

Compiler Support for WCET Analysis

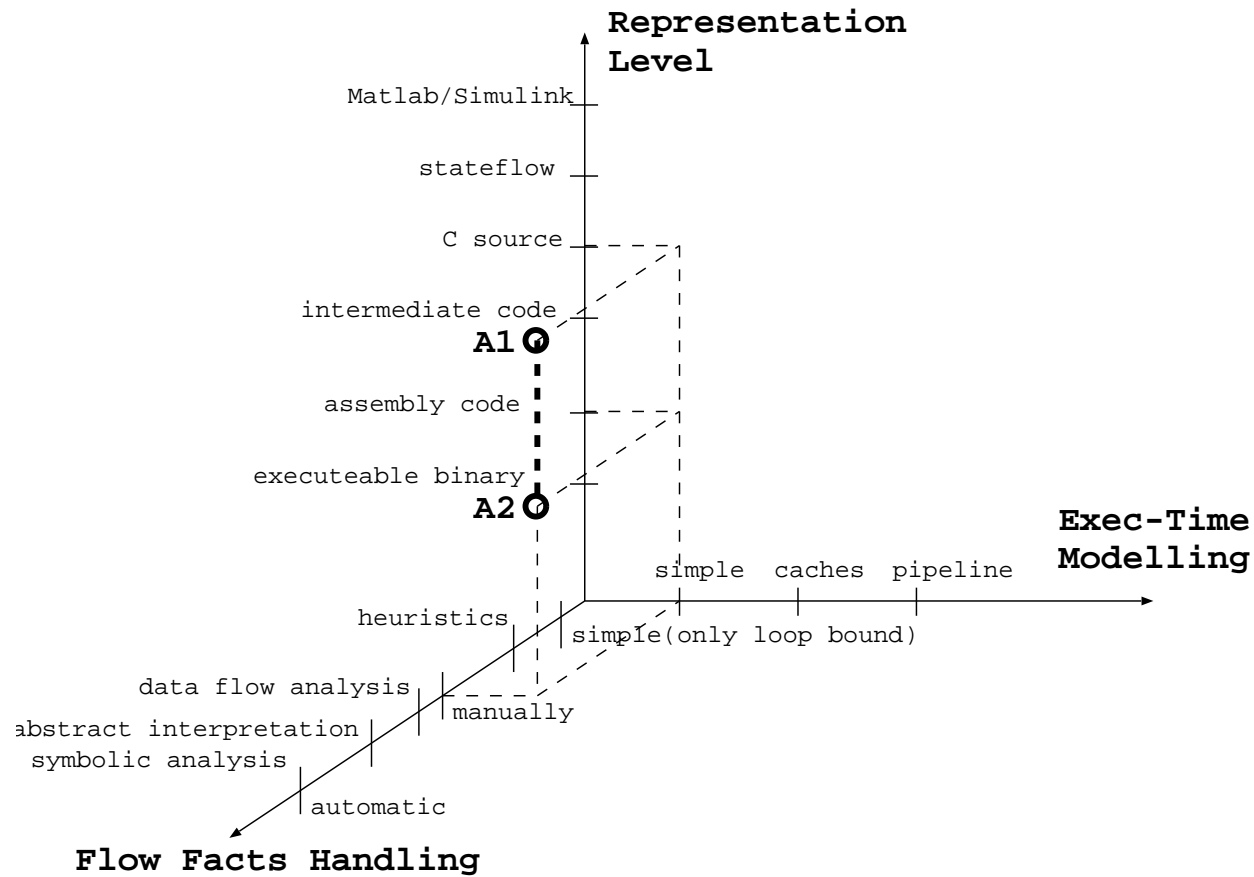
Extending Optimizing Compilation to Support Worst-Case Execution Time Analysis

Raimund Kirner
Institut für Technische Informatik
Vienna University of Technology
`raimund@vmars.tuwien.ac.at`

Problem Description

- SW for *hard real-time systems*: safe upper WCET bounds required
- SW is coded in a third-generation programming language (or even higher abstraction level in combination with code generation)
- Flow facts: program annotations to describe the possible control flow (e.g., infeasible paths, loop iteration bounds)
- HW-costs have to be minimized:
 - using an optimizing compiler
 - the WCET analysis tool must provide tight WCET bounds

WCET Analysis



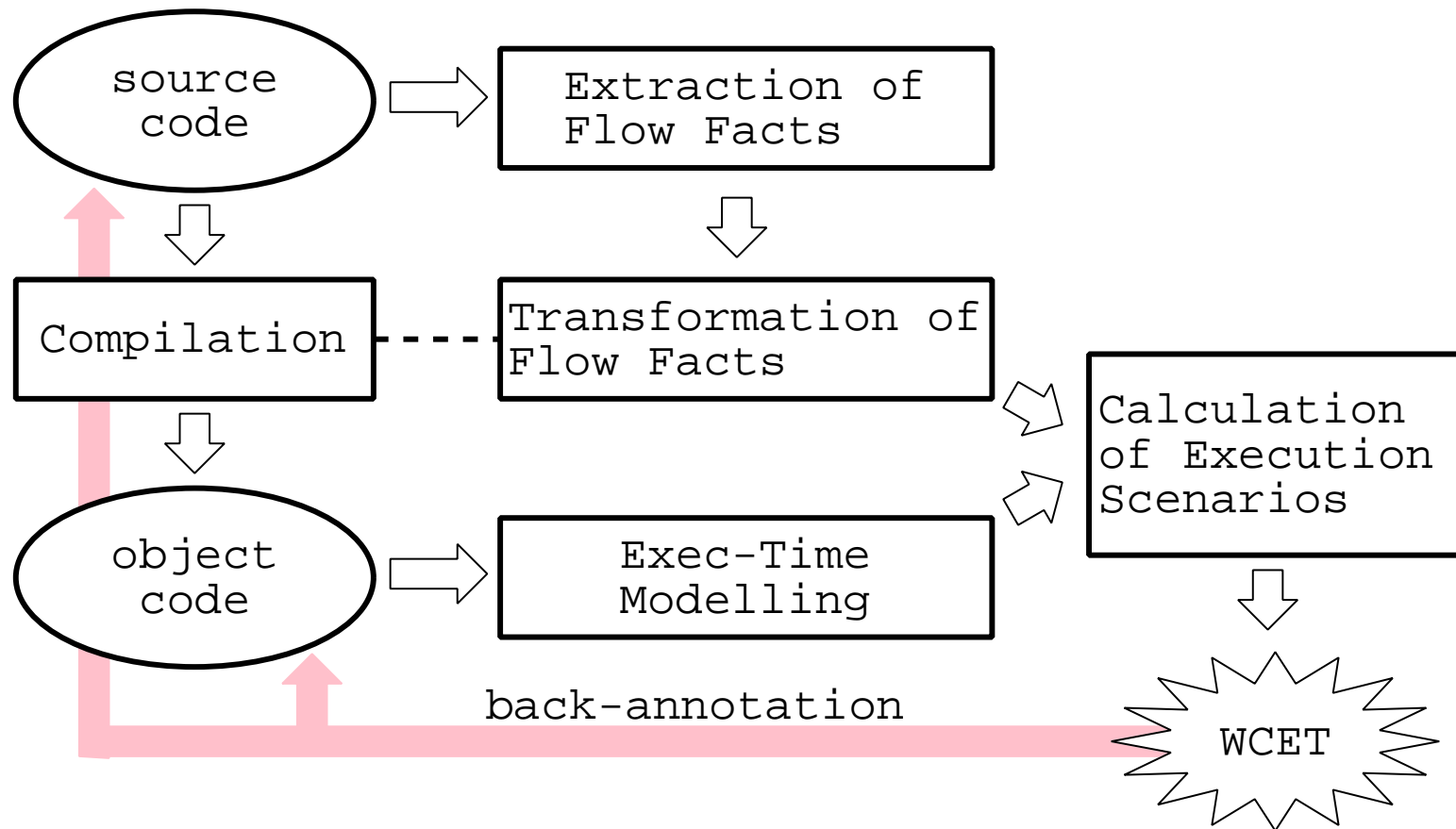
WCET Analysis, continued

- WCET analysis for object code (\rightarrow precision)
- WCET calculation based on ILP (integer linear programming)
- Information about possible program control flow required (*Flow Facts*)
- Flow facts may be extracted by semantic code analysis
- Basic flow facts given by manual code annotation \rightarrow **undecideability**

Why Compiler Support for WCET Analysis?

- Code development performed at “high” representation-level (C, C++, hierarchic program development → MATLAB/Simulink)
- Flow facts are required at object code level.
- Annotation of flow facts at object code level is error-prone, prohibits fully automatic WCET analysis.
- Annotation of flow facts preferred at source code level (more obvious, structured code).
- Transformation of flow facts from source to object code required!
- Transformation of flow facts cannot be done completely externally (ambiguous structural mapping due to code optimizations by the compiler).

A Generic Tool Chain for WCET Analysis



Abstract Co-Interpretation

- Flow facts have to be transformed according performed code optimizations.
- Number of possible program optimizations performed by compiler is huge.
- Abstract interpretation (AI) is used to simplify flow facts transformation by considering only structural changes.
- AI provides formalism to construct correct transformations.
- AI allows to construct flow facts transformations systematically by induction.

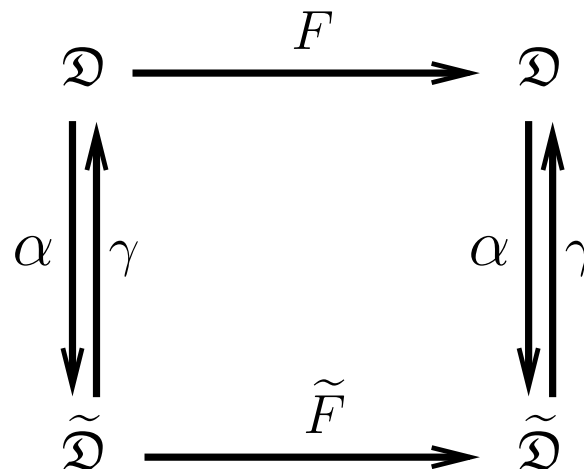
Abstract Co-Interpretation, continued

Concrete domain \mathcal{D} ... Intermediate program representation

Abstract domain $\tilde{\mathcal{D}}$... Control Flow Path (CFP)

Concrete transfer function F ... Program transformation

Abstract transfer function \tilde{F} ... Structural update



Abstract Co-Interpretation, continued

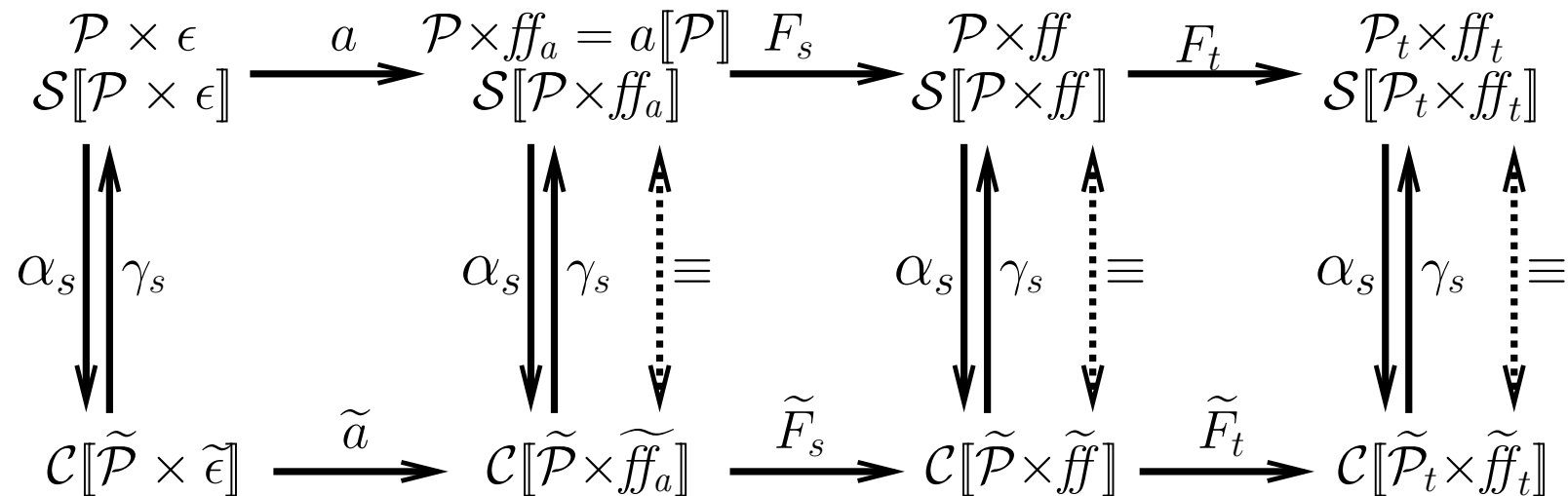
$\mathcal{P} \times ff$, \mathcal{P} ... Program, ff ... Flow Facts

$\mathcal{S}[\mathcal{P} \times ff]$... Program semantics considering ff

a ... manual program annotation

F_s ... semantic code analysis

F_t ... code optimization (transformation)



Abstract Co-Interpretation, continued

- Transfer function for program transformation: $\tilde{F}_t = \tilde{F}_{t1} \times \tilde{F}_{t2}$.
- $\tilde{F}_{t1} : \tilde{\mathcal{P}} \rightarrow \tilde{\mathcal{P}}$... structural update of program
(induced by concrete transfer function F_{t1} which represents performed code optimizations).
- $\tilde{F}_{t2} : \tilde{ff} \rightarrow \tilde{ff}$... update of flow facts
(induced by the structural CFP changes done by \tilde{F}_{t1}).
- Basic operations of \tilde{F}_{t1} :
insert, move, copy, delete, replace.
- \tilde{F}_{t2} implied from \tilde{F}_{t1} : $\tilde{F}_{t2} = \text{impl}(\tilde{F}_{t1}/\tilde{ff})$

Abstract Co-Interpretation, continued

- Considering only basic operations of \tilde{F}_{t1} does not allow to maintain WCET tightness described by the flow facts.
- Instead, \tilde{F}_{t2} have to be induced from:
 - structural CFG changes
 - changes of the graph flow
- Therefore, these basic operations of \tilde{F}_{t1} are subsumed to more expressive operations.
- Calculation of BCET (best-case execution time) is similar to WCET. Only one set of flow facts has to be maintained.
- Most complex flow facts transformations are required in case of optimizations involving loops.

The *ff* Transformation Framework

Three types of *ff* transitions:

Update of marker bindings (\xrightarrow{M}): Markers are attached to control flow edges to give them a label. With this function it is possible to distribute a marker binding from a control flow edge to any other control-flow edges.

Update of restrictions (\xrightarrow{R}): Restrictions are constraints to limit the possible control flow of a program. With this function it is possible to distribute each term of a restriction to any set of control-flow edges.

Update of Loop Flow Facts (\xrightarrow{L}): This function can create, modify, or delete iteration bounds for loops.

The *ff* Transformation Framework, continued

Handling of Code Optimizations:

- *ff* transitions have to be performed in parallel for each code transformation.
- Creation of new restrictions: (to model *ff* updates more precisely)
- Grouping of transition functions:
ff transitions are applied in parallel but they have to be grouped for each code transformation.

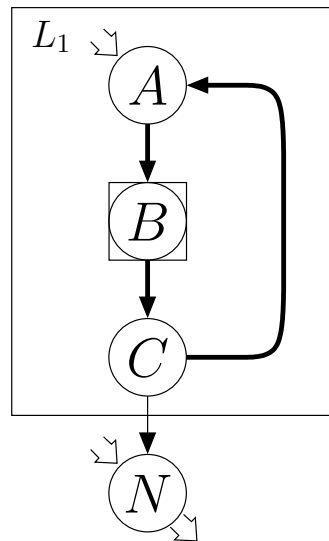
The *ff* Transformation Framework, continued

Properties of the *ff* Transformation Framework:

- Support of flexible and expressive flow facts.
- Can be applied to every code transformation performed by a compiler.
- Code transformations can be modelled precisely.
- The *ff* transformation framework is hardware independent.

Involved Data Structures

Example of Control Flow Paths, $CFP(\tilde{\mathcal{P}})$



Involved Data Structures, continued

Data Representation of $CFP(\tilde{P})$

Implementation independent representation of the static program structure:

CFPS:	$\wp(\text{STATCONN}) \times \wp(\text{LOOPSCOPE})$
STATCONN:	$P \times P \times \text{FTYPE}$
FTYPE:	$\{s, b\} \cup \text{NUM}$
LOOPSCOPE:	$\text{LID} \times \text{LID} \times P \times P$
P:	<i>... reference to basic block</i>
LID:	<i>... identifier for loop scope</i>

Involved Data Structures, continued

Examples of Flow Facts (\tilde{ff}):

$$mAB[s] \leq 5$$

$$2 \cdot mCD[s] = 3 \cdot mAB[s] + 4 \cdot mAB[b]$$

Notation:

$mAB[s]$... control flow edge from node "A" to node "B" with index "s"

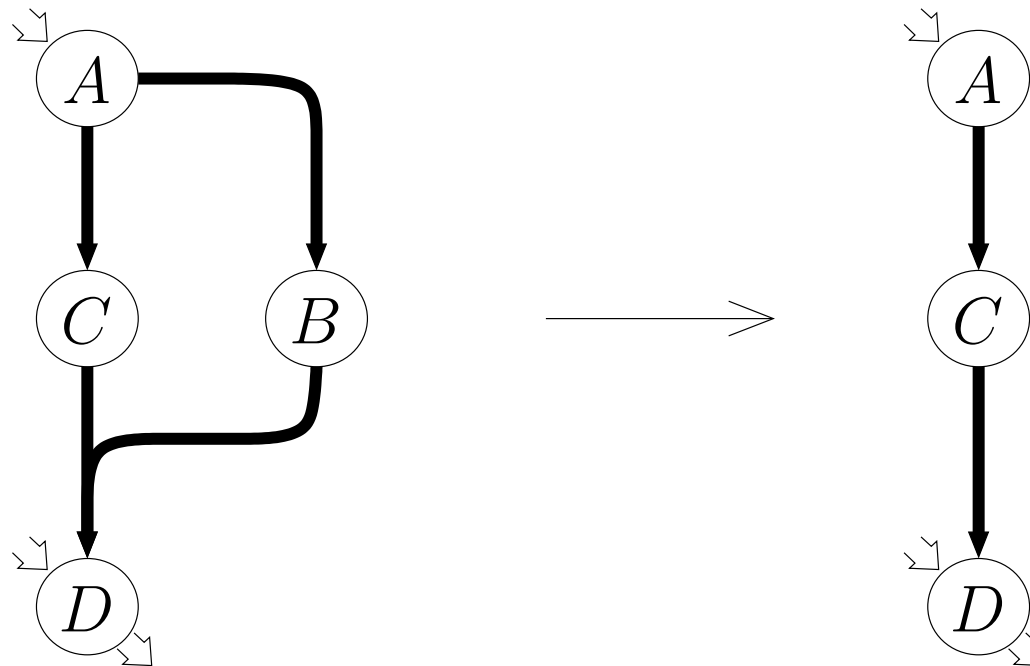
Involved Data Structures, continued

Data Representation of Flow Facts (\tilde{ff})

FF:	$\wp(\text{MB}) \times \wp(\text{RESTR}) \times \wp(\text{FFLF})$
MB:	MARKER \times STATCONN
RESTR:	$\wp(\text{TERM}) \times \text{REL} \times \wp(\text{TERM})$
TERM:	NUM \times MARKER
REL:	$\{=, <, \leq\}$
FFLF:	LID \times BOUND \times LOOPMARKER
BOUND:	NUM \times NUM
LOOPMARKER:	MARKER \times MARKER
MARKER:	<i>... reference to a marker name</i>

Example 1 (If Conversion)

Graph transformation:



Example 1 (If Conversion), continued

Induced ff update transitions:

$$mAB[b] \xrightarrow{M} mAC[s]$$

$$mBD[s] \xrightarrow{M} mCD[s]$$

$$\langle n \cdot mAC[s] \rangle \xrightarrow{R} \langle [0 \dots n] \cdot mAC[s] \rangle$$

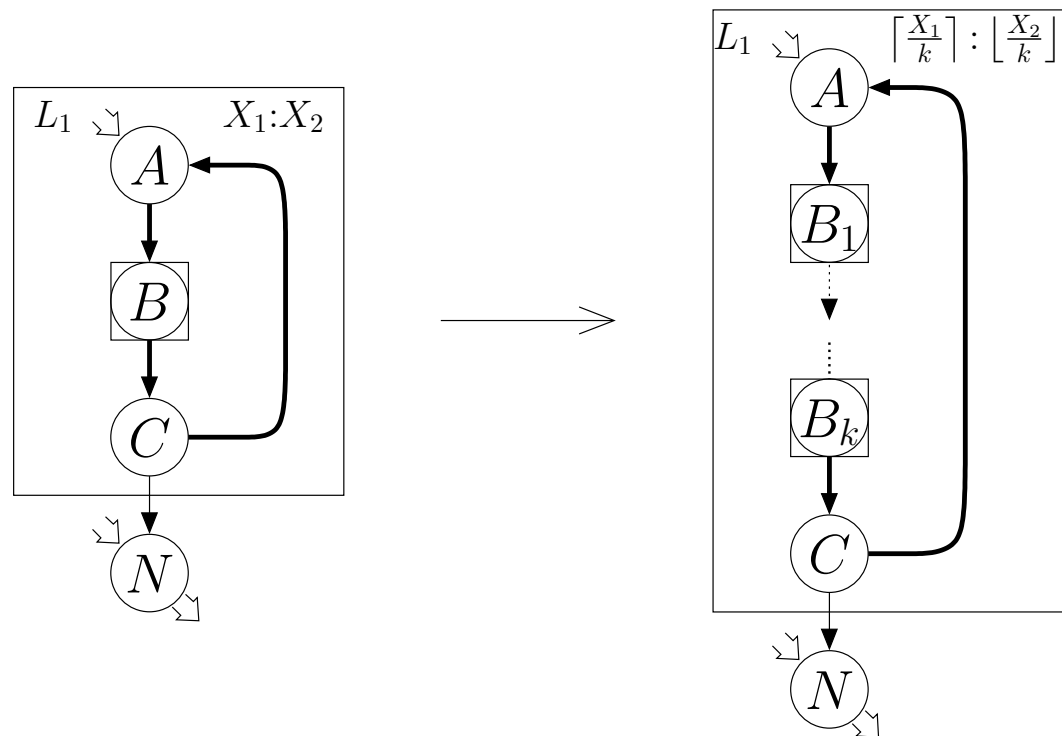
$$\langle n \cdot mAB[b] \rangle \xrightarrow{R} \langle [0 \dots n] \cdot mAC[s] \rangle$$

$$\langle n \cdot mCD[s] \rangle \xrightarrow{R} \langle [0 \dots n] \cdot mCD[s] \rangle$$

$$\langle n \cdot mBD[s] \rangle \xrightarrow{R} \langle [0 \dots n] \cdot mCD[s] \rangle$$

Example 2 (Loop Unrolling)

Graph transformation:



Example 2 (Loop Unrolling), continued

Induced *ff* update transitions:

$$mAB[s] \xrightarrow{M} mAB_1[s]$$

$$M_i(B) \xrightarrow{M} M_i(B_1), M_i(B_2), \dots, M_i(B_k)$$

$$mBC[s] \xrightarrow{M} mB_kC[s]$$

$$L_1\langle X_1, X_2 \rangle \xrightarrow{L} L_1\left\langle \left[\frac{X_1}{k} \right] : \left[\frac{X_2}{k} \right] \right\rangle$$

Example 2 (Loop Unrolling), continued

$$\langle n \cdot LMB(L_1) \rangle \xrightarrow{R} \langle n \cdot k \cdot LMB(L_1) \rangle$$

$$\langle n \cdot mAB[s] \rangle \xrightarrow{R} \langle n \cdot k \cdot mAB_1[s] \rangle$$

$$\langle n \cdot M_i(B) \rangle \xrightarrow{R} \langle n \cdot M_i(B_1) + \dots + n \cdot M_i(B_k) \rangle$$

$$\langle n \cdot mBC[s] \rangle \xrightarrow{R} \langle n \cdot k \cdot mB_k C[s] \rangle$$

$$\langle n \cdot mCA[b] \rangle \xrightarrow{R} \left\langle \left[n \frac{X_1 - 1}{\left\lceil \frac{X_1}{k} \right\rceil - 1} \dots n \frac{X_2 - 1}{\left\lceil \frac{X_2}{k} \right\rceil - 1} \right] \cdot mCA[b] \right\rangle$$

Experiments

- Runtime measurements to compare results.
- Experiments with five algorithms.
- Deviation at most 3%; typically less than 1%.
- Results from *Bubble Sort*:

<i>Method</i>	<i>Code optimizations</i>				
	<i>unoptimized</i>		<i>optimized</i>		<i>improvement</i>
	<i>cycles</i>	<i>+%</i>	<i>cycles</i>	<i>+%</i>	
static,basic	198 741	91.08	57 816	89.65	70.91
static,flow	104 625	0.59	30 492	0.02	70.86
measurement	104 007	–	30 486	–	70.69

Conclusion

- This framework enables calculation of tight WCET bounds from optimized code.
- Transformation of flow facts is required to perform WCET analysis at object code.
- Support of flow facts update required from the compiler.
- *Abstract Co-Interpretation*: formalism to design correct and precise flow fact transformations.
- Code optimizations are abstracted to structural program updates.
- Transformation of flow facts is induced from abstract structure update.
- Can be applied to every code transformation.

Outlook

The presented ff transformation framework allows to model each code transformation in a safe and accurate way, but

- Need of ff annotations should be minimized:
 - advanced semantic code analysis techniques required (e.g., abstract interpretation)
 - hard real-time code typically with simple structure \rightarrow use of more predictable programming paradigms.
- *abstract co-interpretation* is a generic concept \rightarrow application to other meta-information transformations.