A Mapping Approach Between IR and Binary CFGs dealing with Aggressive Compiler Optimizations for Performance Estimation

Omayma Matoussi Frédéric Pétrot

TIMA Laboratory – 46 Avenue Félix Viallet, 38031, Grenoble, France

01/24/2018

Introduction

- MPSoCs are getting more software-centric.
- SW has an impact on the performance of MPSoCs.
- Accurate feedback on SW performance is necessary during early phases of MPSoC design.

$\Longrightarrow$ Instruction Interpretation Approaches (ISS, DBT, etc.):

- target instructions transformed into host instructions,
- accurate,
- **very slow**.

Introduction

- MPSoCs are getting more software-centric.
- SW has an impact on the performance of MPSoCs.
- Accurate feedback on SW performance is necessary during early phases of MPSoC design.

$\Longrightarrow$ Instruction Interpretation Approaches (ISS, DBT, etc.):

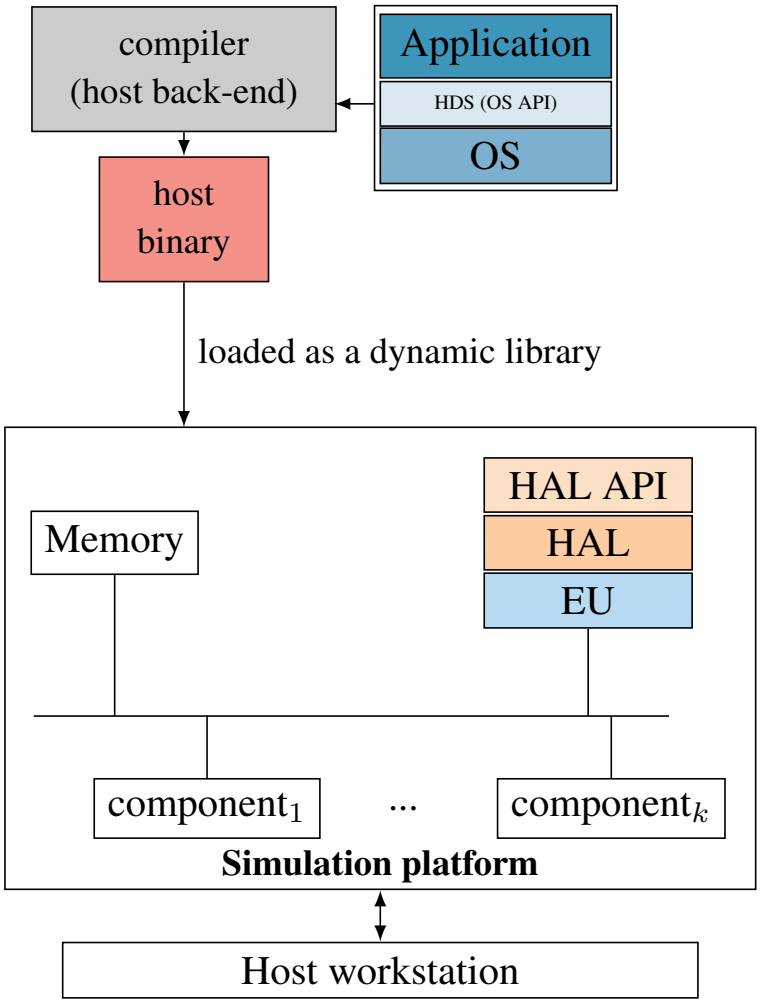
- target instructions transformed into host instructions,
- accurate,
- **very slow**.

$\Longrightarrow$ Native Simulation (a.k.a. host-compiled simulation):

- SW compiled and executed on the host machine,
- abstraction of low-level architectural details,
- **fast**.

Introduction

- Native Simulation



Overview of a Native simulation platform

- Execution Unit (EU) implements:
  - Hardware Abstraction Layer (HAL) API.

Introduction

Lack of performance information in Native Simulation

- Originally developed for purely functional verification of SW on top of a virtual platform,
- Absence of non-functional information (e.g. execution time).

$\Longrightarrow$ How to obtain performance estimates using Native Simulation?

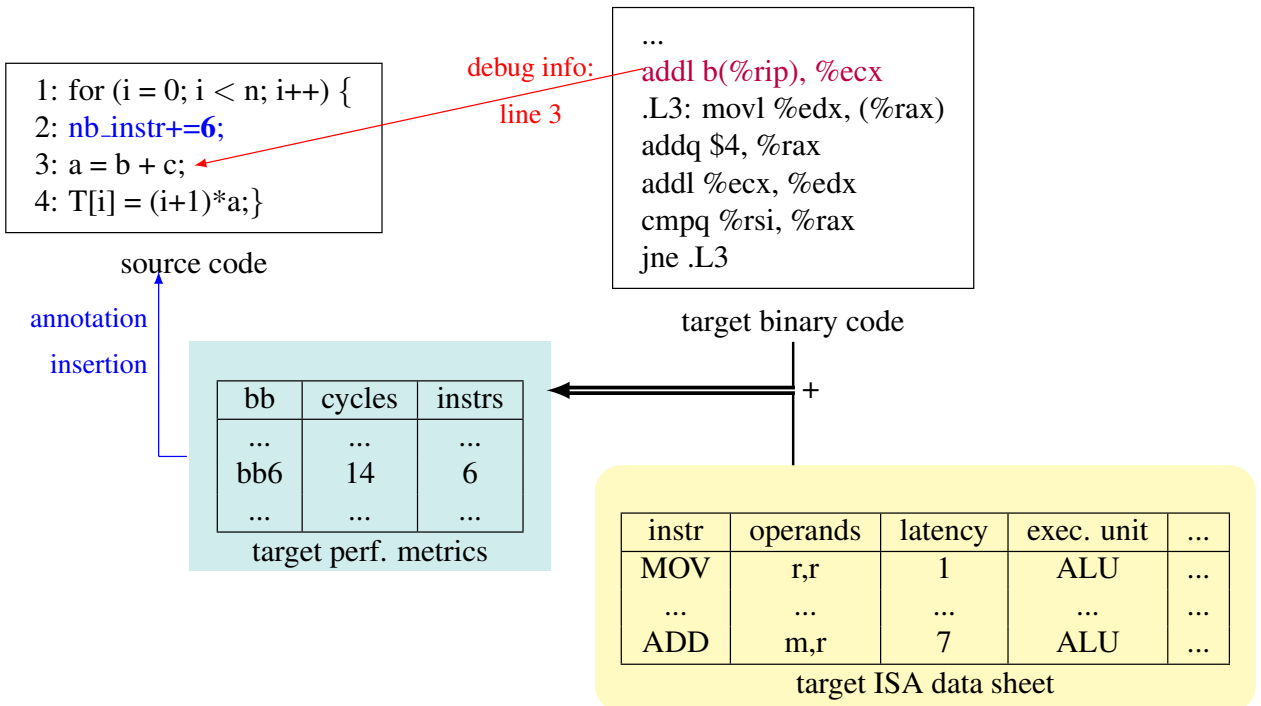
Outline

Outline

Software Back-annotation

Software Back-Annotation

Non-functional information (e.g. timing properties) is computed using low-level analysis and is inserted into the functional model (SW).



Source code annotation

Software Back-annotation

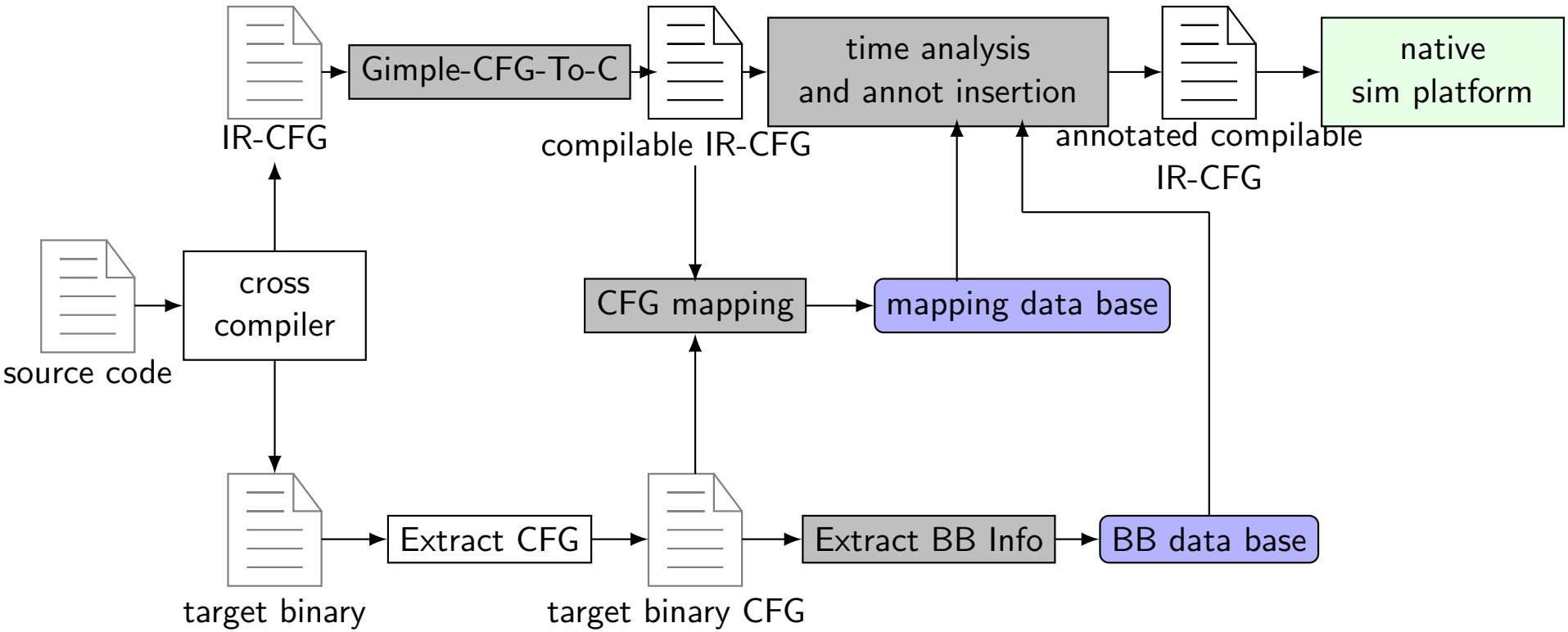
- How to compute non-functional information? (target binary analysis + modeling micro-architectural components)
- How to introduce target-specific performance metrics into the functional model (SW)?
  - Which software representation (source code, compiler Intermediate Representation-IR or target binary code) to opt for?
  - How to find correspondences between target binary control flow graph (CFG) and high-level code CFG when:
    - compiler optimizations, even the aggressive ones, are enabled (e.g. gcc -O3)?

Software Back-annotation

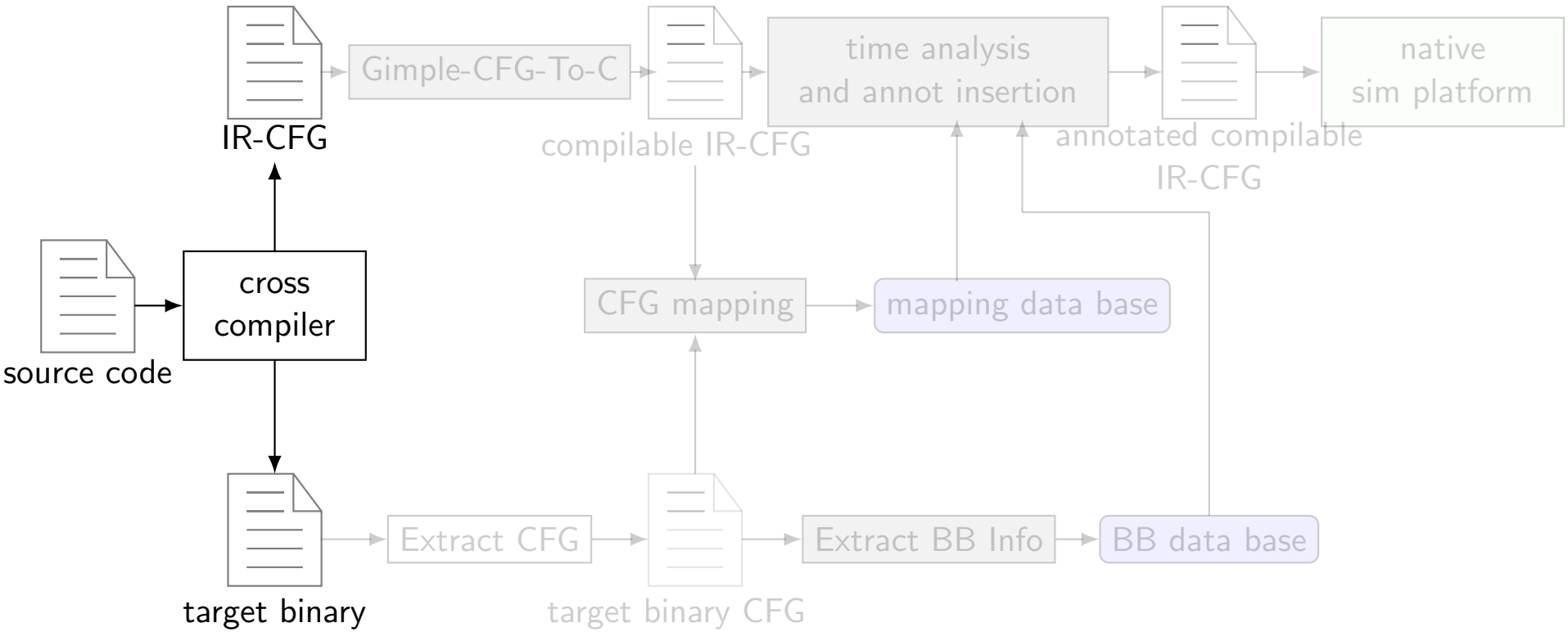
- How to compute non-functional information? (target binary analysis + modeling micro-architectural components)
- How to introduce target-specific performance metrics into the functional model (SW)?
  - Which software representation (source code, compiler Intermediate Representation-IR or target binary code) to opt for?
  - How to find correspondences between target binary control flow graph (CFG) and high-level code CFG when:
    - compiler optimizations, even the aggressive ones, are enabled (e.g. gcc -O3)?

Outline

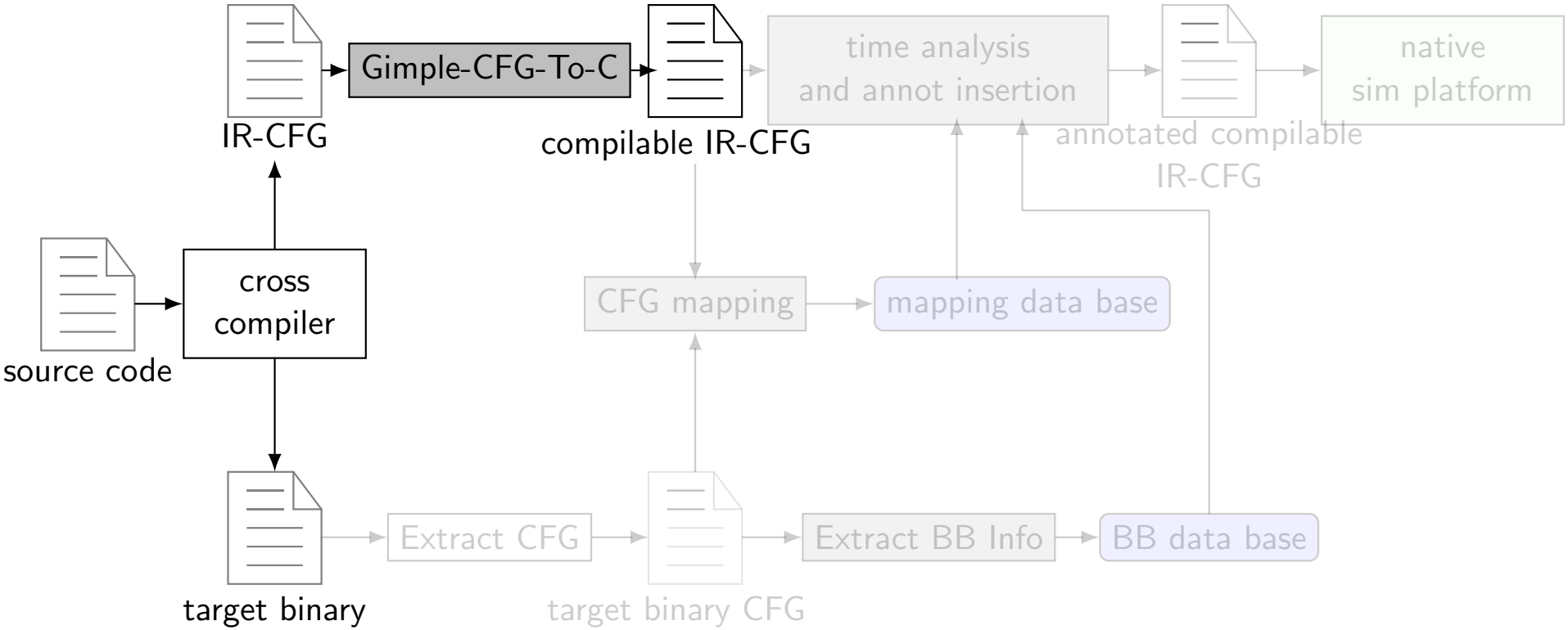
IR-Level Annotation Framework



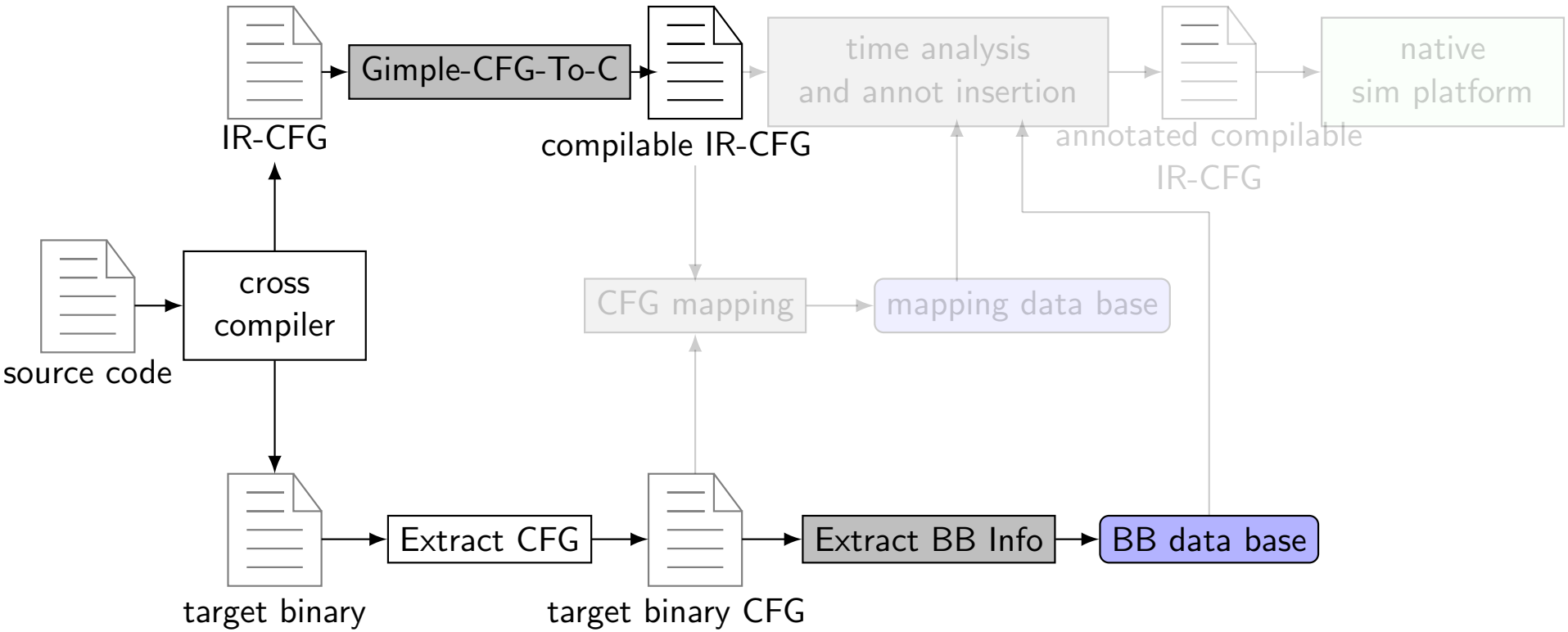
IR-Level Annotation Framework



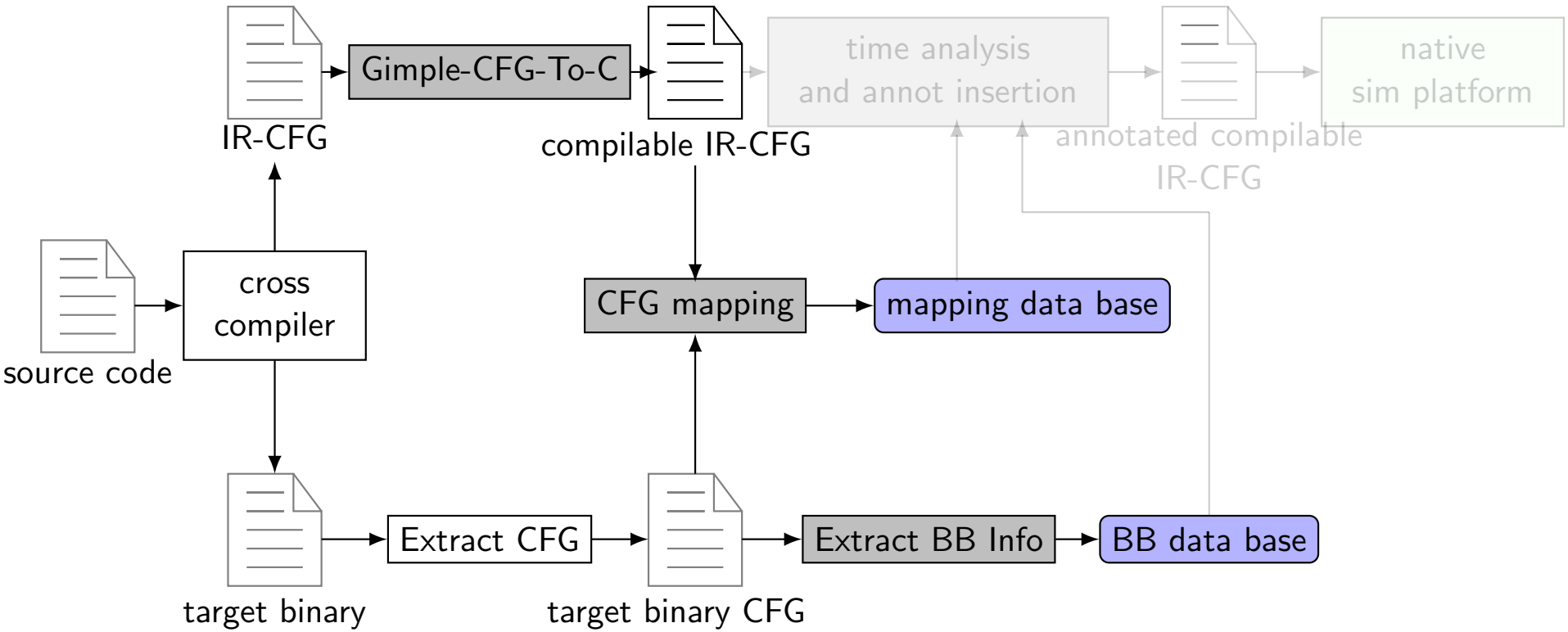
IR-Level Annotation Framework



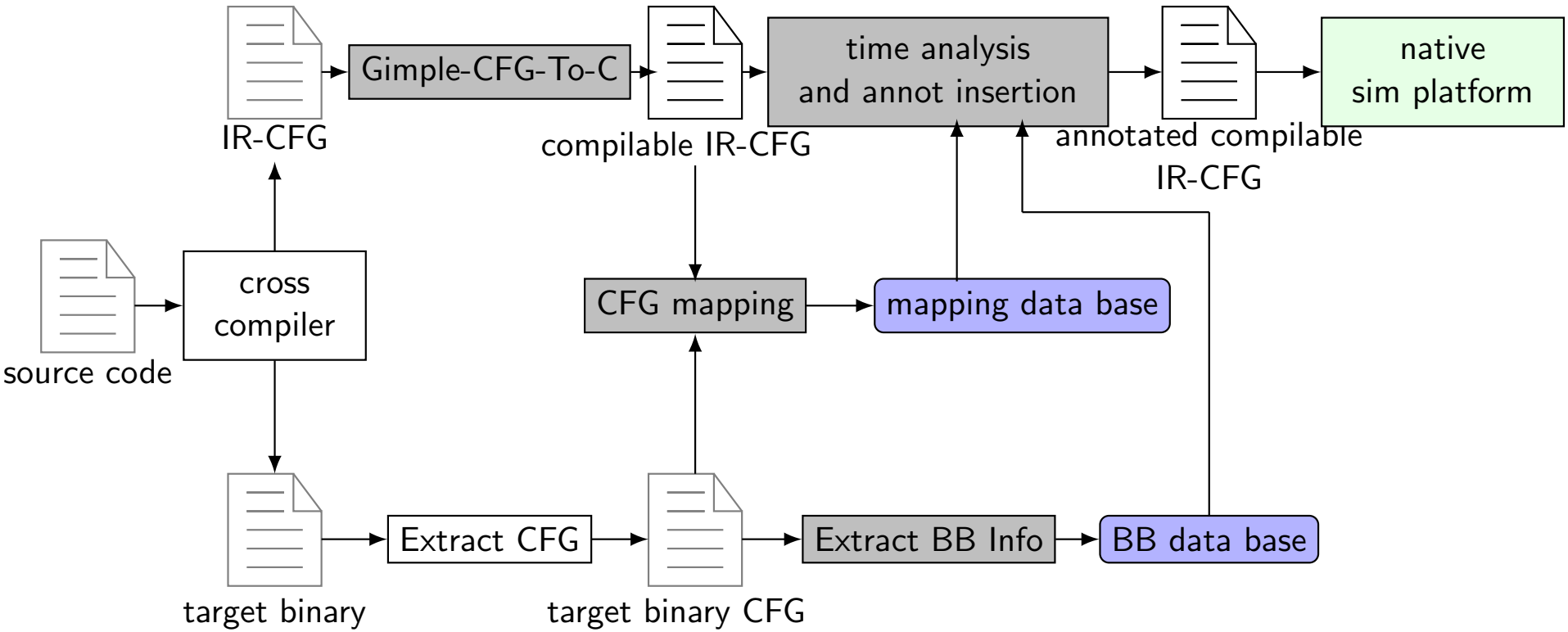
IR-Level Annotation Framework



IR-Level Annotation Framework

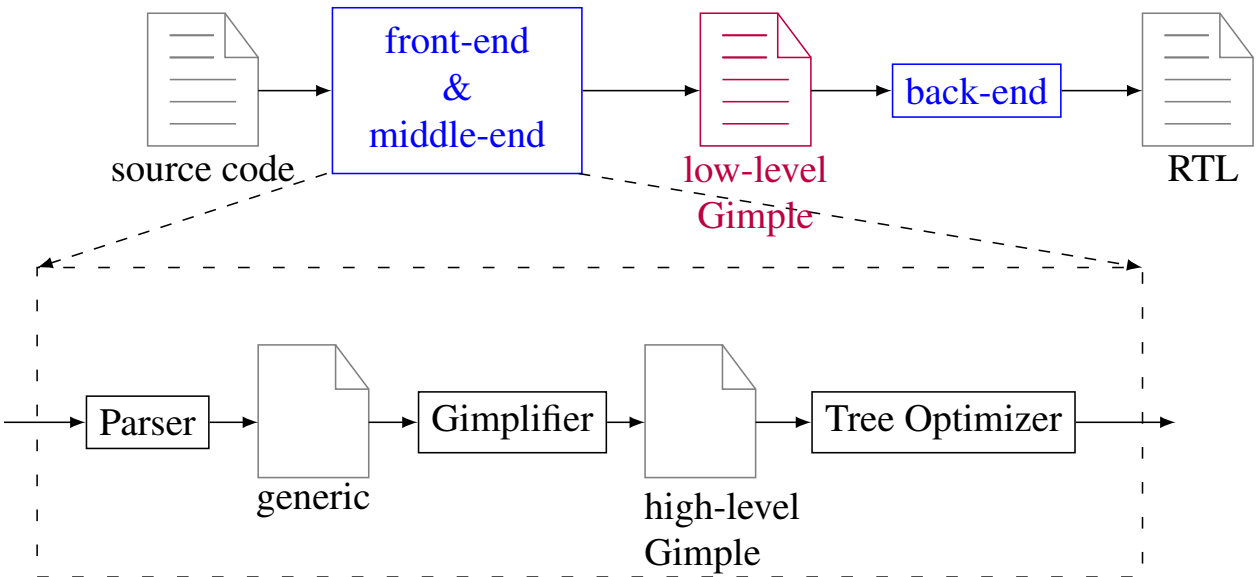


IR-Level Annotation Framework



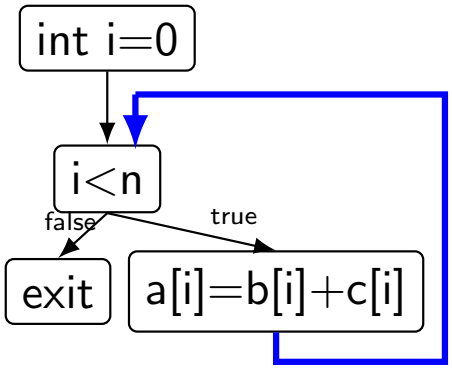
Choice of the Software Representation

- How to accurately place non-functional information into the functional model?
  - **Choice of the Intermediate Representation (IR),**
  - Accurate mapping of the functional model to the target binary code.

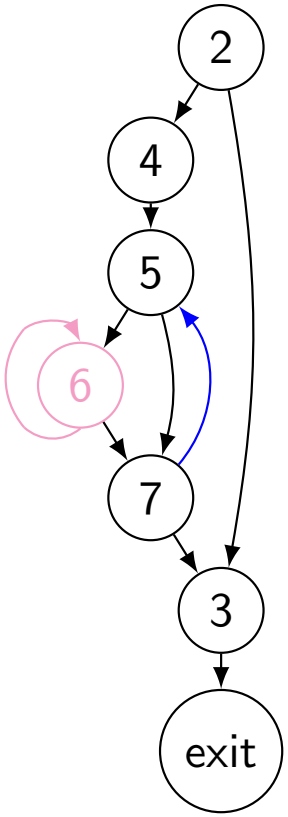


GCC's intermediate representations

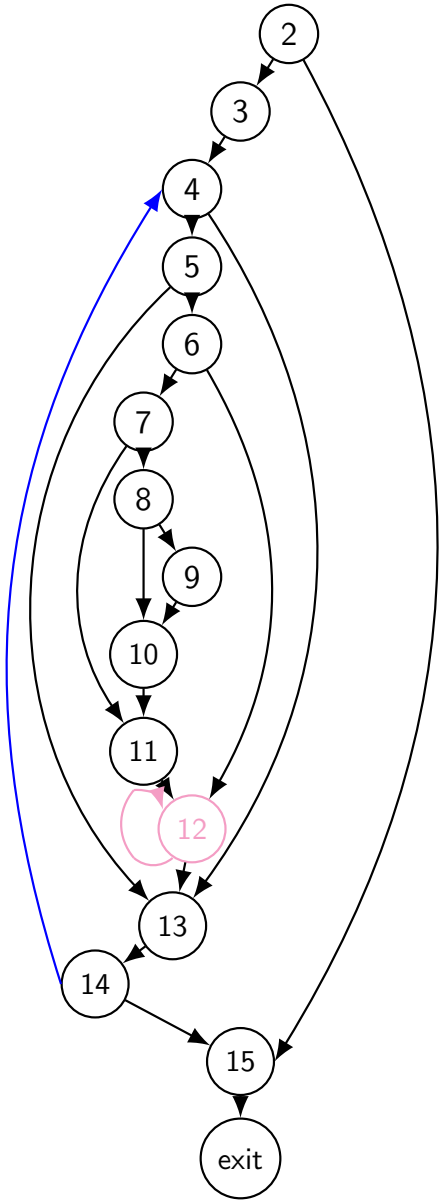
IR and binary CFGs Are Not Always Identical



Source Code CFG

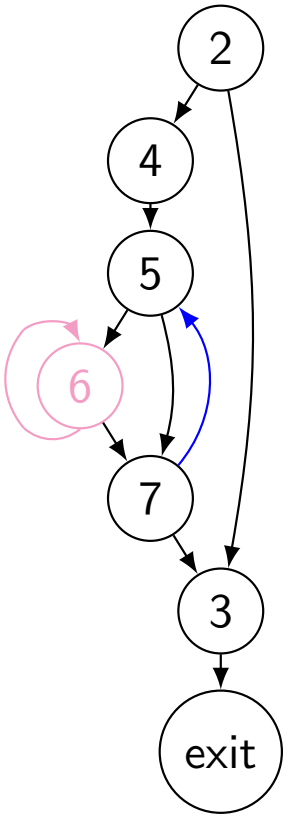


IR CFG (gcc -O3)

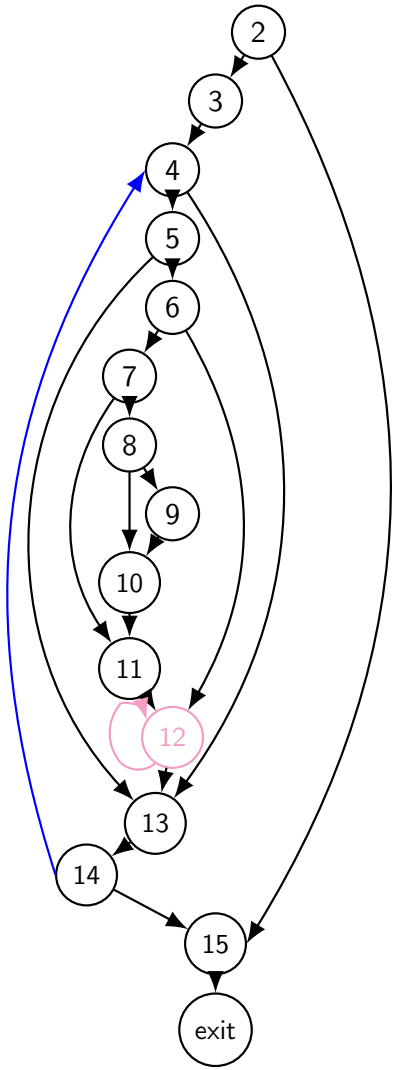


Binary CFG (gcc -O3)

Existing Mapping Approach[1]



IR CFG (gcc -O3)

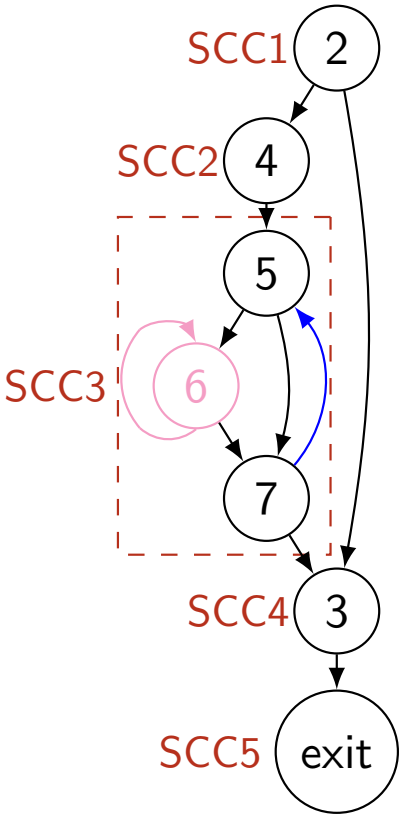


Binary CFG (gcc -O3)

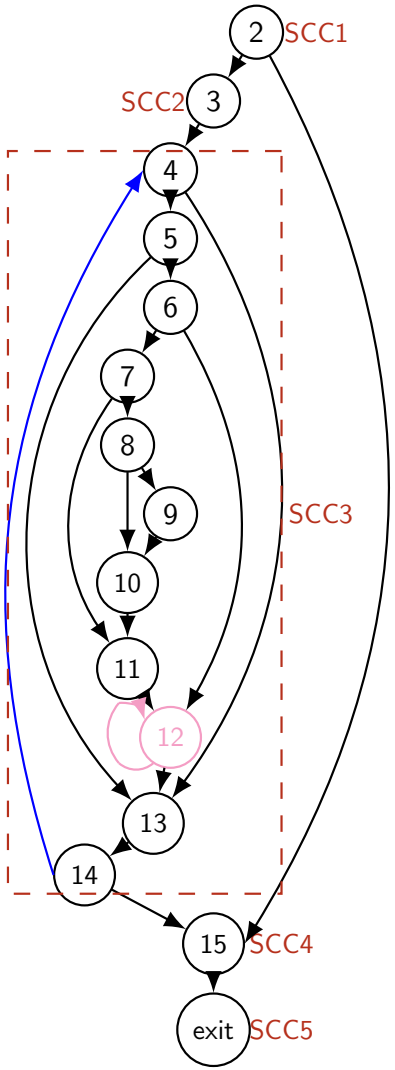
Existing mapping algorithm is efficient with O2 optimization level

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Existing Mapping Approach[1]



IR CFG (gcc -O3)

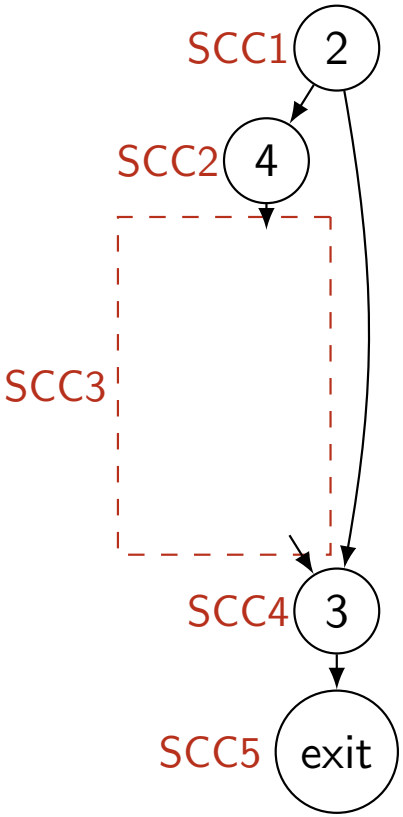


Binary CFG (gcc -O3)

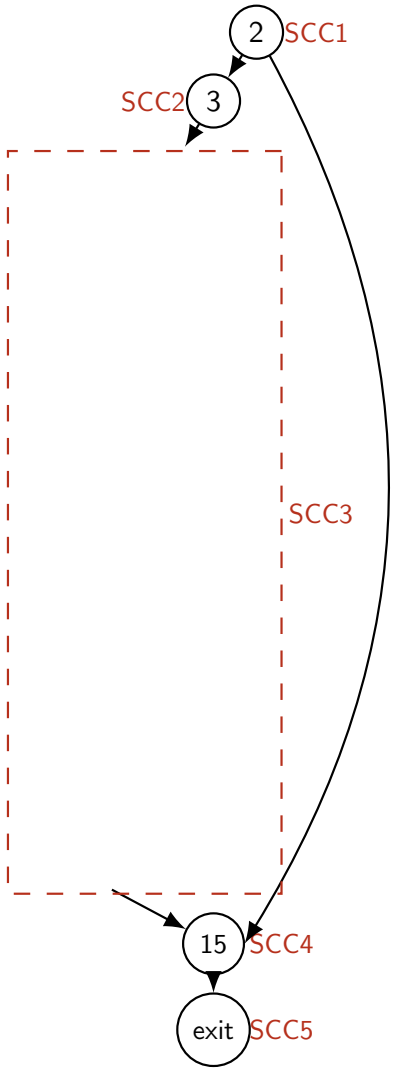
SCC: Strongly Connected Component

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Existing Mapping Approach[1]



IR CFG (gcc -O3)

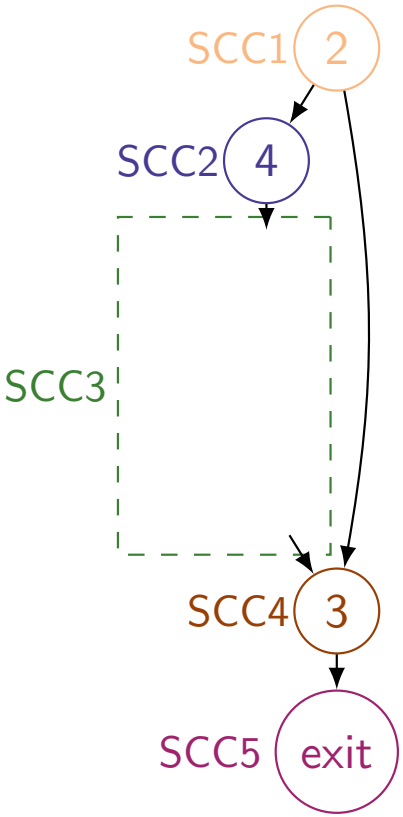


Binary CFG (gcc -O3)

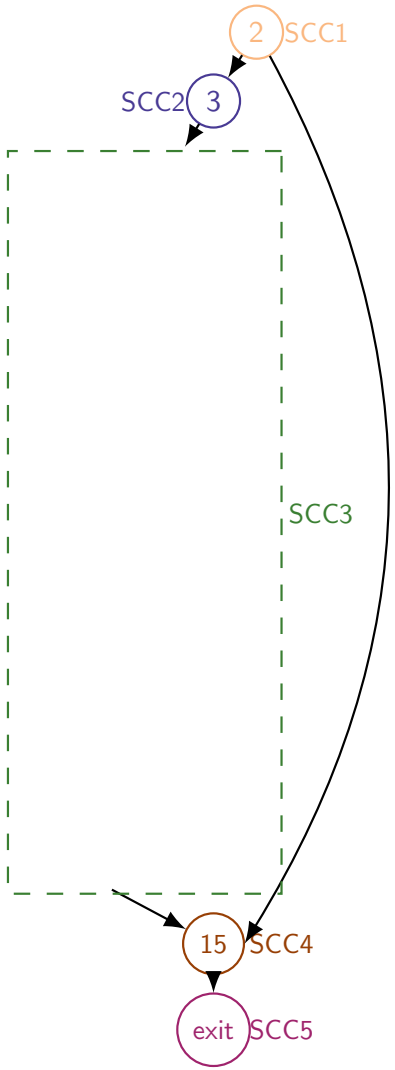
Loop block contraction

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Existing Mapping Approach[1]



IR CFG (gcc -O3)

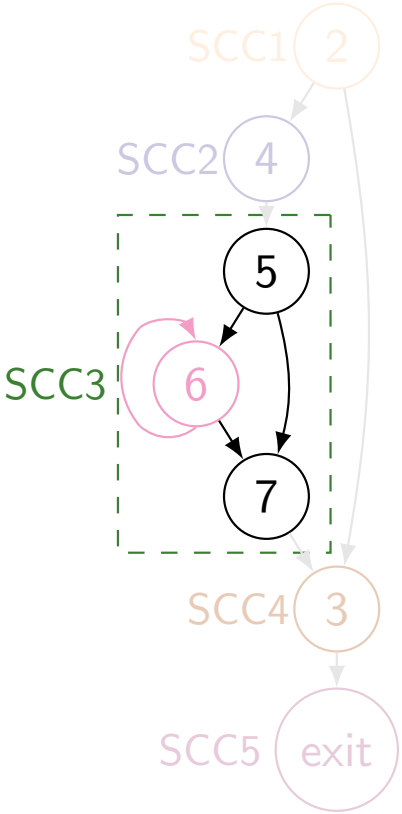


Binary CFG (gcc -O3)

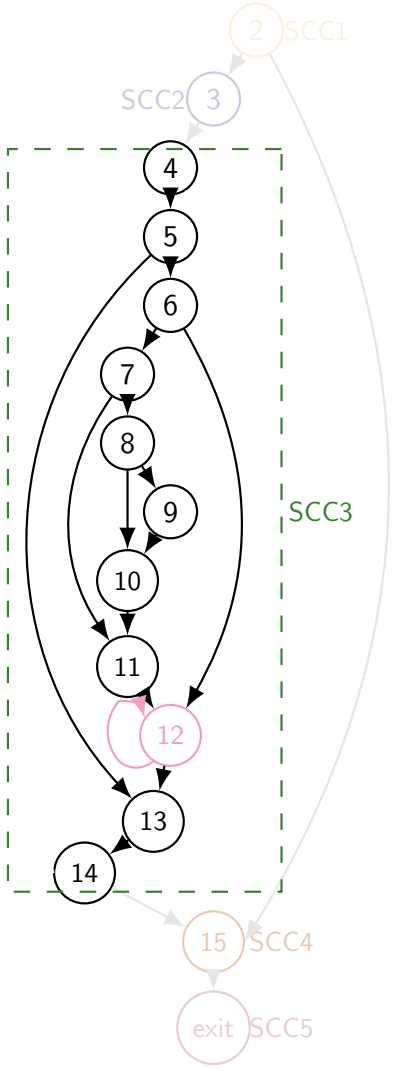
- Entry SCCs are fixed-points,
- Loop blocks are fixed-points,
- fixed-points are propagated using $PRED(SCC)$ and $SUCC(SCC)$.

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Existing Mapping Approach[1]



IR CFG (gcc -O3)

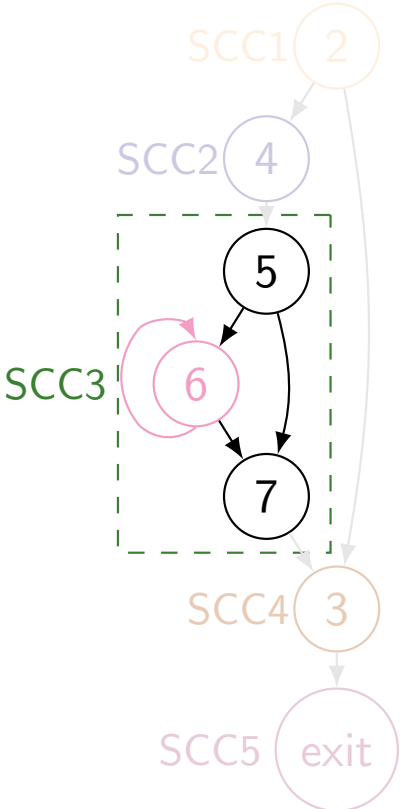


Binary CFG (gcc -O3)

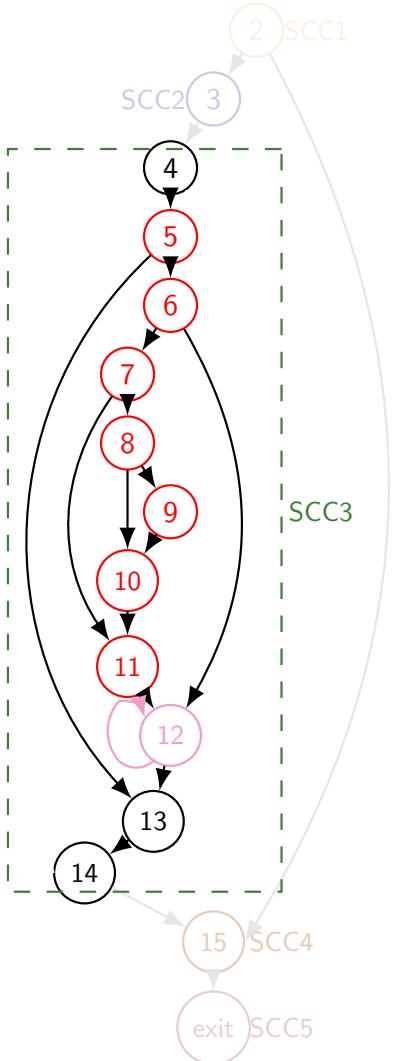
- Entry SCCs are fixed-points,
- Loop blocks are fixed-points,
- fixed-points are propagated using $PRED(SCC)$ and $SUCC(SCC)$.

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Existing Mapping Approach[1]



IR CFG (gcc -O3)

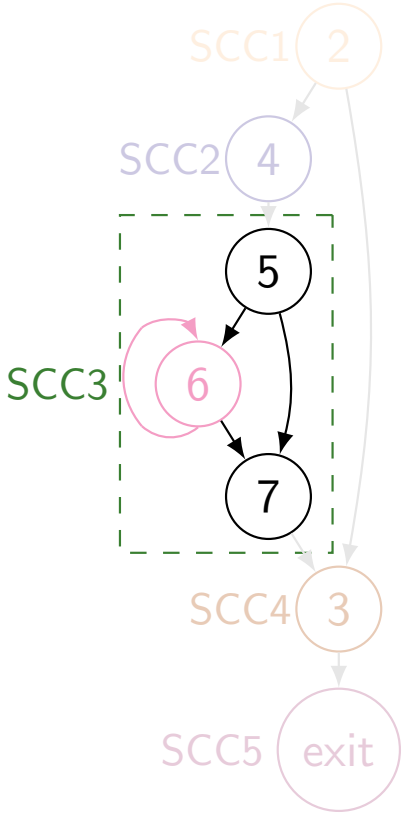


Binary CFG (gcc -O3)

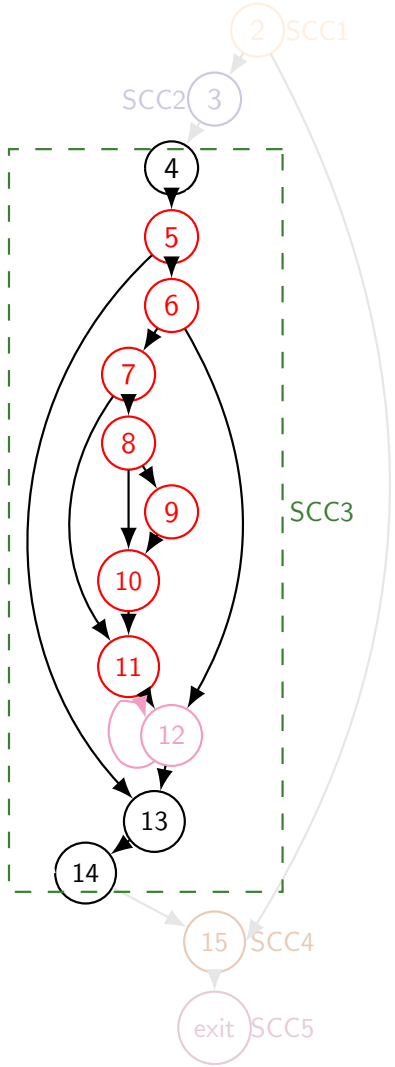
$bb5^{gcc}_{bin}$, $bb6^{gcc}_{bin}$, ..., $bb11^{gcc}_{bin}$ have no match in the IR

[1] Omayma Matoussi and Frédéric Pétrot. "Loop Aware IR-Level Annotation Framework ..." In: *ASP-DAC*. 2017.

Loop Unrolling



IR CFG (gcc -O3)



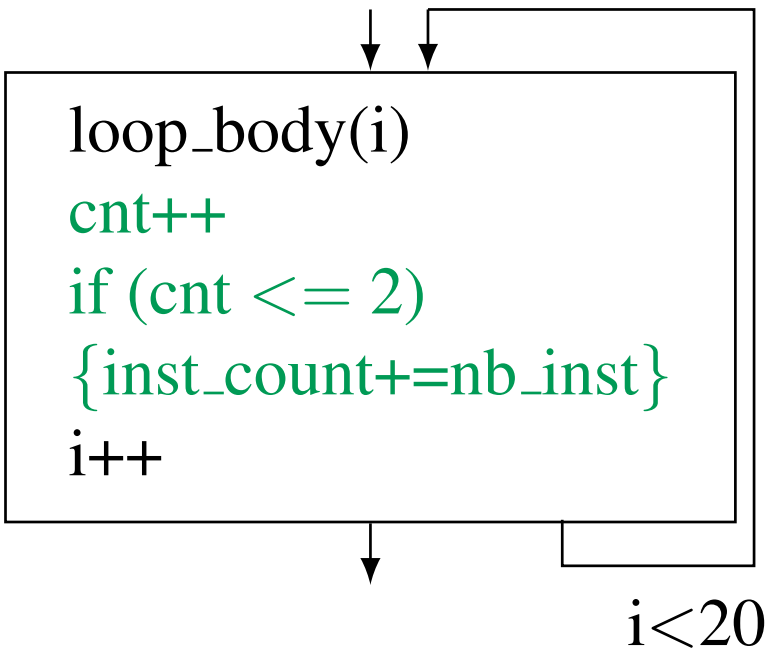
Binary CFG (gcc -O3)

Loop Unrolling replicates the loop body **UF** (Unrolling Factor) times.

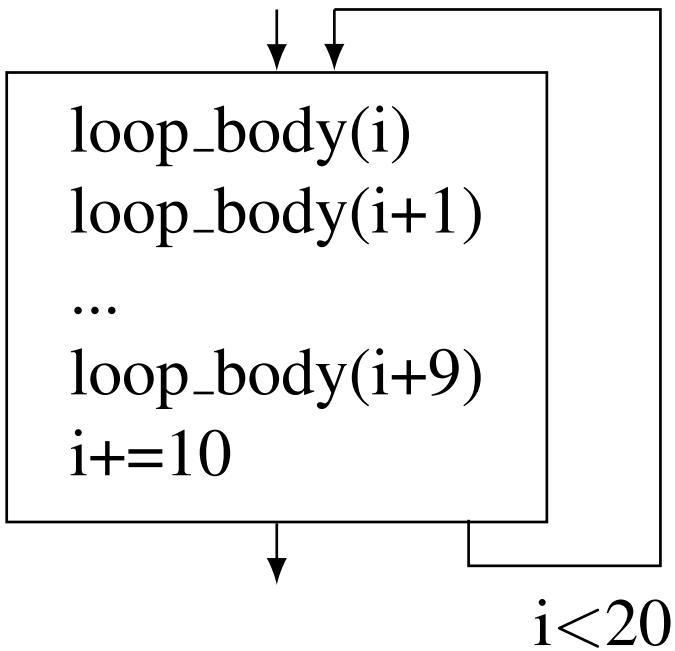
Mapping Scheme for Aggressive Compiler Optimizations

case1 :

- The loop trip count is **known** at compile time
  - The trip count is a **multiple** of (U F + 1)



(a) IR loop (max itr bound=20)

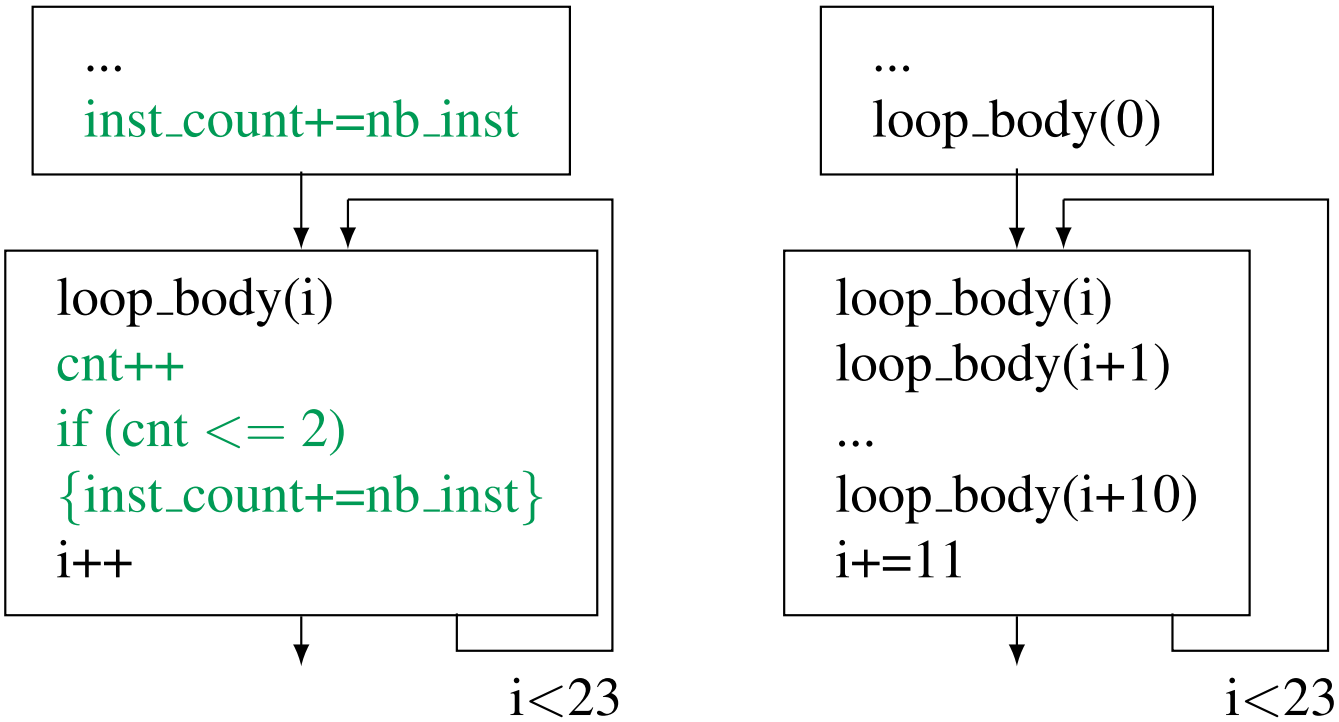


(b) Unrolled binary loop (max itr bound=2, UF=9)

Mapping Scheme for Aggressive Compiler Optimizations

case2 :

- The loop trip count is **known** at compile time
  - The trip count is **NOT a multiple** of (U F + 1)



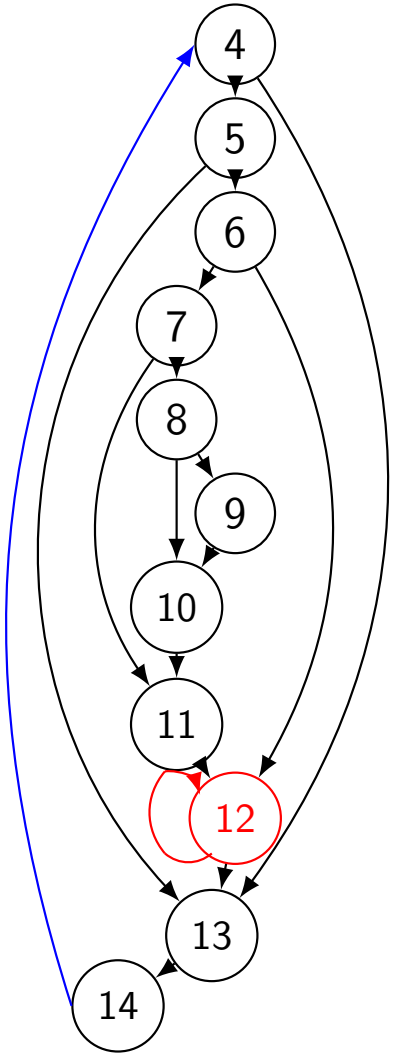
(a) IR loop (max itr bound=23)

(b) Unrolled binary loop (max itr bound=2, UF=10, first itr peeled)

Mapping Scheme for Aggressive Compiler Optimizations

case3 :

- The loop trip count is **unknown** at compile time (gcc)



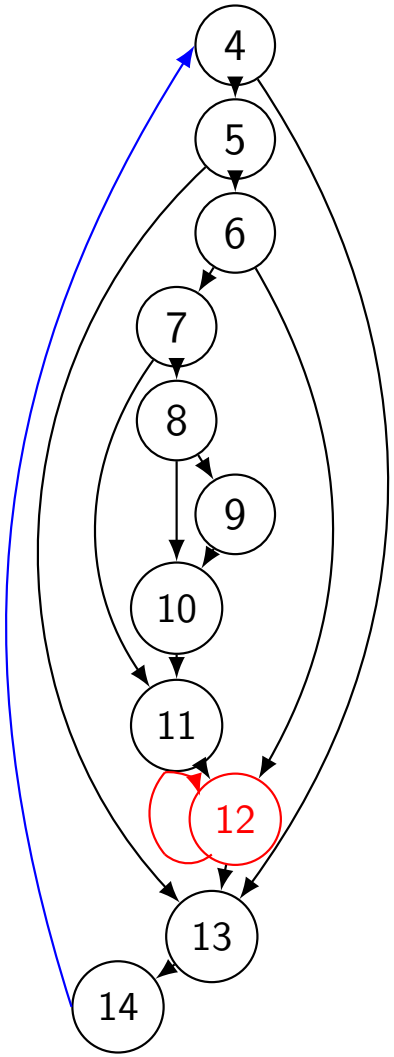
Partially Unrolled binary loop (gcc)

- Only the innermost loop is unrolled.
- GCC adds a prologue.
- Number of tests depends on the UF.

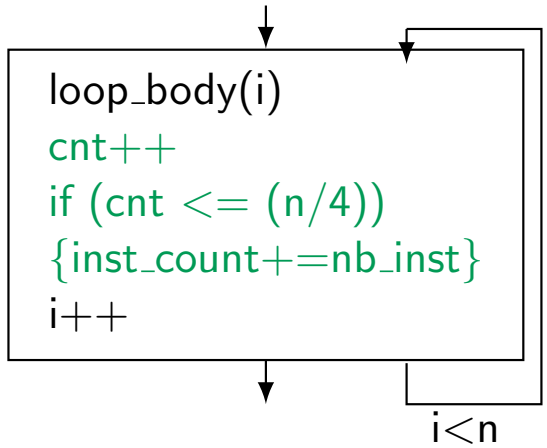
Mapping Scheme for Aggressive Compiler Optimizations

case3 :

- The loop trip count is **unknown** at compile time (gcc)



Partially Unrolled binary loop (gcc)

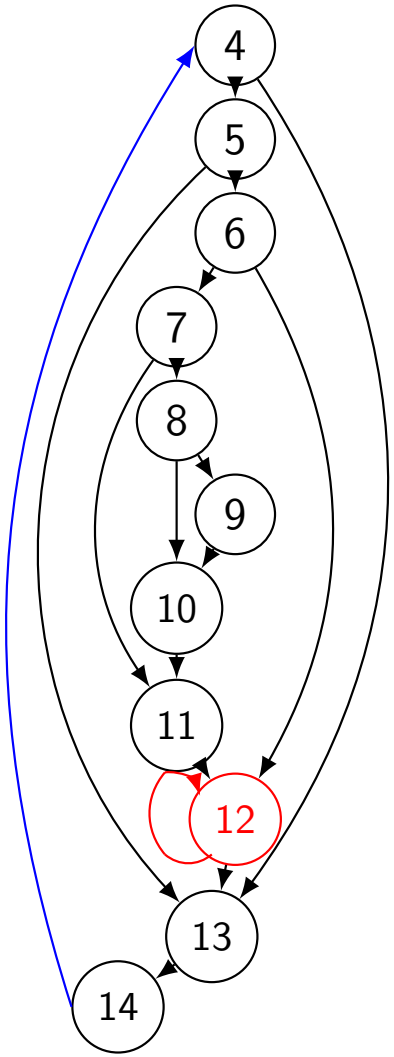


adding a prologue with if statements to the IR

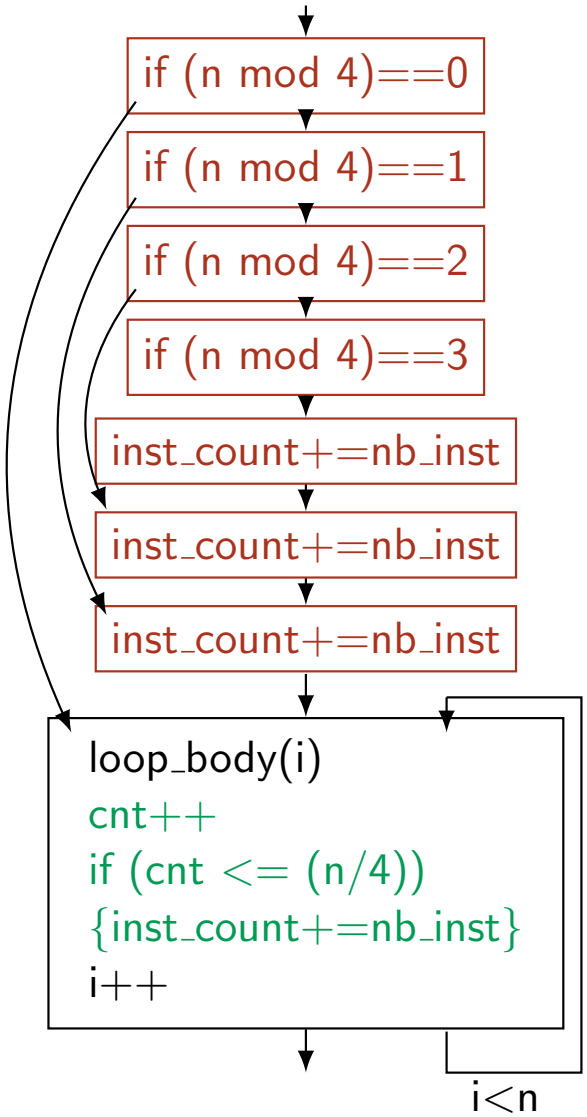
Mapping Scheme for Aggressive Compiler Optimizations

case3 :

- The loop trip count is **unknown** at compile time (gcc)

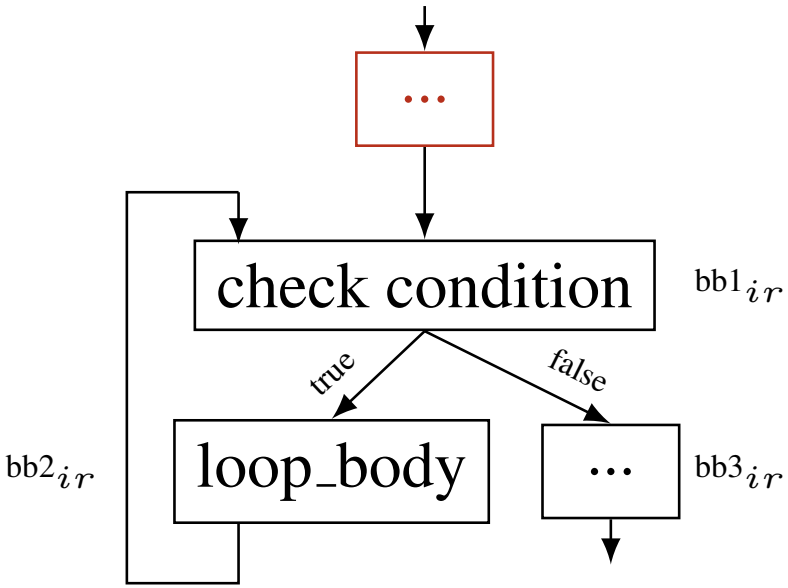


Partially Unrolled binary loop (gcc)

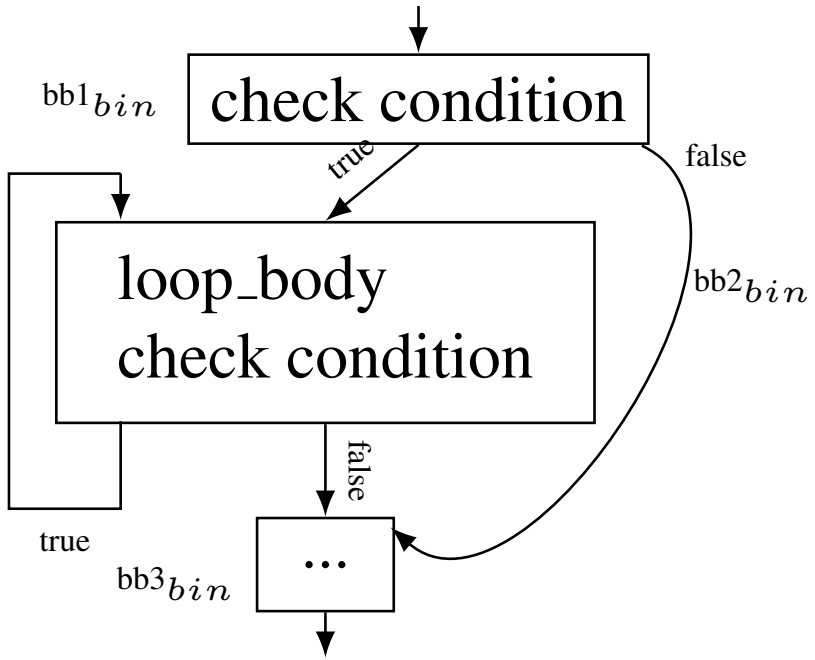


Mapping Scheme for Aggressive Compiler Optimizations

- Miscellaneous Optimizations



(a) while loop

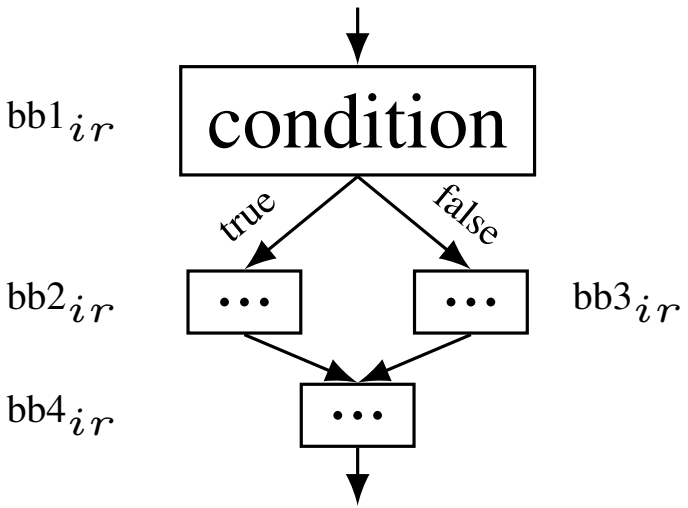


(b) do-while loop

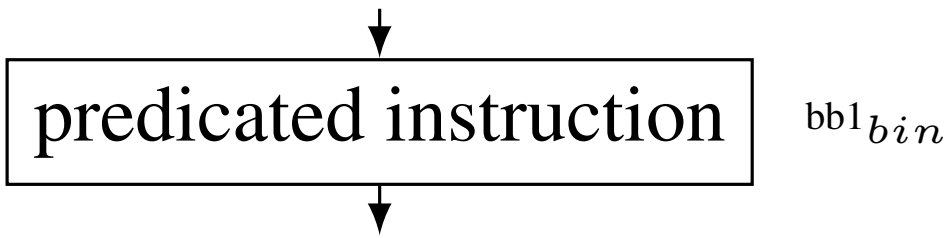
Loop Inversion

Mapping Scheme for Aggressive Compiler Optimizations

- Miscellaneous Optimizations



(a) If-then-else in the IR



(b) If-conversion in the binary

If conversion

Outline

Experimentation

- Target architecture: Kalray k1 core,
- Host Machine: Intel x86-64 core,
- Native simulation platform:
  - Based on KVM,
  - The HW components are modeled with SystemC-TLM,
- ISS provided by Kalray.

Experimentation

Table 1: A sample of the used benchmarks

Benchmark	Description
Polybench	
covar	Covariance Computation
atax	Matrix Transpose and Vector Multiplication
reg-detect	2-D Image processing
trmm	Triangular matrix-multiply
other	
matmult	1 Matrix Multiplication
bubbleSort	Bubble Sort
blowfish	Symmetric-key block cipher

Experimentation

Table 2: Comparison of the simulation time

		matmult	bubbleSort	covar	atax	reg-detect	trmm	gemver
sim_time(s)	ISS	0.624	2.863	9.006	2.020	1.396	38.086	4.208
	ILS+O3Map	0.180	0.184	0.284	0.196	0.180	0.348	0.196
	speedup_O3Map	3.47	15.56	31.71	10.31	7.76	109.44	21.47
	ILS+O2Map	0.170	0.180	0.282	0.192	0.172	0.348	0.188
	speedup_O2Map	3.67	15.91	31.94	10.52	8.12	109.44	22.38
	ILS+O2Map+	0.176	0.180	0.282	0.194	0.176	0.348	0.188
	speedup_O2Map+	3.56	15.91	31.94	10.41	7.93	109.44	22.38

$$speedup(ILS + OxMap) = \frac{sim\_time(ISS)}{sim\_time(ILS + OxMap)}.$$

Experimentation

Table 3: Comparison of the instruction count

		matmult	bubbleSort	covar	atax	reg-detect	trmm	gemver
instr_count	ISS	155993	2646028	151302	25748	9892	136033	40556
	ILS+O3Map	155993	2656128	154561	25684	10011	136321	40809
	error_O3Map	+0.0%	+0.38%	+2.15%	-0.25%	+1.2%	+0.21%	+0.62%
	ILS+O2Map	954293	10498510	902115	109985	18213	862273	176398
	error_O2Map	+512%	+297%	+496%	+327%	+84%	+534%	+335%
	ILS+O2Map+	102893	3600010	98327	14625	5741	88129	29686
	error_O2Map+	-34%	+36%	-35%	-43%	-42%	-35%	-27%

$$error(\%) = \frac{|nb\_exec\_instrs(ILS + OxMap) - nb\_exec\_instrs(ISS)|}{nb\_exec\_instrs(ISS)} \times 100.$$

Outline

Conclusion

- We proposed a mapping approach between IR and binary CFGs, when aggressive compiler optimizations (gcc -O3) are enabled.
- We modify the IR CFG without changing its functional behavior.
- Experiments show considerable speedup yet high accuracy in instruction count.

A Mapping Approach Between IR and Binary CFGs dealing with Aggressive Compiler Optimizations for Performance Estimation

Omayma Matoussi Frédéric Pétrot

TIMA Laboratory – 46 Avenue Félix Viallet, 38031, Grenoble, France

01/24/2018