Worst Case Execution Time (WCET) estimation through Abstract Interpretation in the presence of Data Caches

Y.N. Srikant

Department of Computer Science and Automation
Indian Institute of Science
Bangalore

NPTEL Course on Compiler Design

Outline

- Introduction
- Four subproblems
  - Address analysis
  - Cache analysis
  - Access Sequencing
  - Worstcase Path Analysis
- Experimental setup
- Conclusions

Introduction

WCET Estimation - 1

- Estimation of worst case execution time of programs
  - extremely important in the context of real time systems where
  - the correctness of the system depends on
    - the computations performed and
    - the timing of such computations

WCET Estimation - 2

- For task scheduling on such systems
  - it is necessary to know whether the task can execute to completion within a predetermined time interval
- Given a program and a target architecture the WCET problem is
  - to estimate a bound on the maximum execution time taken by the program for any input data set

WCET Estimation - 3

- A simple approach
  - to assume worst case latency for every instruction
  - determine the maximum execution time of each basic block
  - solve an integer linear program for maximizing the execution time along any path, subject to structural constraints.
- This approach may over-estimate theWCET by a large amount
  - it fails to recognize the presence of performance enhancing features such as caches and pipelines in the architecture

WCET Estimation - 4

- In the context of hard real time systems
  - WCET estimate of a program must be safe
  - estimate cannot be exceeded by the actual execution time for any input data set
  - simultaneously, estimate must be tight to reduce resource allocation costs
- Safety may be relaxed in the case of soft real-time systems where
  - deadlines may occasionally be missed without having a significant impact on the quality of service offered

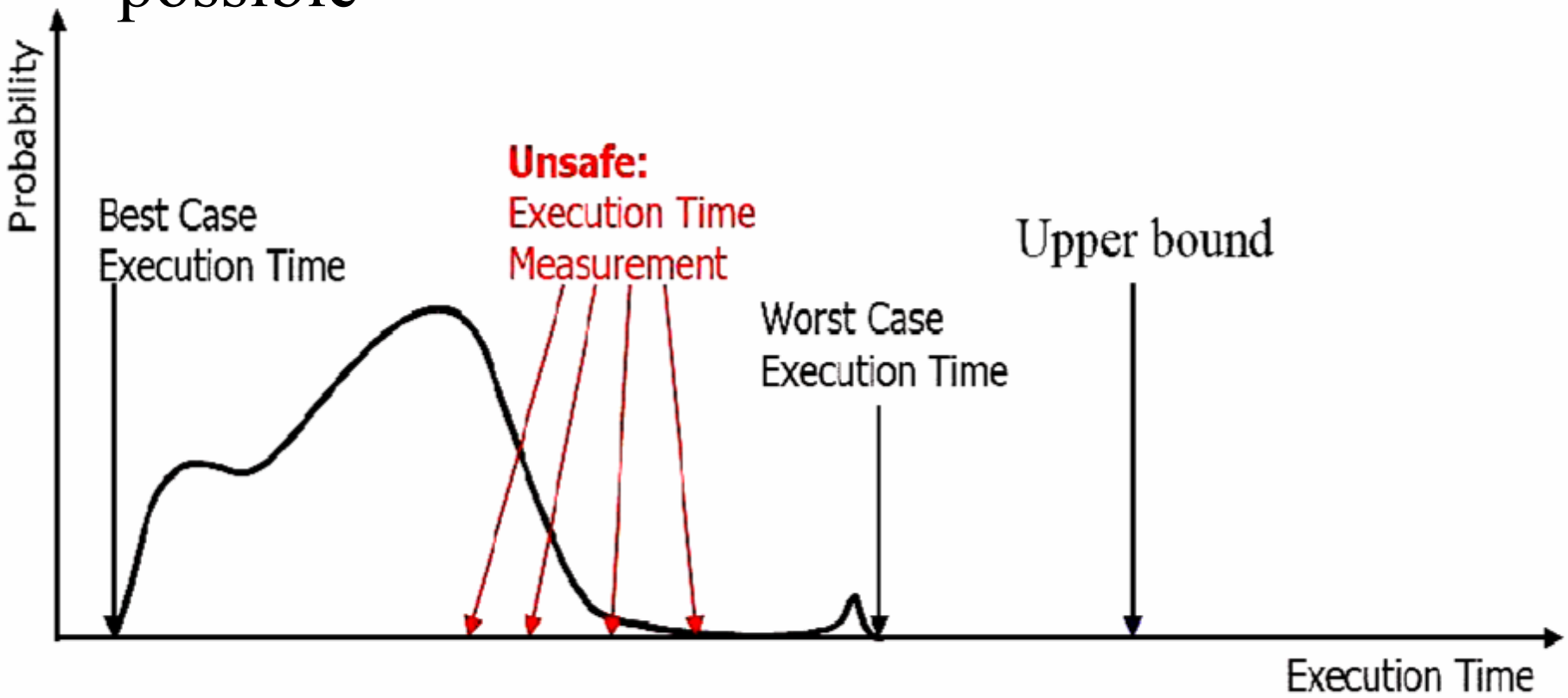
Data cache effect on WCET

Program Name	WCET (cycles)	
	Simulation	All-Miss
bsort100	81091 (× 2)	315897
cnt	4410	9895
edn_fir	52990	103133
edn_fir_no_red_ld	40328	73114
edn_iir	2274	5032
edn_latsynth	3281	5875
edn_mac	3139	6104
jfdctint	2275	3481
matmult	147413	298022

- Configuration → 4 way, 32 byte blocks, 256 sets
- Latency (cycles) → hit: 1, rd miss: 6, write miss: 4

WCET estimates

- WCET estimates must be *safe* and as *tight* as possible



Existing Art for WCET (with Dcache)

- Linear algebra based
  - Cache Miss Equations
  - Presburger Arithmetic
- Abstract Interpretation based
  - MUST analysis
- Data flow based
  - Static cache simulation
- Simulation based

Four Subproblems

- Address analysis
  - Abstract Interpretation
- Cache analysis
  - Abstract Interpretation
- Access Sequencing
  - Partial unrolling (physical and virtual)
- Worstcase Path Analysis
  - ILP formulation

Subproblem 1: Address Analysis

- Objective
  - To compute a safe approximation of the set of memory locations that can be accessed by any memory reference
- A special case of general executable analysis

Executable Analysis - Applications

- Detecting malicious content
- Algorithm learning
- Code comparison
- Timing analysis
- Cross platform porting
- Source code recovery
- Verification

Some Issues

- Absence of type information
- Difficult to separate address generation and data computations
- Compiler transformations might have changed apparent code structure
- Difficult to reverse-map registers to source variables

Traditional Analysis

- Static objects tracked
  - registers
  - statically known memory partitions
    - absolute offsets
    - stack operations
    - all locations within a partition are tracked collectively

Traditional Analysis

- Memory partitions are determined by scanning the global data section and program code for numeric offsets and stack operations.
- Simultaneous numeric and pointer analyses
- All computations are tracked
- Abstractions for the computations are used

Abstract Interpretation

- Define
  - an abstract domain
  - operations on the elements of that domain
    - must be consistent with the concrete execution semantics
- At any point, the set of abstract values is an over-approximation of the possible set of concrete values

Abstract Interpretation - An Example

- A language with *integers and* *
  - $e ::= int \mid e*e$
- Concrete Semantics
  - $\mu : Exp \rightarrow Z$
    - $\mu(i) = int.value$
    - $\mu(e_1*e_2) = \mu(e_1)*\mu(e_2)$

An Abstract Semantics

- Compute only sign of the result
  - $\sigma : Exp \rightarrow \{+, -, 0\}$
  - $\sigma(i) = \begin{cases} +, \text{ if } i > 0 \\ 0, \text{ if } i = 0 \\ -, \text{ if } i < 0 \end{cases}$
  - $\sigma(e_1 * e_2) = \sigma(e_1) \square \sigma(e_2)$

□	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

Abstract and Concrete Values

- Associate each abstract value with the set of concrete values it represents

$\gamma : \{+, -, 0\} \rightarrow 2^Z$

$\gamma(+) = \{i \mid i > 0\}$

$\gamma(0) = \{0\}$

$\gamma(-) = \{i \mid i < 0\}$

- We need to add $\top$ (top) and $\bot$ (bottom) elements to the set of abstract values
- Our abstract domain is now a *lattice*
- We can now map other operations such as +, -, and / to suitable operations on the abstract domain

Concretization Function

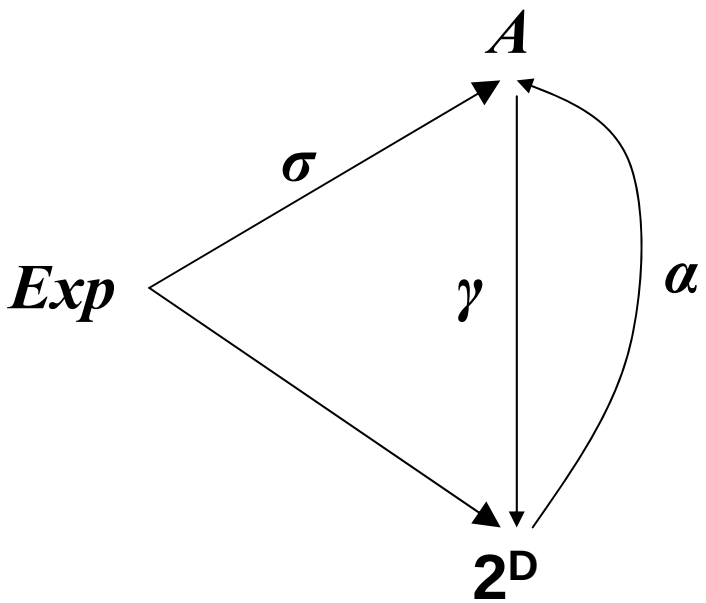
- Mapping from abstract values to sets of concrete values (γ: A → 2^D)

$$\mu(e) \ \varepsilon \ \gamma(\sigma(e))$$

μ : *Exp* → *S*, *S* ε 2^D

D: Concrete domain

A: Abstract domain



Abstraction Function

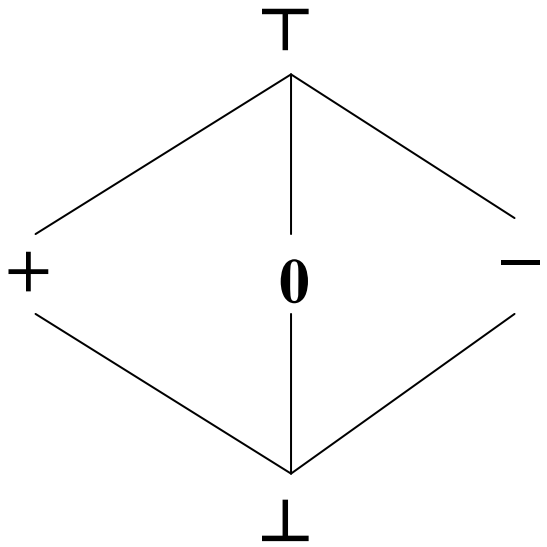
- Mapping from concrete values to abstract values
  - The dual of concretization
  - The smallest value of A that is the abstraction of a set of concrete values

$\alpha : 2^Z \rightarrow A$

$\alpha(S) = lub(\{ - \mid i < 0 \Lambda\ i\ \varepsilon\ S\},$

$\{ 0 \mid 0\ \varepsilon\ S\},$

$\{ + \mid i > 0 \Lambda\ i\ \varepsilon\ S\})$



α({24,45,3}) = +
α({-2,-87,-123}) = -
α({0}) = 0
α({-5, 2}) = ⊤

Abstract Interpretation

- Consists of
  - An abstract domain A, and a concrete domain D
  - An abstraction function α, and a concretization function γ, forming a *Galois Connection* (or *insertion)*
  - A Sound abstract semantic function σ
    - approximates standard semantics

Abstract Domains

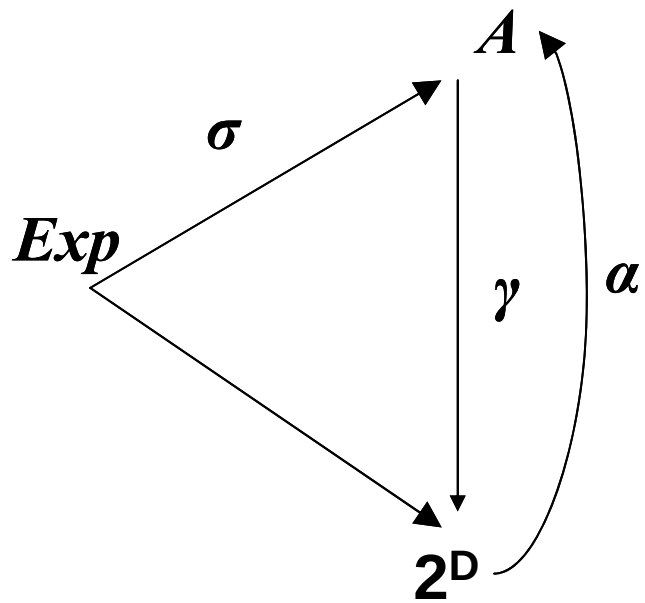
- The abstract domains can be thought of as dividing the concrete domain into subsets (not disjoint)
- The abstraction function maps a subset of the concrete domain to the smallest abstract value
- The concretization function maps abstract values to sets of concrete values

Galois Connection

- **This diagram must commute**
- $id \leq \gamma \,.\alpha$
  - **for all $x \in 2^D$, x is a subset of $\gamma(\alpha(x))$**
- $id = \alpha \,.\gamma$
  - **for all $x \in 2^D$, $x = \alpha(\gamma(x))$**
- **α and γ are monotonic**
- **Abstract operations op^A are locally correct, i.e.,**

$\gamma(op^A(a_1,...,a_n)$ is a superset of

$op(\gamma(a_1),..., \gamma(a_n))$



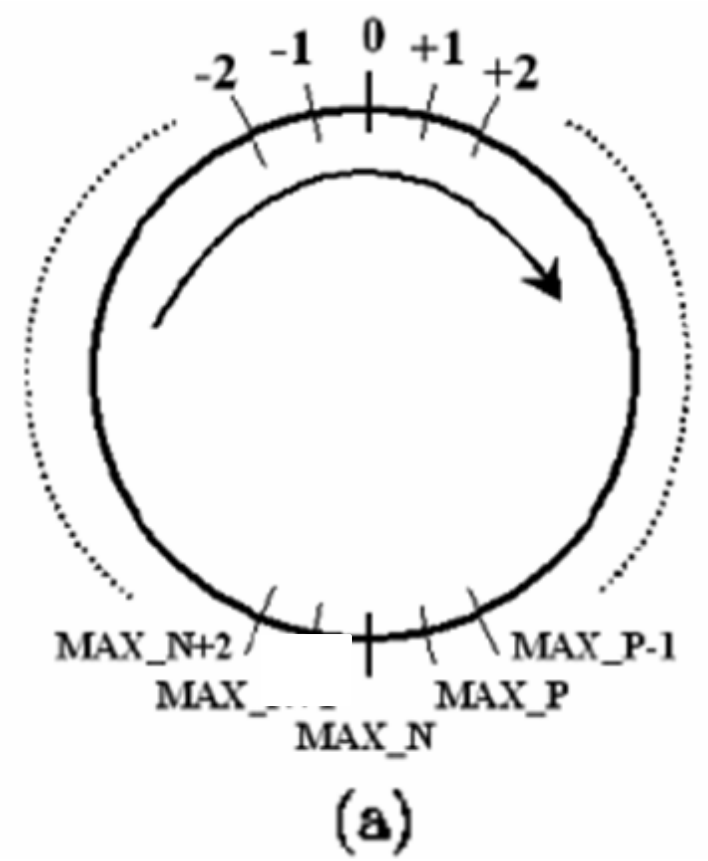
$\mu(e) \in S, \quad S \in 2^D$

Circular Linear Progressions (CLP)

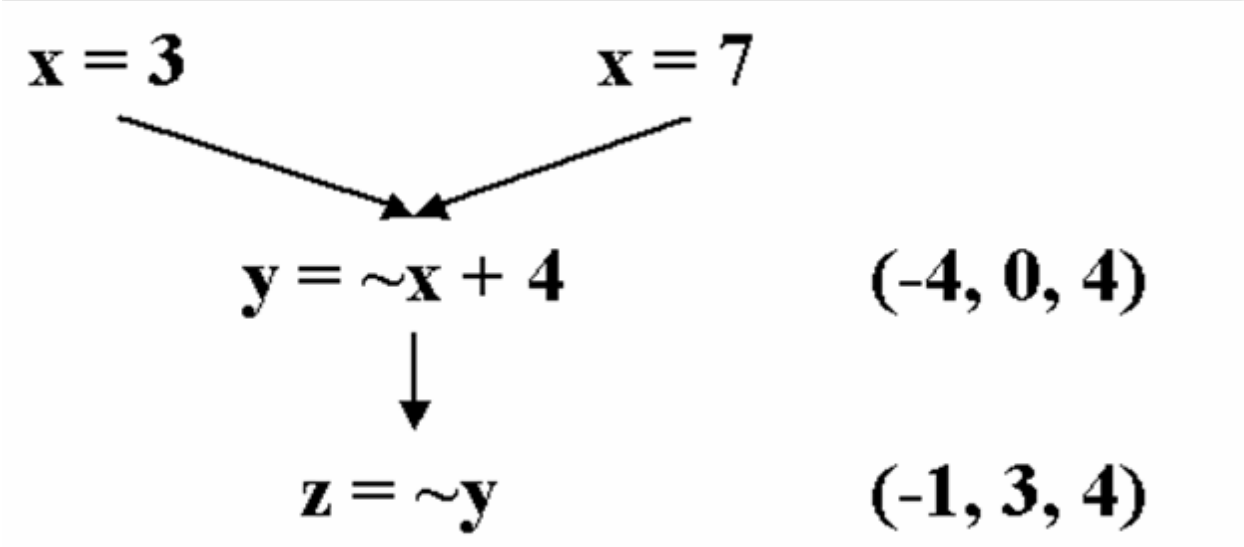
- Abstraction for finite width computations
- CLPs are used to represent the discrete values contained in various static objects, *viz.*, registers, memory partitions, etc.
- Safety on overflow
- Easily Composable
  - Definitions for arithmetic, logical, set, bitwise operations
- Efficient analysis
  - Quadratic space and time complexity

The CLP domain

- 3-tuple representation (l,u,δ), *using* a finite number of bits
  - Lower bound l
  - Upper bound u
  - Step δ
- Visualization
- (-1,1,2) vs (1,-1,2)



Example




```
  11111100 = -4 = ~3
+ 00000100 =  4
-----------------
1 00000000 =  0
  (overflow)
```

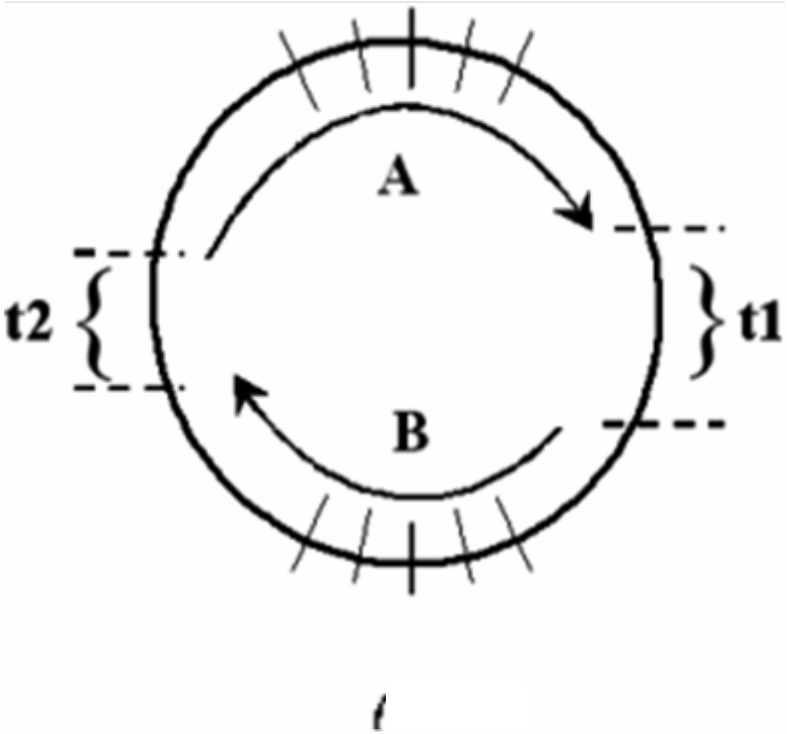
```
  11111000 = -8 = ~7
+ 00000100 =  4
-----------------
  11111100 = -4
 (no overflow)
```

Compositions

- Set
  - Union
  - Intersection
  - Difference
- Arithmetic
  - Addition
  - Subtraction
  - Multiplication
  - Division
- Shift
  - Left, Right
- Bitwise
  - AND
  - NOT
- Comparison
  - Equality, Inequality
  - Less than, Greater than

Example - Union

- Select alternative for *diff* as t_1 or t_2 for minimum over-approximation



$$(l_1, u_1, \delta_1) \cup (l_2, u_2, \delta_2) \subseteq (a, b, gcd(diff, \delta_1, \delta_2))$$

Subproblem 2: Cache Analysis

- Objective
  - To compute a lower bound on the number of cache hits
- Extension of the Abstract Cache model and Must Analysis technique

Cache Must-Analysis

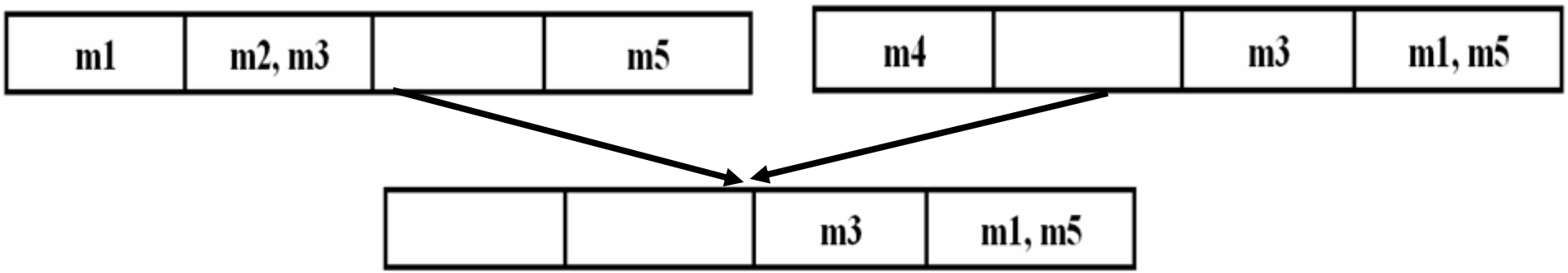
- Tracks the set of memory blocks definitely residing in the cache at any program point
- Useful for tracking memory accesses that will always result in cache hits regardless of program input
- Only set associative caches with perfect LRU replacement policy
- Extensions
  - To support sets of access addresses
  - When individual accesses cannot be guaranteed

Overview of Cache Analysis

- Abstract Interpretation using elements from abstract cache domain
- Abstract cache
  - blocks in a set arranged in increasing order of age
  - each block can hold data corresponding to a set of memory blocks (not one block as in the concrete case)
- Abstract cache state at any point in the program
  - a safe approximation of all possible concrete cache states at that point over various execution sequences

MUST Analysis

- Provides guarantees of upper bounds of ages of memory blocks in the cache
  - If a memory block is present in the abstract cache state, the corresponding access will *always* be a hit
  - Lower bound on the number of hits
- "Join" computation takes *maximum* ages



Key Differences Between Instruction and Data References

- Address set for the latter may not be a singleton set, as for example, array references
- When the address set is not singleton, we cannot say which particular subset of addresses will be definitely accessed during actual execution
- No new element can be brought into the abstract cache as that element may never be accessed during any concrete execution
- If the address set is singleton, the addressed memory block will always be brought into the cache

State Update (Extended)

- Straightforward for singleton address set
- Others (say array access)
  - Individual accesses cannot be guaranteed
  - No new memory block can be brought into the abstract cache
  - Memory blocks in the cache cannot decrease in age
- Example:

		m1	m2	m3, m4
m3	m3		m1	m2, m4
m5	m5	m3		m1
m2, m3			m3, m5	

Reference Classification

- At fix-point
  - ah : if all memory blocks in the access set (CLP) are in the abstract cache (always hit)
  - nc : otherwise (non-classified)
- Latency calculation
  - Hit latency for ah references
  - Miss latency for nc references
  - Conservative, but safe

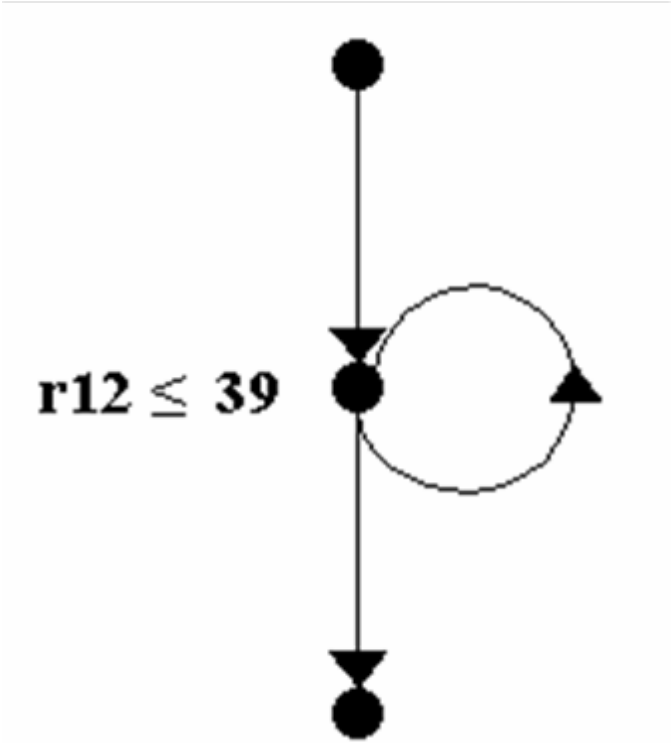
Subproblem 3: Access Sequencing

- Objective
  - Determine frequency and ordering of accesses to distinct memory locations (referenced during execution)

Overview

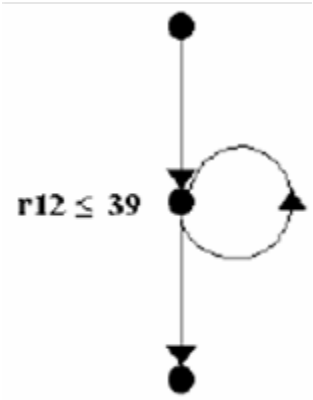
- Sets of memory addresses do not incorporate reuse and conflict information
  - {x,y} represents accesses x,x,x,y and x,y,x,y
- Idea is to unroll loops partially
  - Both physical and virtual unrolling
    - Physical unrolling creates “regions”
    - Analysis alternates between *expansion* and *summary* modes
  - Extent of unroll is controlled by the user
    - Two parameters: *frac_exp* and *samples*

Example Loop

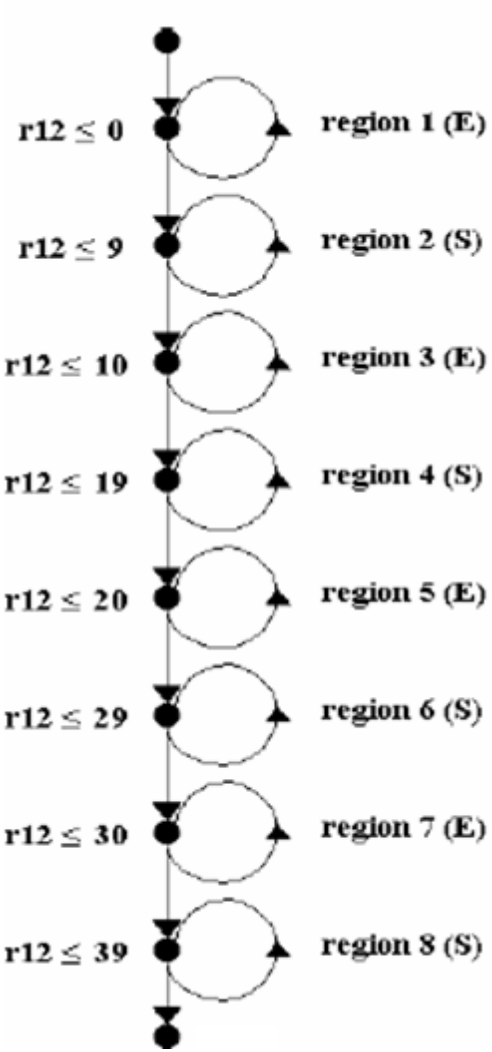



```
#define MAX 40

short b[MAX],c[MAX];

int main()
{
     int i,k;
     k=0;
     for(i=0;i<MAX;i++)
          k+=b[i]*c[i];
     return k;
}
```

Example



- frac_exp = 0.1 (10%)
- samples = 4
- #regions = 4*2 = 8
- 10% of the iterations will be analyzed in E-mode spread over 4 regions

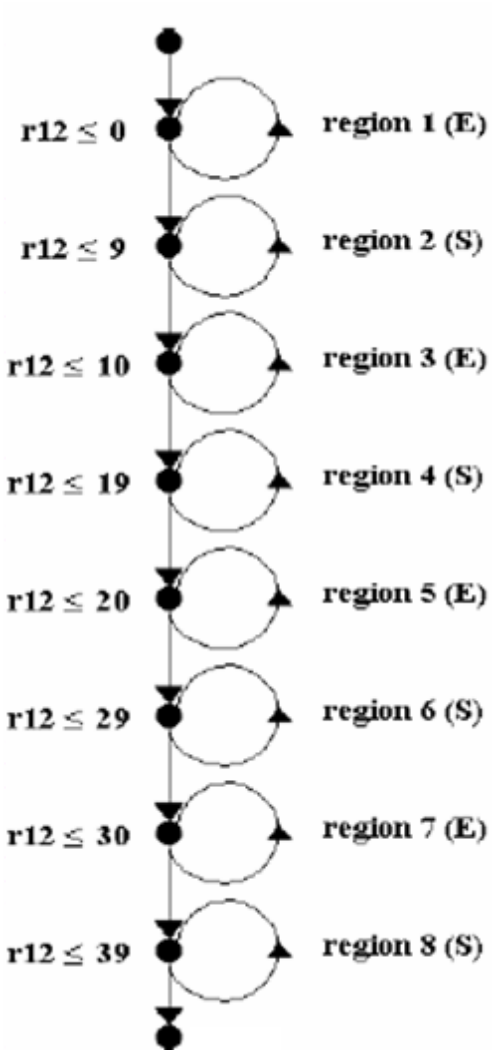


Analysis Modes

- Expansion
  - Virtual unrolling
  - No fix-point iteration
  - Simultaneous address & cache analyses
  - Slow
  - Helps to prime dcache
- Summary
  - No virtual unrolling
  - Fix-point iterations
  - First address, then cache analysis
  - Fast

- Usually higher incidence of singleton accesses in expansion than in summary mode
- Modes are equivalent for non-loop portions

Sample Analysis



Region	Address(hex)	Block(hex)	Set	Class
1	20080e0	100407	7	nc
	20080e1	100407	7	ah
	2008130	100409	9	nc
	2008131	100409	9	ah
2	(20080e2,20080f2,2)	100407	7	ah
	(20080e3,20080f3,2)	100407	7	ah
	(2008132,2008142,2)	100409,10040a	9,10	nc
	(2008133,2008143,2)	100409,10040a	9,10	nc
3	20080f4	100407	7	ah
	20080f5	100407	7	ah
	2008144	10040a	10	nc
	2008145	10040a	10	ah
4	(20080f6,2008106,2)	100407,100408	7,8	nc
	(20080f7,2008107,2)	100407,100408	7,8	nc
	(2008146,2008156,2)	10040a	10	ah
	(2008147,2008157,2)	10040a	10	ah
5	2008108	100408	8	nc
	2008109	100408	8	ah
	2008158	10040a	10	ah
	2008159	10040a	10	ah
6	(200810a,200811a,2)	100408	8	ah
	(200810b,200811b,2)	100408	8	ah
	(200815a,200816a,2)	10040a,10040b	10,11	nc
	(200815b,200816b,2)	10040a,10040b	10,11	nc
7	200811c	100408	8	ah
	200811d	100408	8	ah
	200816c	10040b	11	nc
	200816d	10040b	11	ah
8	(200811e,200812e,2)	100408,100409	8,9	ah
	(200811f,200812f,2)	100408,100409	8,9	ah
	(200816e,200817e,2)	10040b	11	ah
	(200816f,200817f,2)	10040b	11	ah

An Estimation Heuristic

- References may be classified as nc even if potential reuse possibilities exist
- Probable average latency:

$$access\_latency = frac\_hit \times hit\_latency + (1 - frac\_hit) \times miss\_latency$$

- May not be safe as accesses cannot be guaranteed
- Useful for
  - Soft real time systems
  - Reasoning about the tightness of the safe estimate

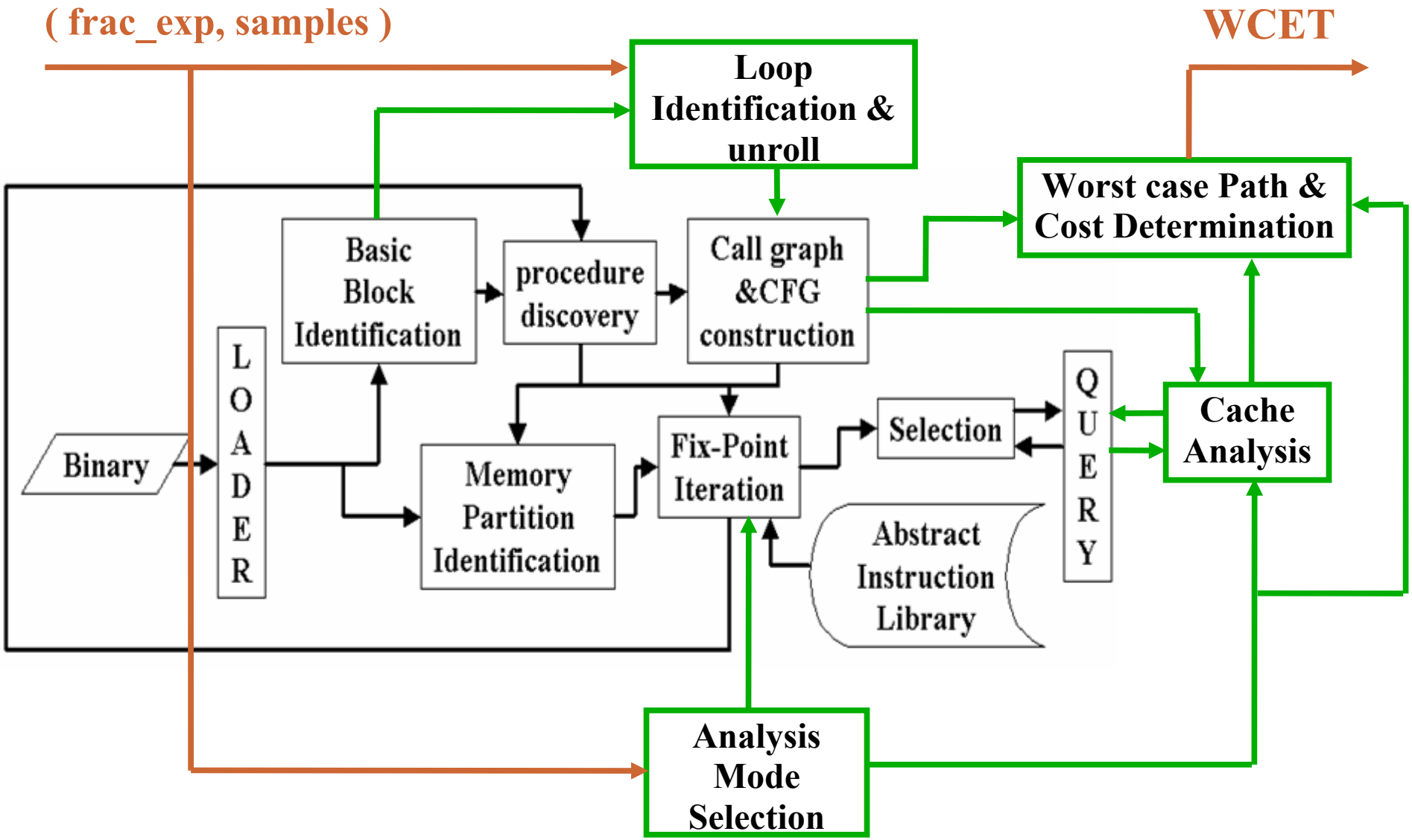
Subproblem 4: Worstcase Path Analysis

- Objective
  - To compute the overall worst case path in the program and the associated cost
- After the worst case execution costs for each basic block has been individually computed, an approximation of the overall worst case cost and corresponding path is obtained by solving an ILP

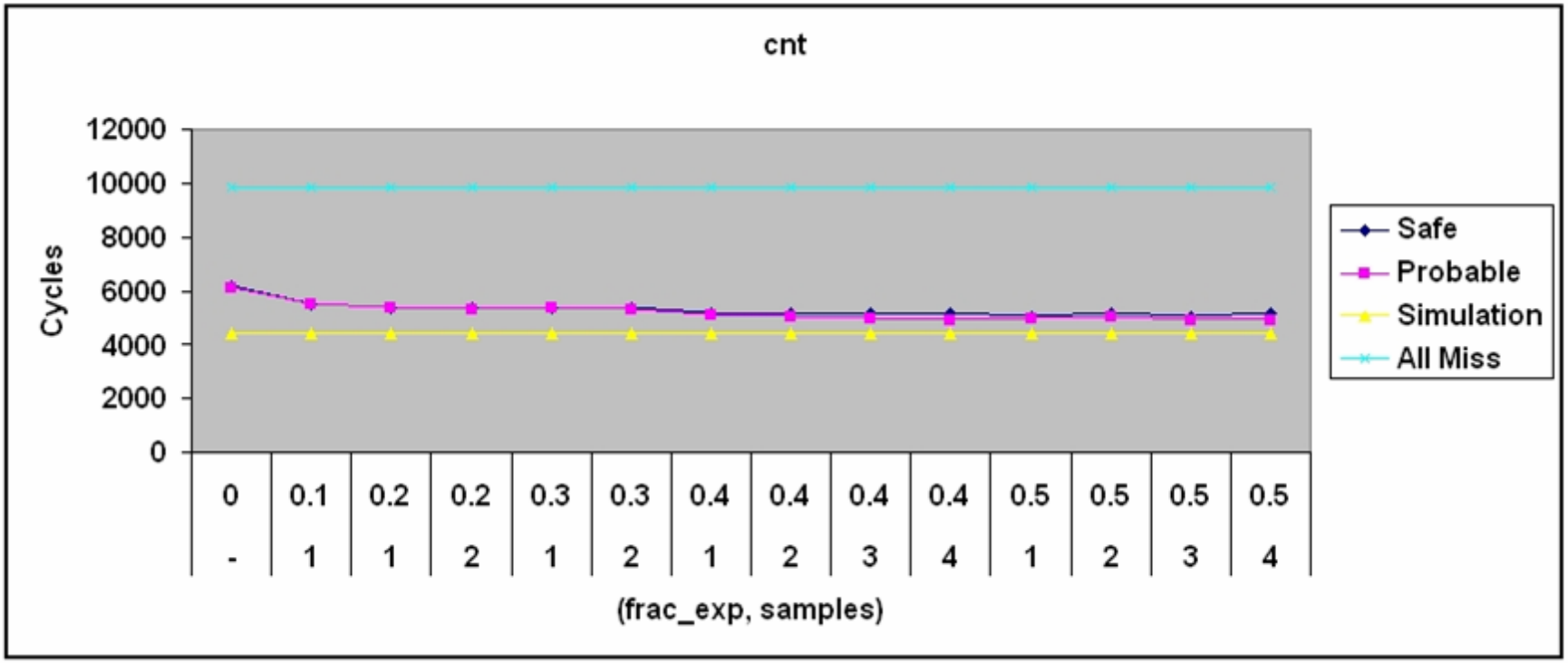
Overview

- Integer Linear Programming to maximize overall execution cost subject to structural constraints
  - Flow
  - Loop
  - Interprocedural
- Objective function: $\sum_{i=1}^{B} w_i \times x_i$
  - x_i is the variable for block i
  - w_i is the worst case cost of basic block i

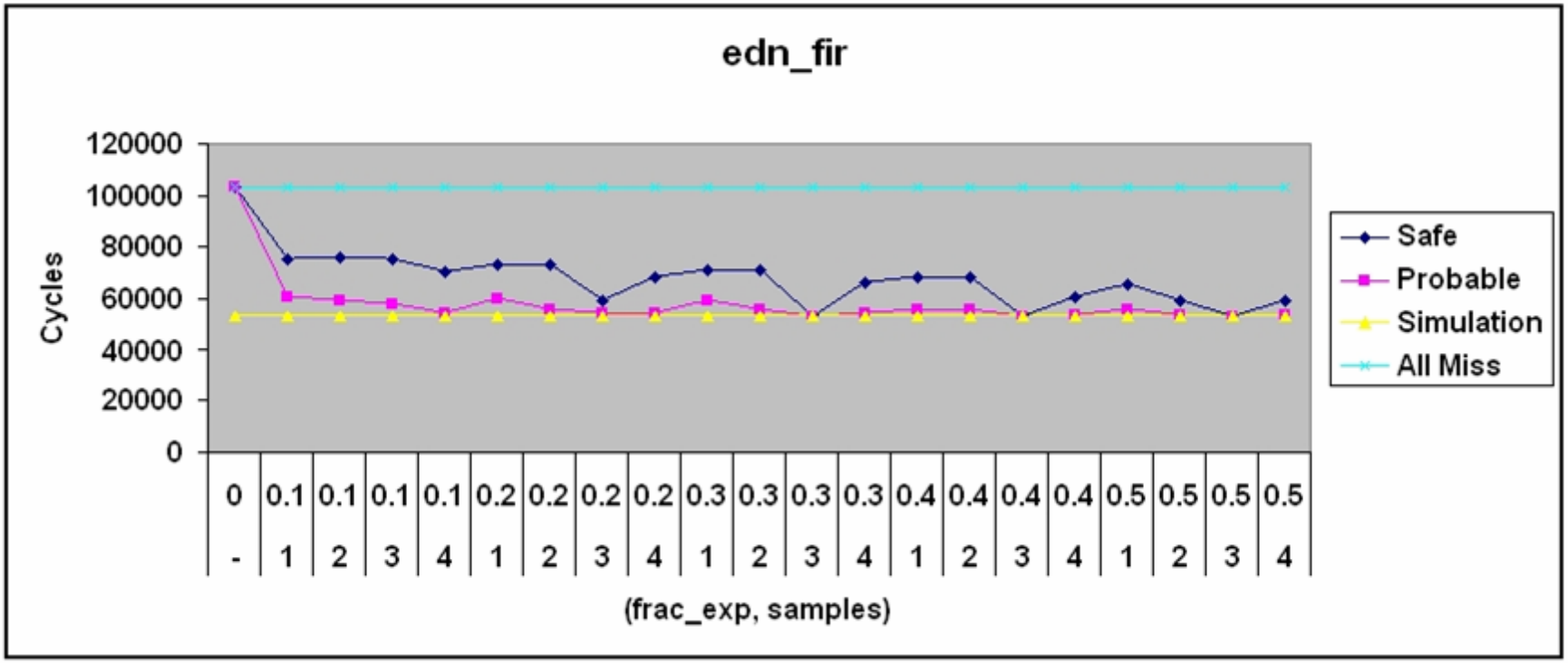
Implementation



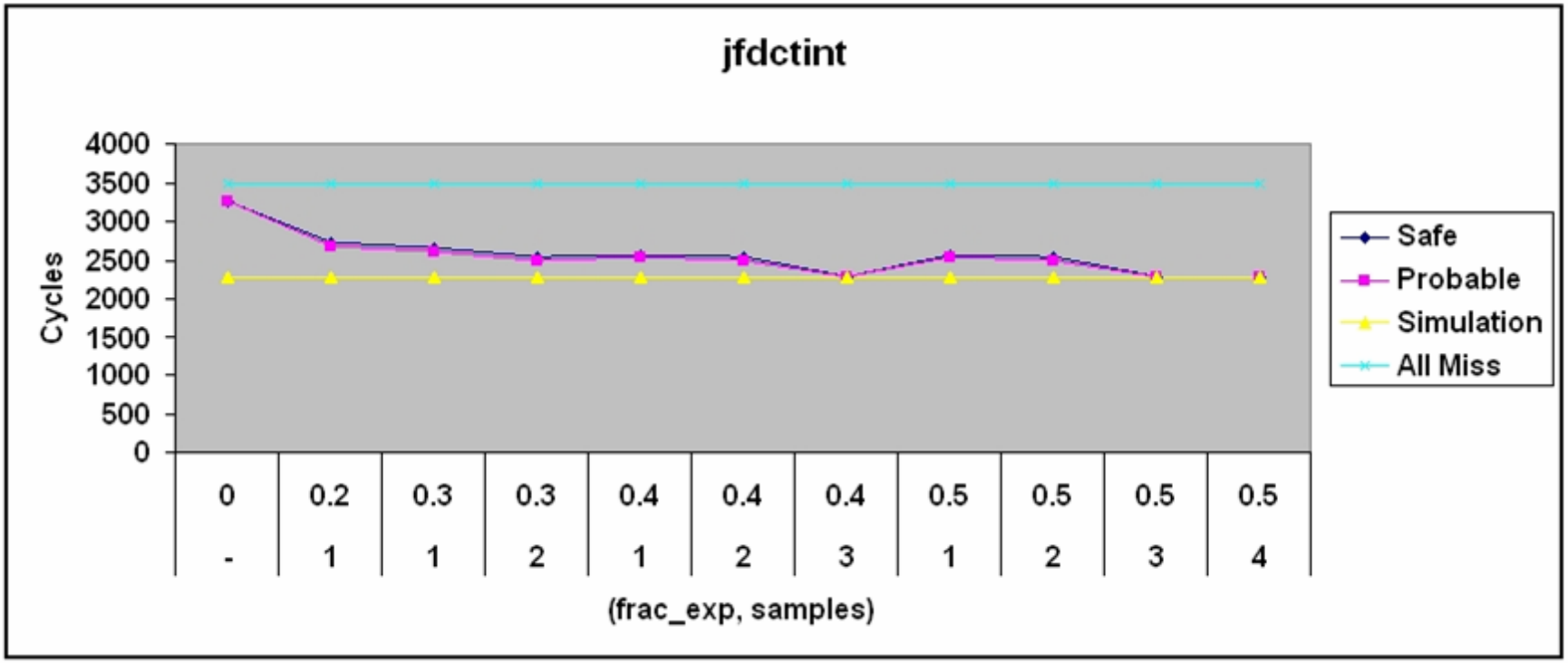
WCET estimates



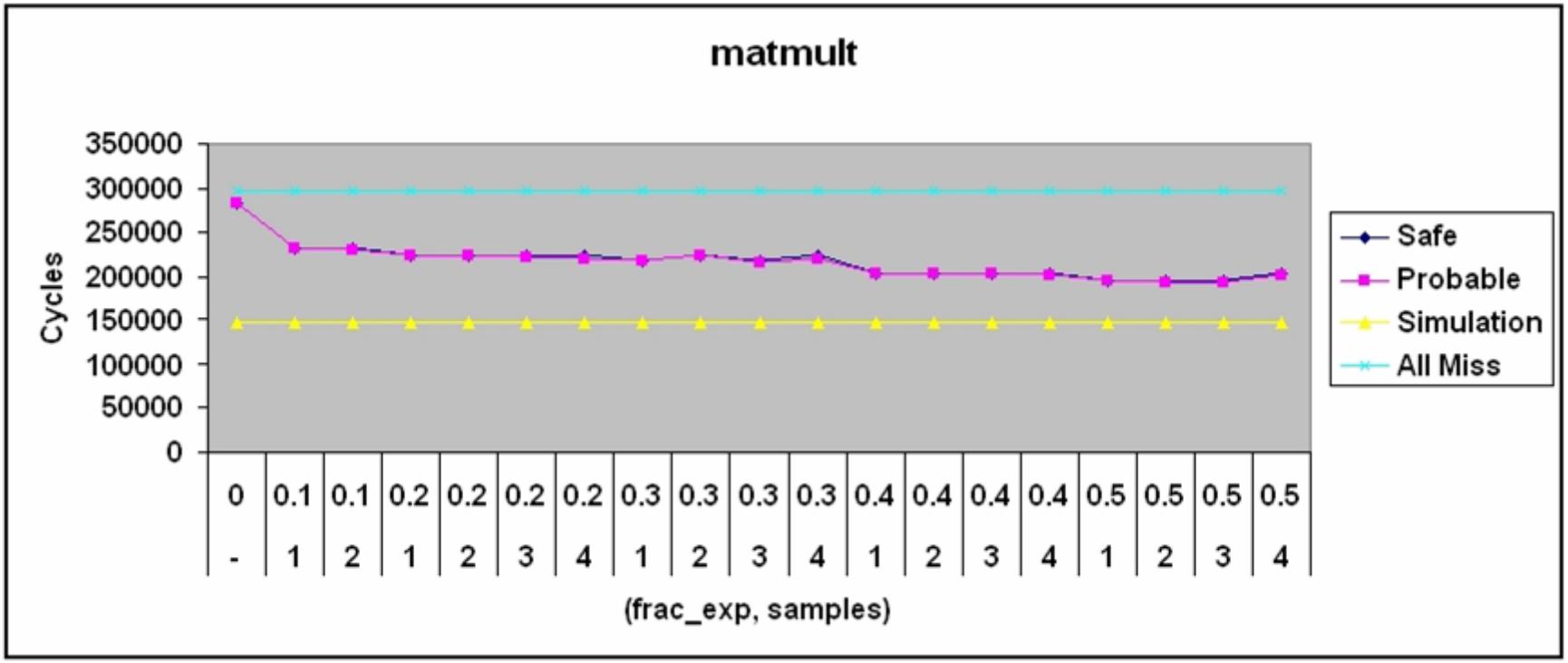
WCET estimates



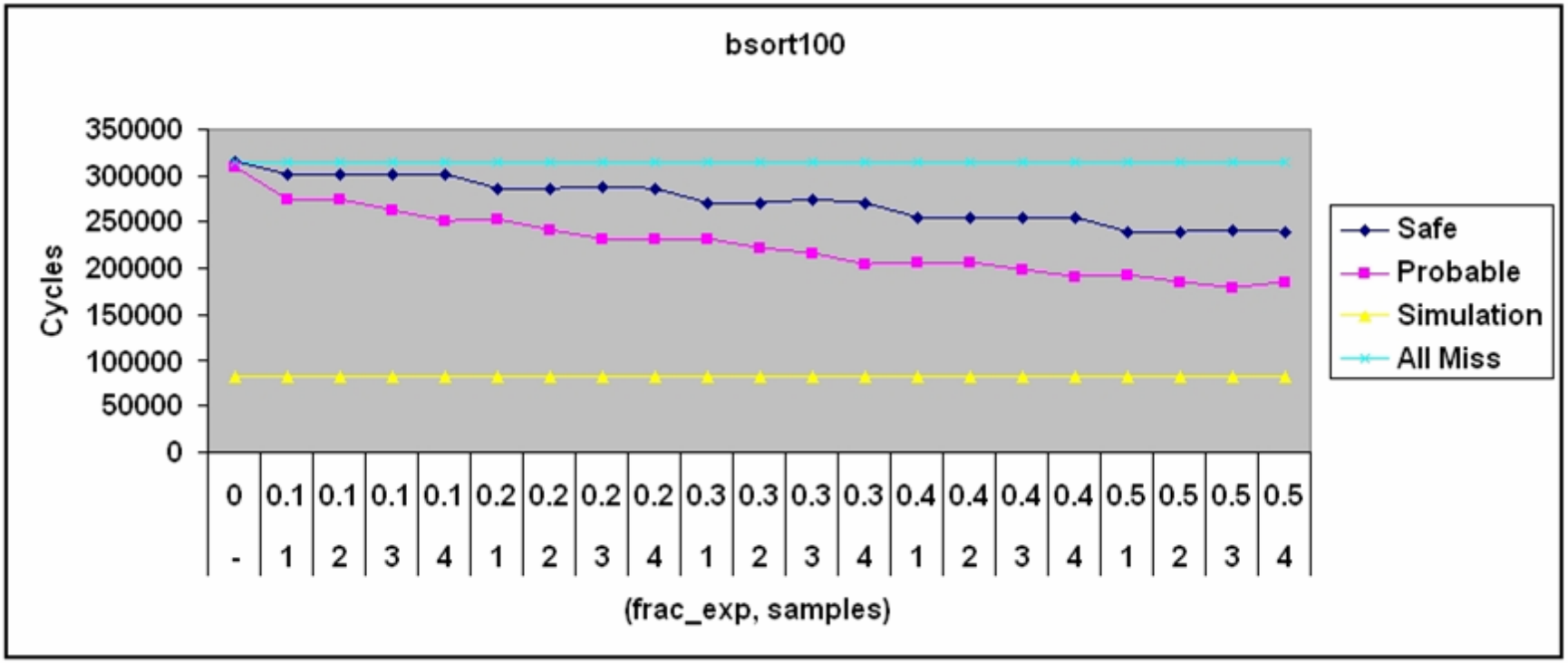
WCET estimates



WCET estimates



WCET estimates



Conclusions

- WCET analysis for executables
- Modular approach
- CLP for address analysis
- Extension of MUST analysis to support
  - Non scalar references
  - When individual accesses cannot be guaranteed
- Partial physical and virtual unrolling for access sequencing
- Heuristic for soft real time systems

Future Work

- WCET estimation
  - in the presence of a cache hierarchy
  - with dynamic voltage scaling
  - for multi-core architectures and concurrent programs
  - with other cache replacement policies

Thank You

References

1. P. Cousot and R. Cousot, *Abstract Interpretation: A Unified lattice model for static analysis of programs by construction or approximation of fixpoints,* ACM POPL, 1977.

2. C. Ferdinand and R. Wilhelm, *Fast and efficient cache behaviour prediction for real-time systems,* Real-Time Systems, 17(2/3), Springer, 1999.

3. R. Sen and Y.N. Srikant, *Executable analysis using abstract interpretation with circular linear progressions*, ACM MEMOCODE 2007.

4. R. Sen and Y.N. Srikant, *WCET Estimation for executables in the presence of data caches*, ACM EMSOFT 2007.