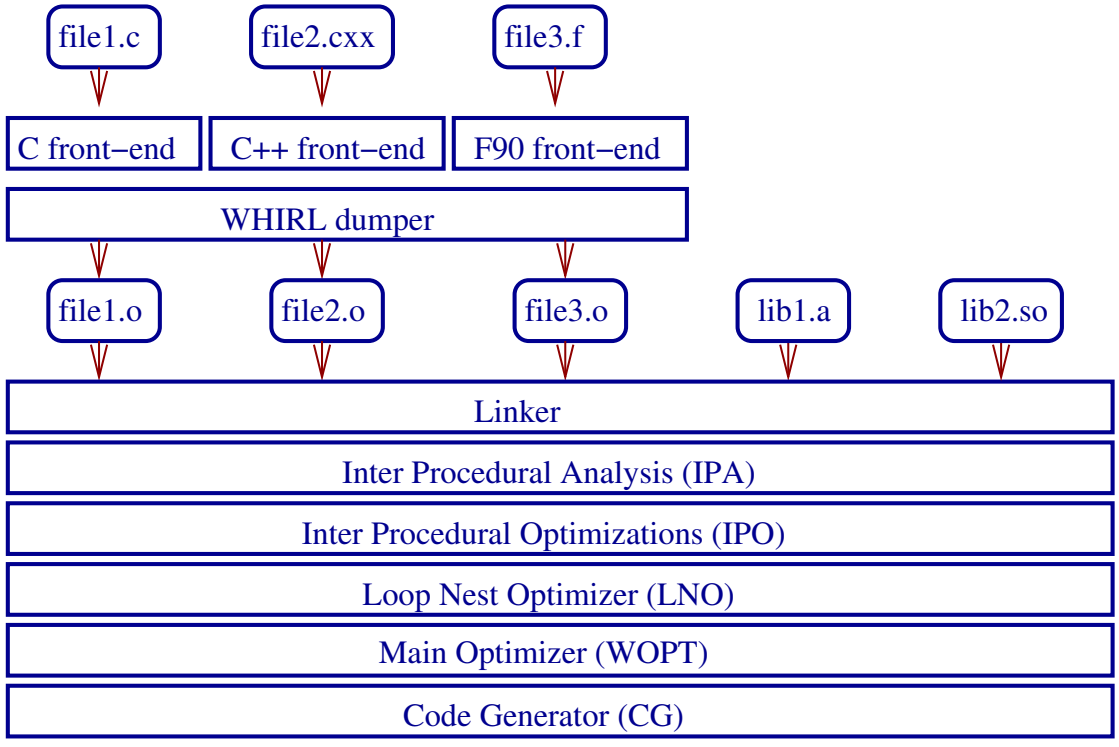
Inter Procedural Analysis



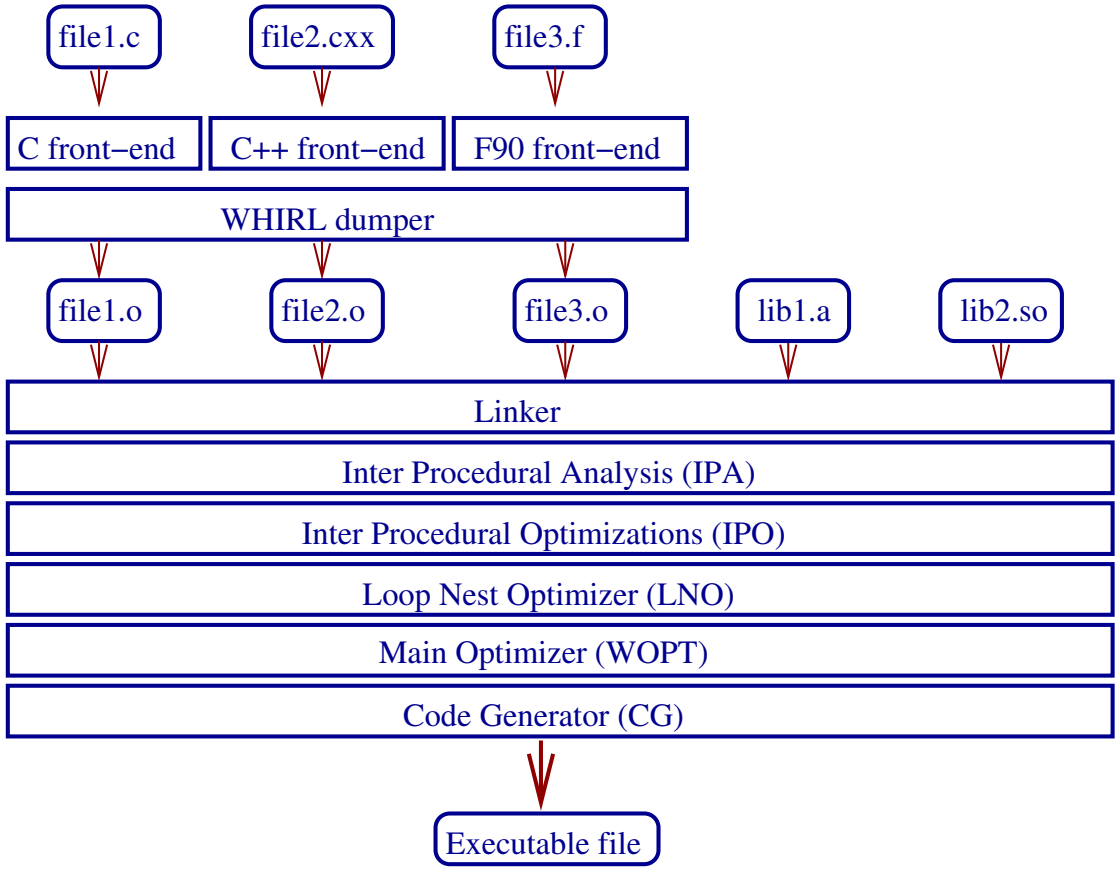
Inter Procedural Analysis



Inter Procedural Analysis



Inter Procedural Analysis



Inter Procedural Analysis

Rassembler l'information sur l'ensemble du projet.

Inter Procedural Analysis

Rassembler l'information sur l'ensemble du projet.
Solution:

- sauver la WHIRL dans les .o
- reconstruire un AST global

Loop Nest Optimizer

LNO travaille sur le *High level WHIRL*.

Loop Nest Optimizer

Représentations intermédiaires spécifiques:

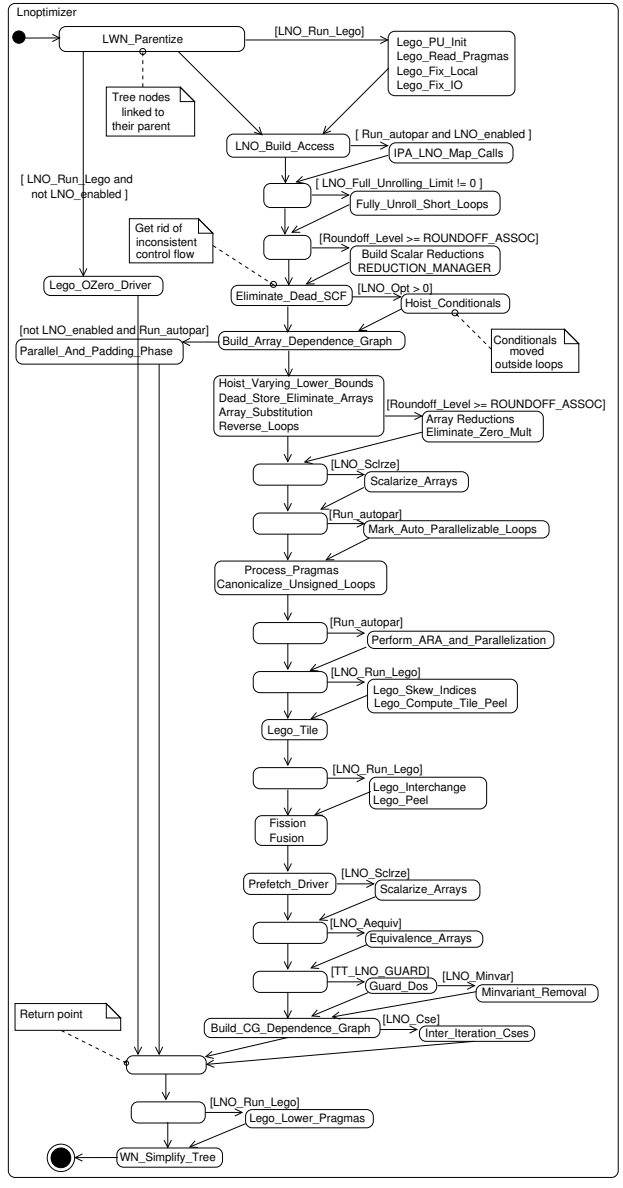
- Array Dependence Graph
- LEGO: for data distributions
- Array and vectors accesses
- Vector space
- Systems of equations
- Polytopes

Loop Nest Optimizer

Quelques optimiseurs du 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

Loop Nest Optimizer



Global Optimizer

WOPT travaille sur le *Medium-level WHIRL*.

Global Optimizer

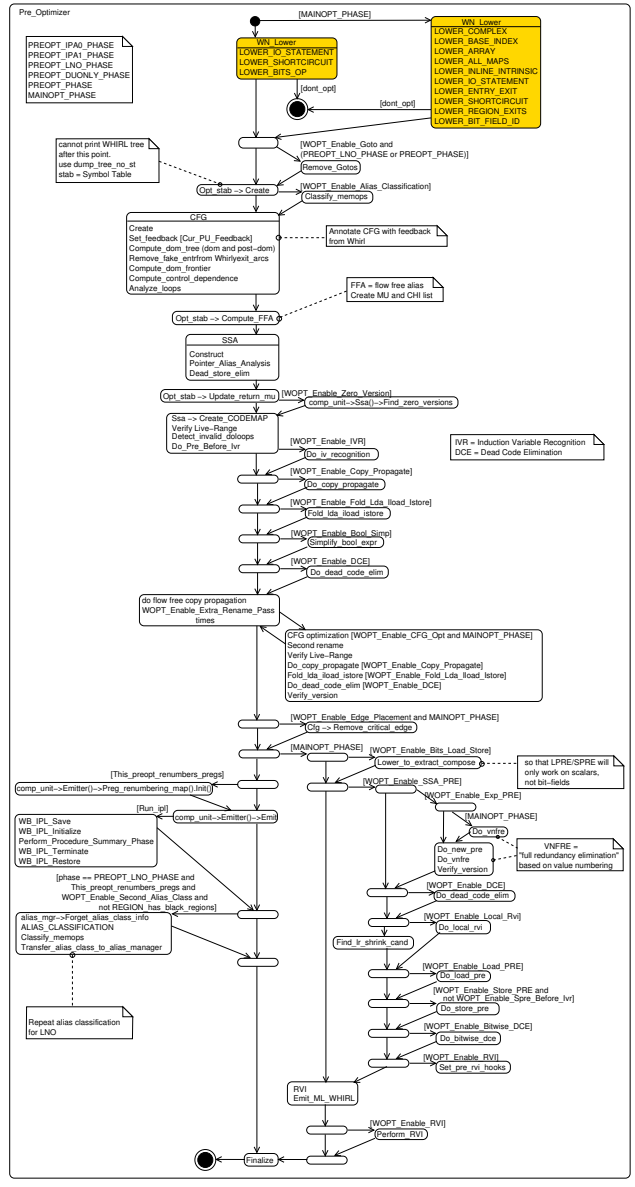
Principales représentations intermédiaires:

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

Quelques optimiseurs:

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

Global Optimizer



Code Generator

Le générateur de code travaille sur la CGIR.

- CFG explicite
- chaque BB contient une liste d'instructions
- chaque instruction est sous la forme:
  OP_result
  OP_code
  OP_opnd

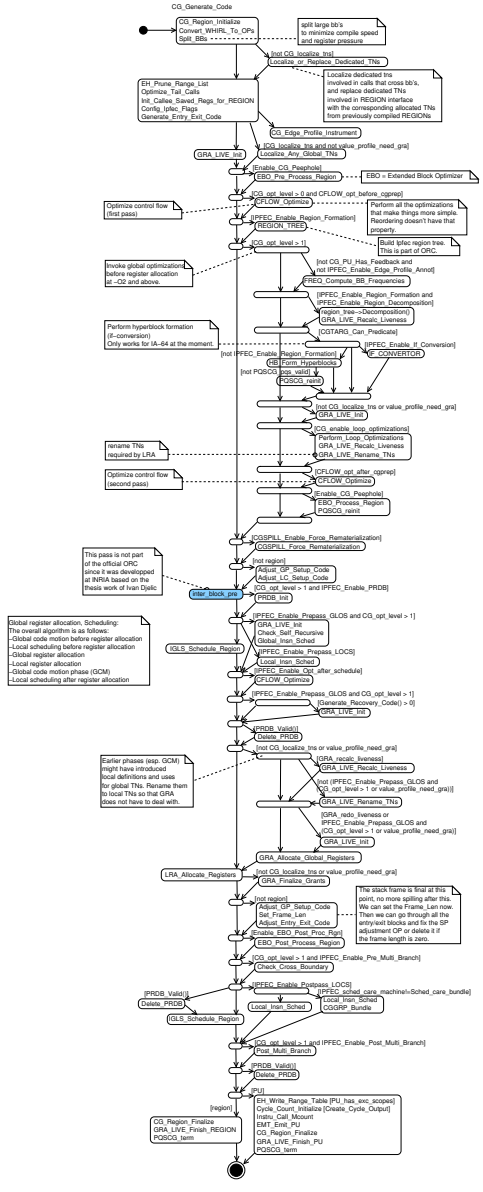
Représentation proche du code assembleur.

Code Generator

Principales passes du CG:

- 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: fréquences d'execution des BBs

Code Generator



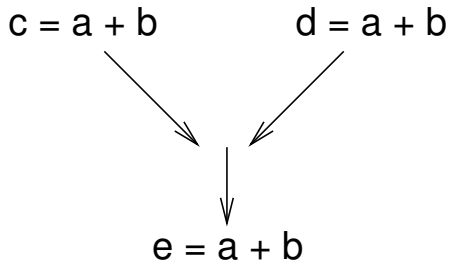
Open Research Compiler

ORC est une extension du générateur de code.

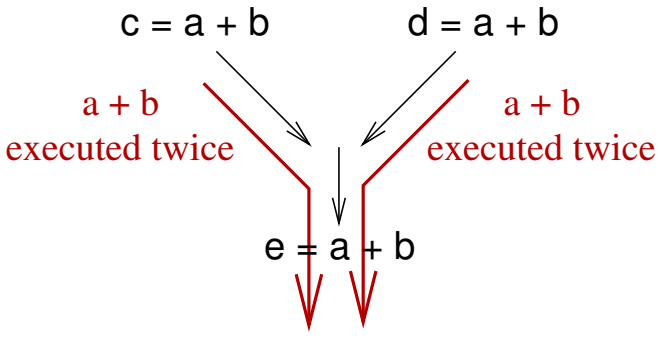
- IPFEC Regions
- If-conversion
- Predicate Relation DataBase
- Microscheduler
- Local/Global instruction scheduling

Partial Redundancy Elimination

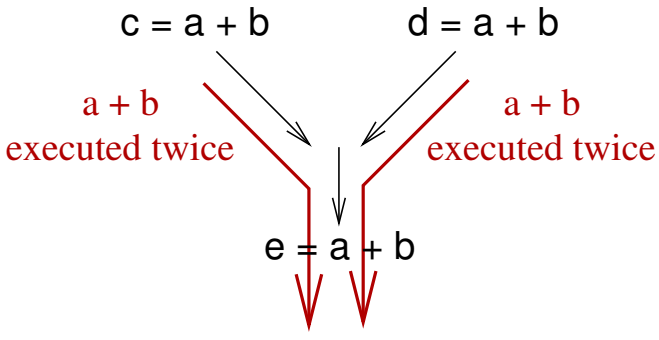
Partial Redundancy Elimination



Partial Redundancy Elimination

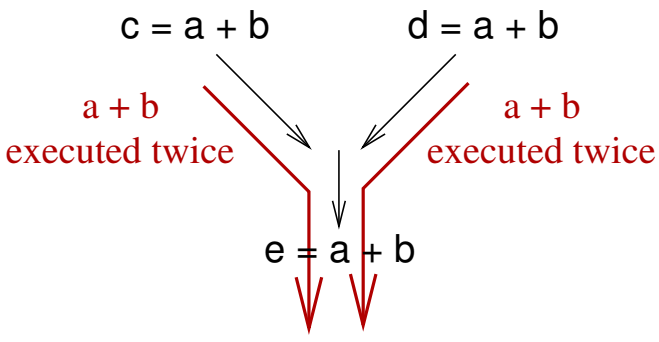


Partial Redundancy Elimination

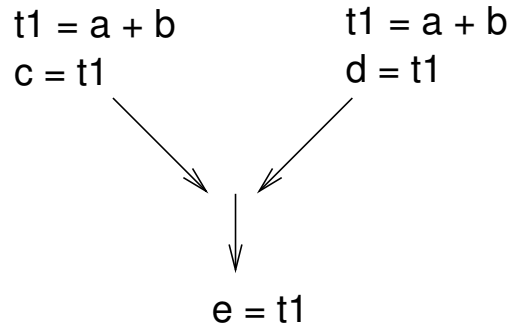


a + b is fully available

Partial Redundancy Elimination

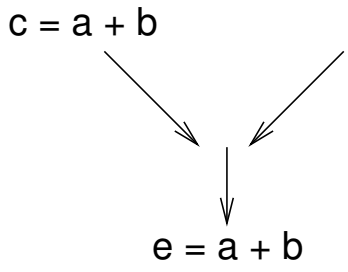


a + b is fully available



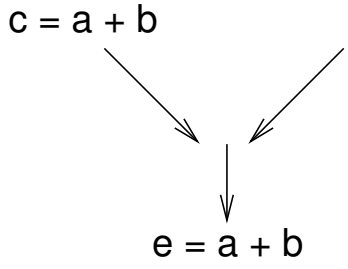
elimination of common subexpression

Partial Redundancy Elimination

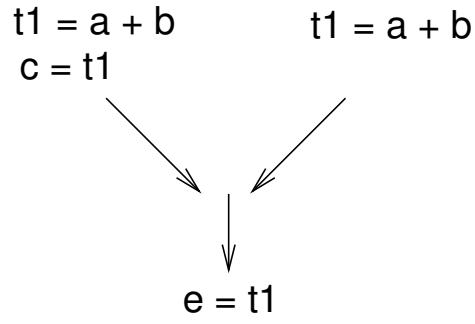


Expression a + b is
partially available

Partial Redundancy Elimination

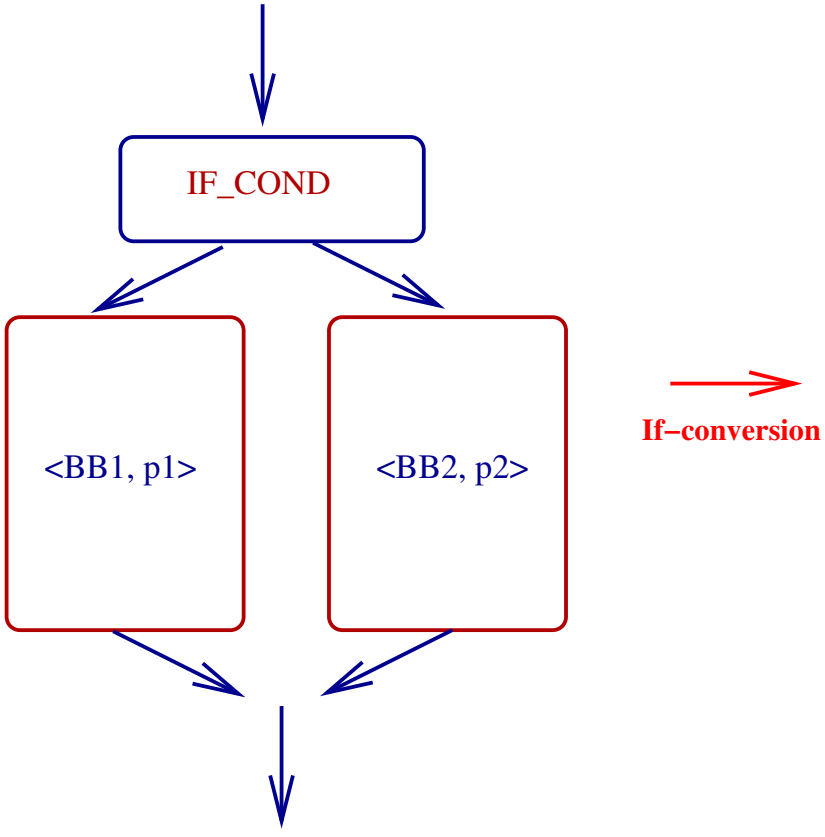


Expression a + b is
partially available

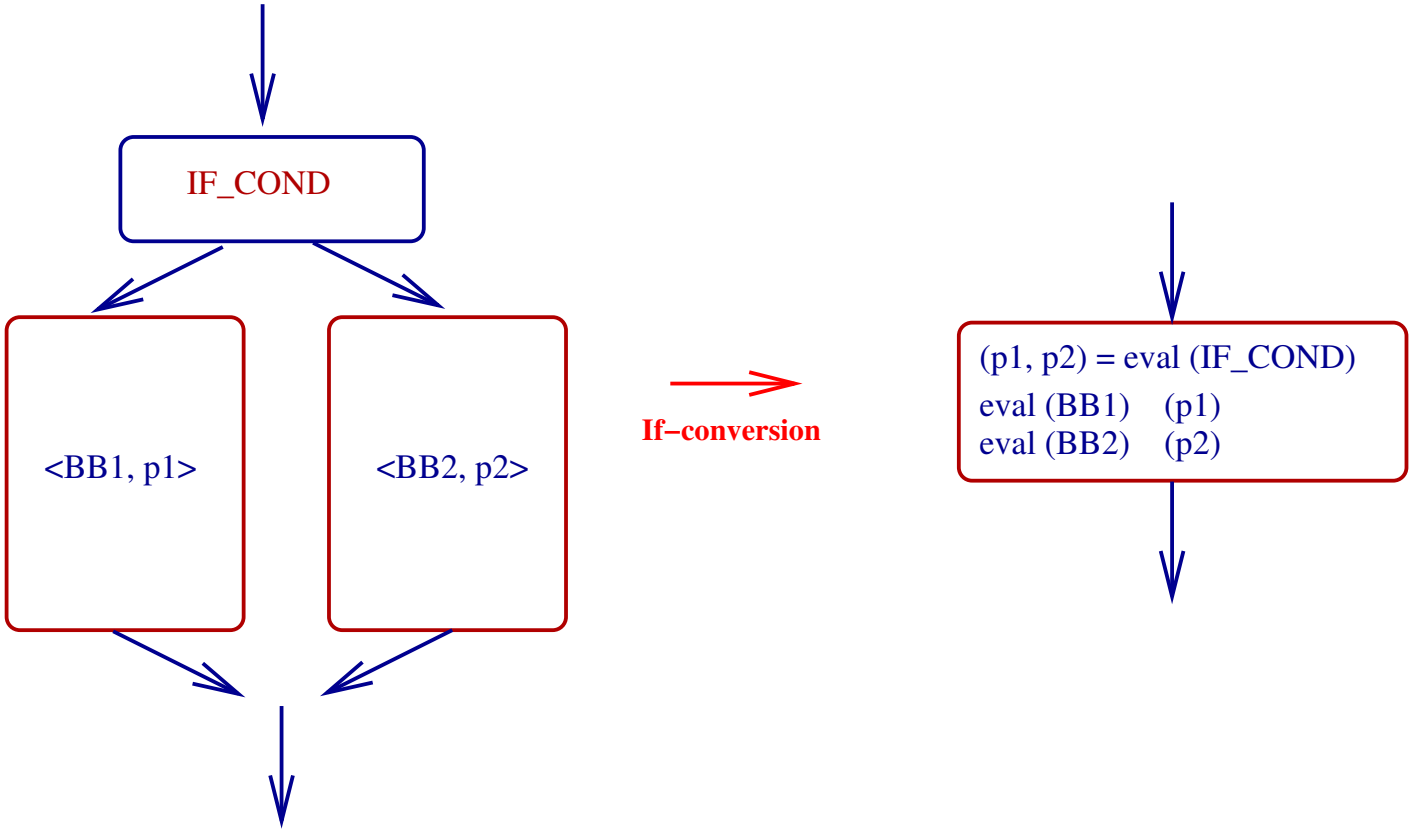


elimination of partial
redundancy

Code Prédicaté



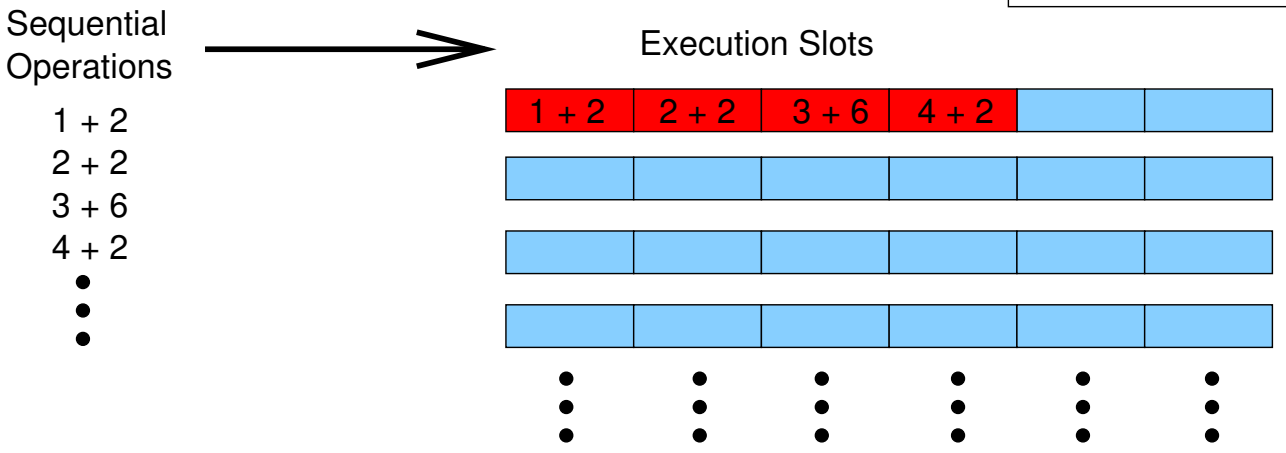
Code Prédicaté



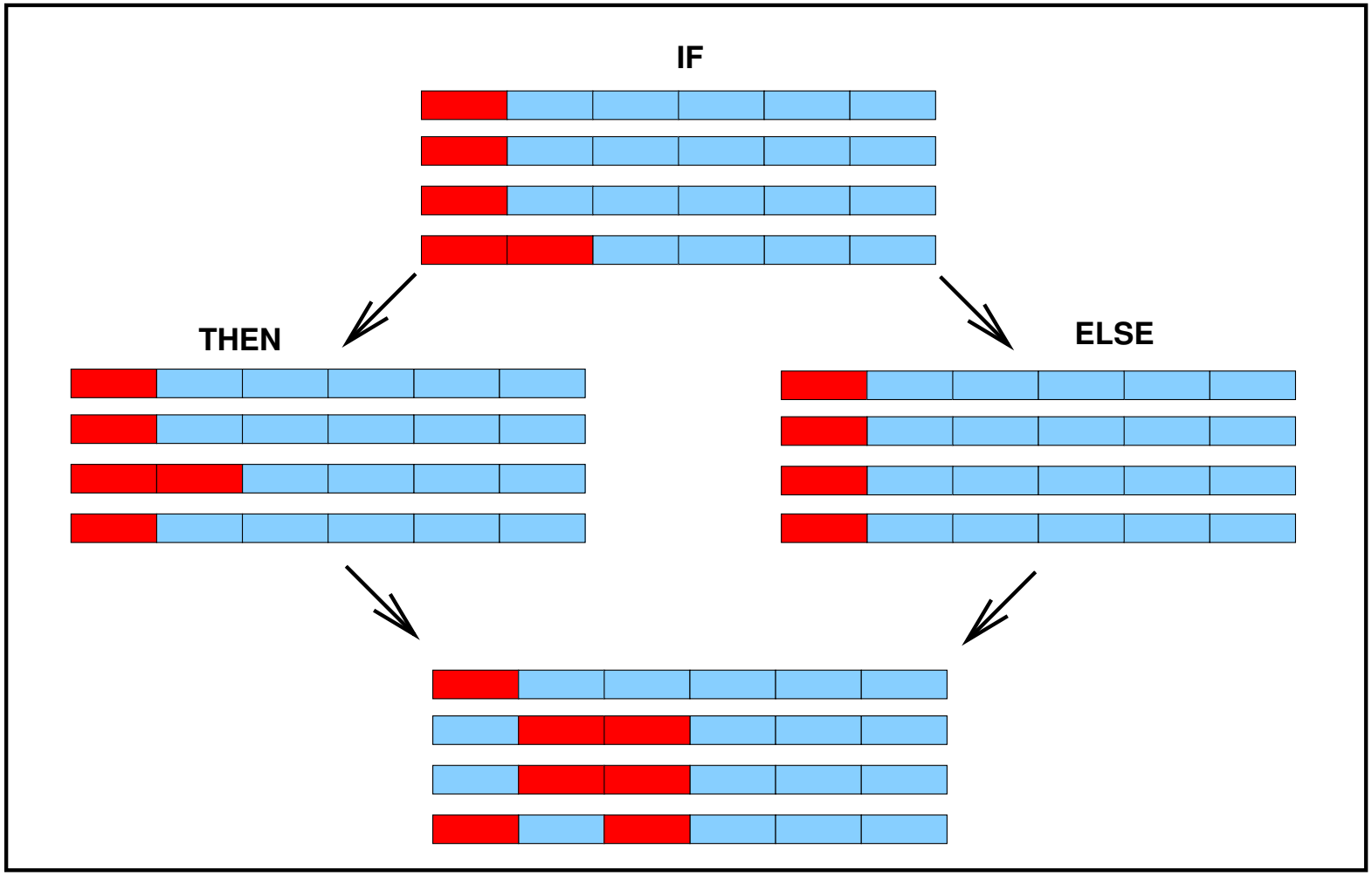
Code Prédicaté

6 instructions can be executed in parallel on Itanium

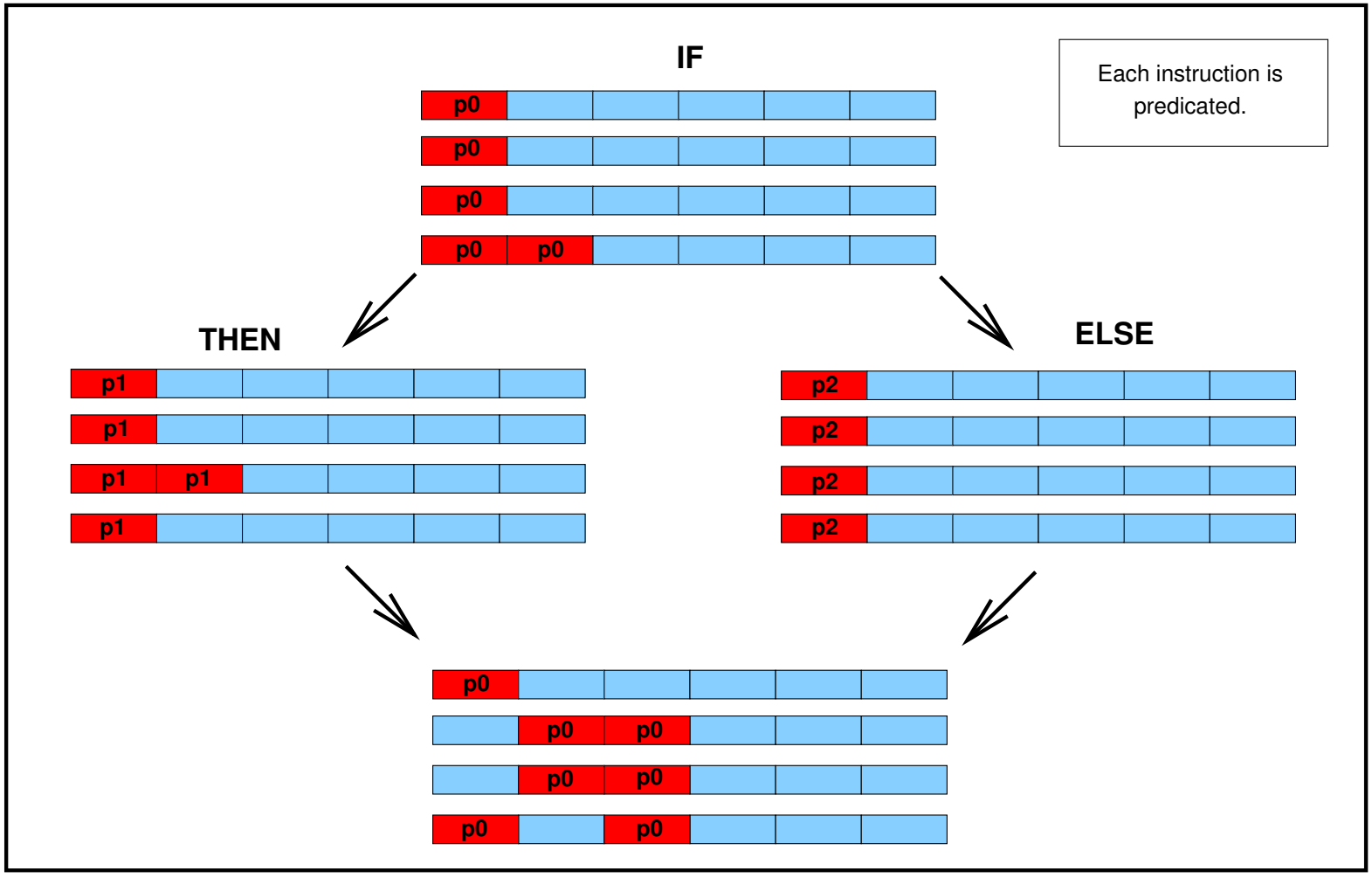
ILP = Instruction Level Parallelism



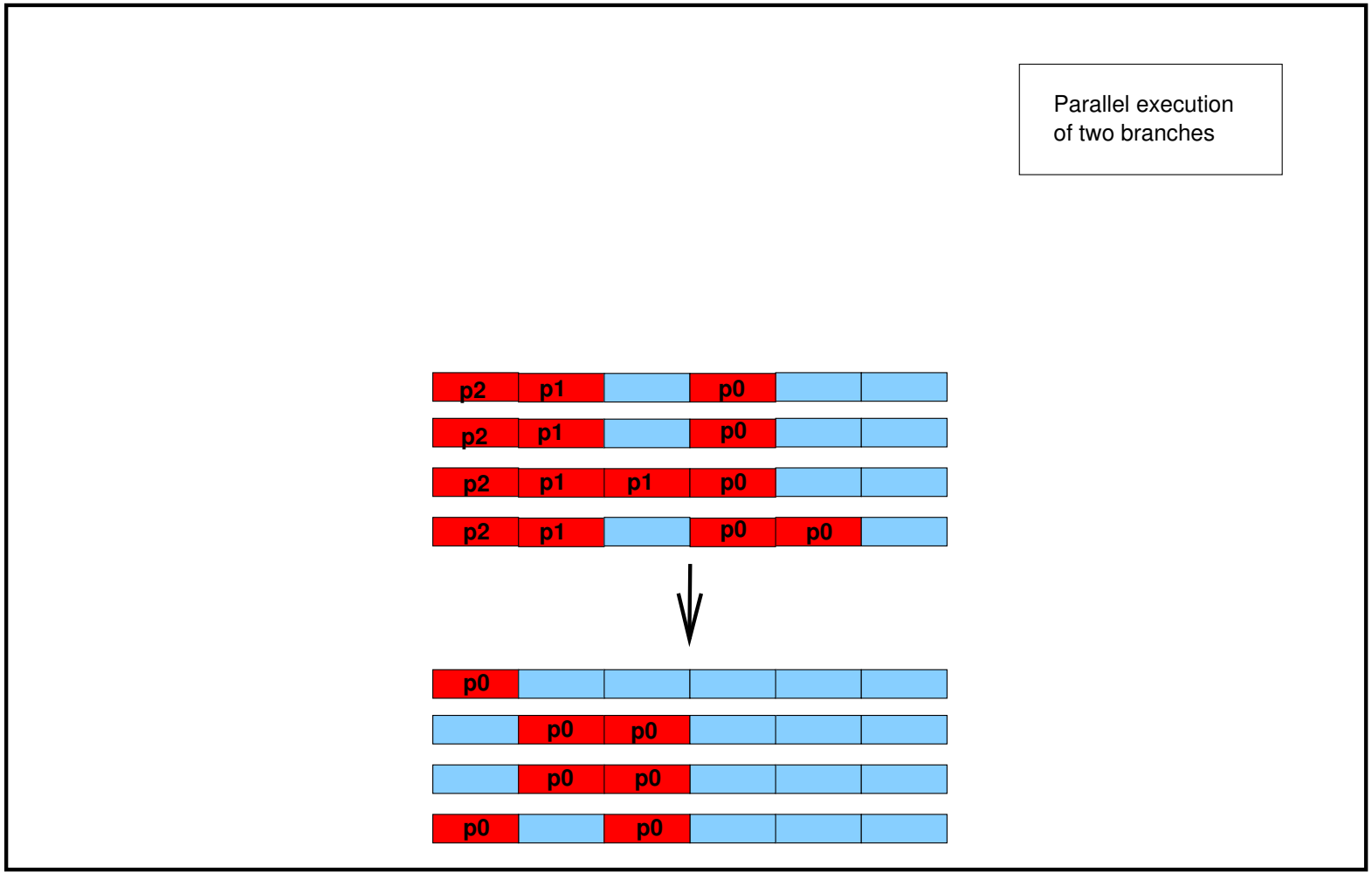
Code Prédicaté



Code Prédicaté



Code Prédicaté

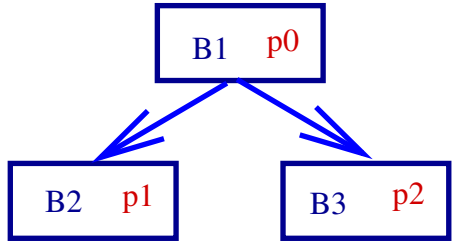


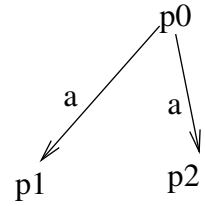
Predicate Partition Graph

p0

Atomic Predicates = {p0}

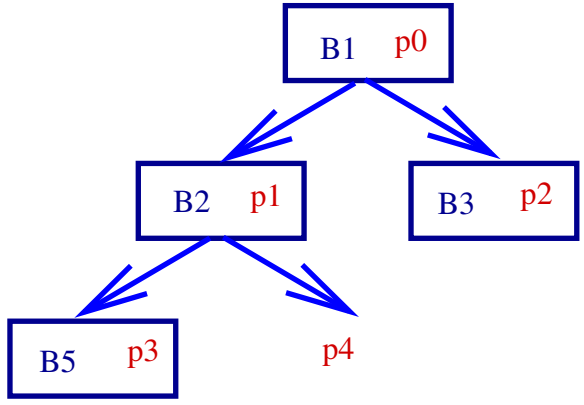
Predicate Partition Graph

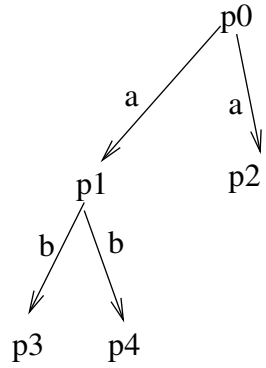




Atomic Predicates = {p1, p2}

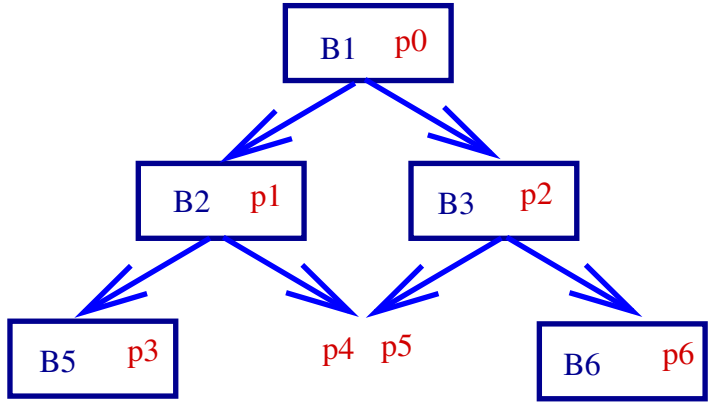
Predicate Partition Graph

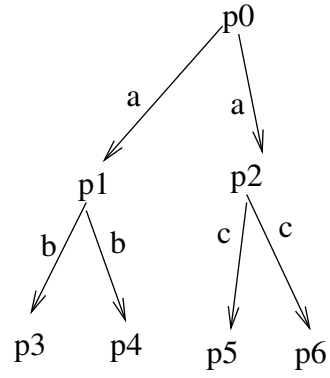




Atomic Predicates = {p3, p4, p2}

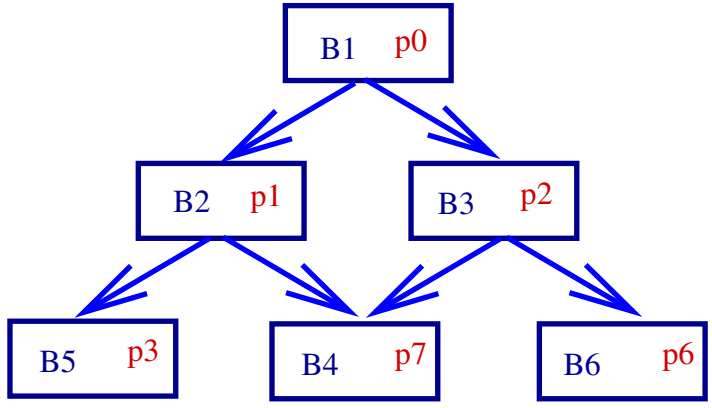
Predicate Partition Graph

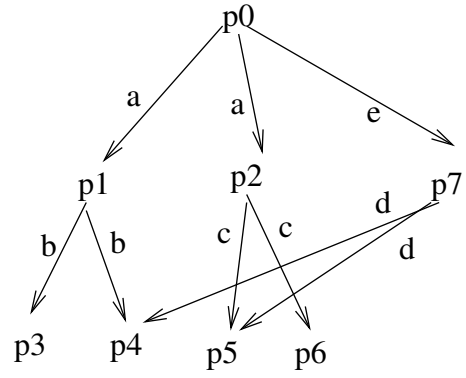




Atomic Predicates = {p3, p4, p5, p6}

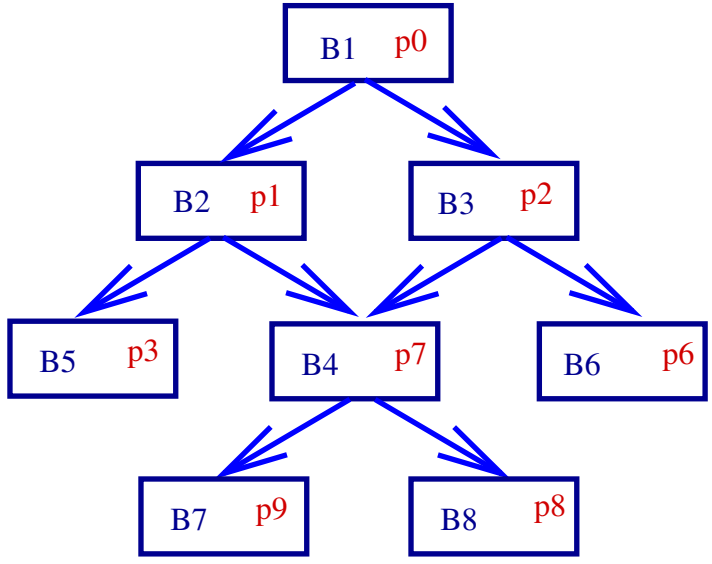
Predicate Partition Graph

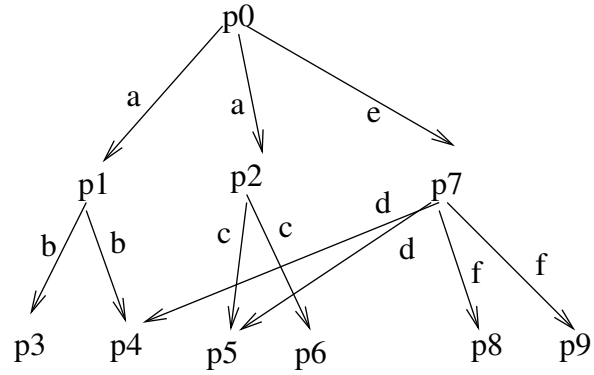




Atomic Predicates = {p3, p4, p5, p6}

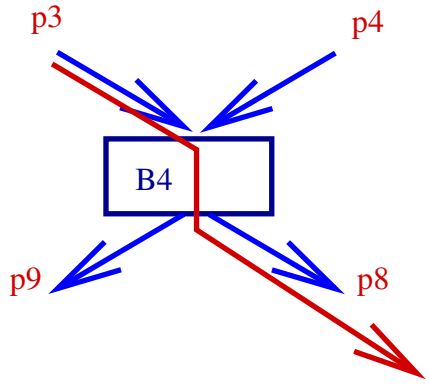
Predicate Partition Graph





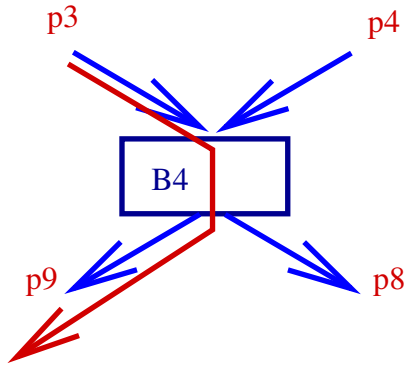
Atomic Predicates = {p3, p4 inter p8, p5 inter p8, p4 inter p9, p5 inter p9, p6}

Prédicats atomiques



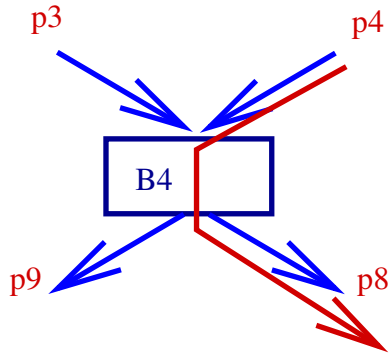
Atomic Predicates = {p3 inter p8,

Prédicats atomiques



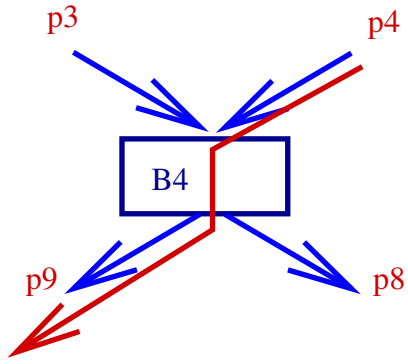
Atomic Predicates = {p3 inter p8, p3 inter p9,

Prédicats atomiques



Atomic Predicates = {p3 inter p8, p3 inter p9, p4 inter p8,

Prédicats atomiques



Atomic Predicates = {p3 inter p8, p3 inter p9, p4 inter p8, p4 inter p9}

PRE sur code prédicaté

1. L'analyse du flot de données propage des ensembles de prédicats.
2. Deux propriétés sont calculées pour chaque BB:
   - *anticipability*
   - *availability*
3. Insertion de variables temporaires aux points au plus tôt.
4. Supression des expression redondantes des BB où *avail* est vraie.

Résultats

Documentation du compilateur:

- Une présentation générale sous forme de slides
- Le rapport de stage
- Une page web http://www-rocq.inria.fr/~pop/

Résultats

Implémentation:

- Spécification algébrique
- Raffinement en C++
- Intégration dans ORC

Spécification Algébrique

- Outil de communication
- Typage évite des erreurs
- Raffinements ultérieurs
- Difficultés du domaine séparés des difficultés de l'intégration dans un système complexe.

Conclusion

- Découverte d'un nouveau compilateur
- Idées pour améliorer GCC
- Travail dans une équipe de recherche

Conclusion

Remerciements:

- Merci à l'équipe du projet A3 pour m'avoir proposé ce stage intéressant.
- Un grand merci à Albert Cohen pour son temps et pour son aide pendant le stage.

Open64 vs. GCC

“state of the art” compilers

Open64 vs. GCC

Open64:

- LNO, IPA
- un excellent paralléliseur de code
- architecture complètement modulaire

Open64 vs. GCC

GCC:

- support pour plus de 40 architectures
- 5 front-ends (C, C++, Java, Fortran, Ada)
- multitude d'autres langages portés (Pascal, Cobol, CLisp, Mercury, . . .)
- support pour le développement:
  - bug database
  - test-suites
  - documentation
  - mailing lists de développement actives
- pas (encore) de support pour LNO ou IPA.