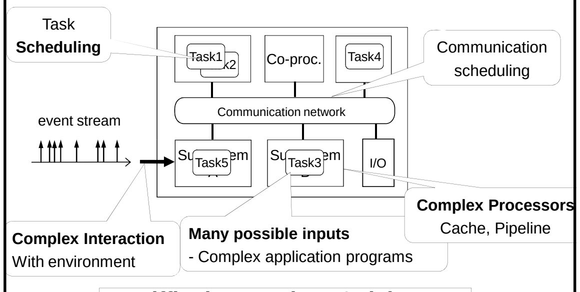
Performance Validation of SW
CS 4271 Lectures 8, 9

Abhik Roychoudhury
National University of Singapore

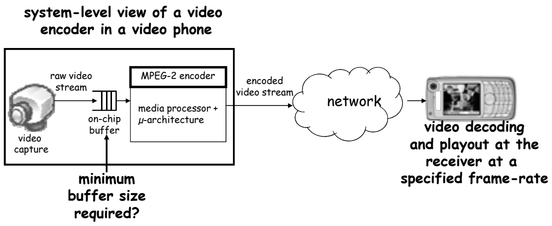
http://www.comp.nus.edu.sg/~abhik/

Timing in Embedded Systems



Difficult to analyze & debug!

Example Set-up


system-level view of a video encoder in a video phone
raw video stream
MPEG-2 encoder
media processor + μ-architecture
on-chip buffer
video capture
encoded video stream
network
video decoding and playout at the receiver at a specified frame-rate
minimum buffer size required?

Need to look inside the different processing tasks, and analyze their timing!

Time is abstracted!

- Our programming languages do not mention time
  - C, Java, C#, C++
- Even an instruction takes variable time
  - Hit/miss in instruction cache
  - Hit/miss in data cache
  - Pipeline stalls
    - Data hazard
    - Resource contention
  - Branch prediction …
- Need timing analysis of programs!

Timing analysis of programs

- Estimating uninterrupted software execution time on a given hardware (processor).
- A building block for more complicated performance analysis.
  - Communicating multi-processor execution.
- Helps estimate performance of a design point.
  - Serves as a sub-routine for Design Space Exploration.

Timing analysis of programs

- Schedulability analysis of Hard Real-time systems.
  - Such analysis assumes knowledge of WCET of each task being scheduled.
    - WCET stands for Worst-case Execution Time
  - Rate Monotonic scheduling with tasks $T_1,\ldots,T_n$
    - Computation times $C_1,\ldots,C_n$
    - Period = deadline $D_1,\ldots,D_n$
    - Here $C_1,\ldots,C_n$ are the WCET (not average execution times of the programs)

Organization

- Software timing analysis
  - WCET analysis
- System level analysis
  - Schedulability analysis
- Design issues to improve timing predictability
  - Scratchpad memories

WCET

- Worst Case Execution Time (WCET) of a program for a given hardware platform.
  - Sequential Terminating Programs.
  - Gets input, computes, produces output.
- Many inputs are possible.
  - Leads to different execution times.
- WCET :An upper bound on the execution time for all possible inputs.

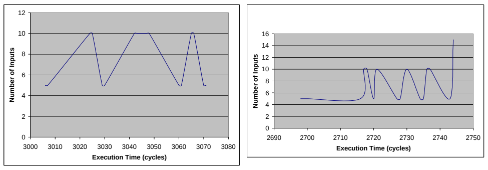
Why need analysis?

- To find WCET of a program, execute it for all possible inputs.
  - WCET by measurement.
  - Exponentially many possible inputs in terms of input size.
    - Insertion sort program
  - Similar problems will be encountered for WCET Analysis via platform simulation.
- Need access to platforms/simulators also!
  - Go for static analysis.

WCET by measurement?

- What about single path programs such as matrix multiplication ?
  - Execution path is independent of input data.
  - Still execution time can be variable.
    - Latency of floating point operation (e.g., multiplication) depends on the input data.
  - Not possible to try it on all possible platforms and then choose one.
    - Often trying to decide the platform as well.

Why Platform-aware Analysis?

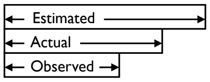


Distribution of execution times across inputs in a quicksort program on a simple and complex processor

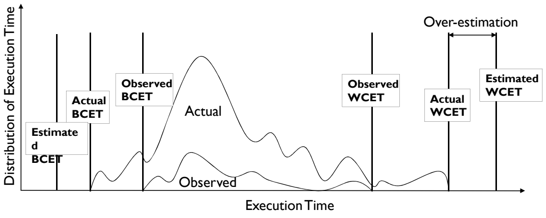
WCET Analysis

- Employ static analysis to compute an upper bound on actual WCET (**Estimated WCET**)
- Run program on selected inputs get a lower bound on actual WCET (**Observed WCET**)

Estimated WCET ≥ Actual WCET ≥ Observed WCET



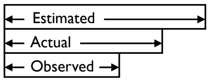
BCET and WCET



Copyright (c) 2009, Abhik Roychoudhury

WCET Analysis

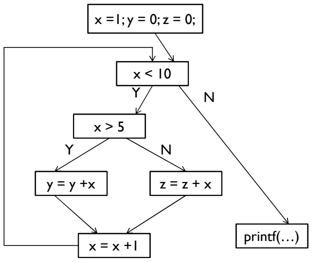
- Program path analysis
  - All paths in **control flow graph** are not feasible.
- Micro-architectural modeling
  - Dynamically variable instruction execution time.
    - Cache, Pipeline, Branch Prediction
    - Out-of-order Pipelines



Copyright (c) 2009, Abhik Roychoudhury

Clarification: Control flow graph

- x = 1; y = 0; z = 0;
- while (x < 10){
-     if (x > 5)
-             y = y + x;
-     else   z = z + x;
-     x = x + 1;
- }
- printf(….);



Nodes of the graph, basic blocks, are maximal code fragments executed without control transfer. The edges denote control transfer.

Copyright 2009 by Abhik Roychoudhury

Exercise: CFG

```
procedure Check_data()
{        int  i = 0, morecheck = 1, wrongone = -1, datasize = 10;
   L:    while (morecheck)
   LB:  {
                 if  (data[i] < 0)
   A:               {  wrongone = i;   morecheck = 0; }
                 else
   B:                    if  (++i >= datasize) morecheck = 0;
              }
              if (wrongone >= 0)
    C:              { handle_exception(wrongone); return 0; }
    C':     else  return i;
 }
```

Copyright 2009 by Abhik Roychoudhury

Exercise on CFG

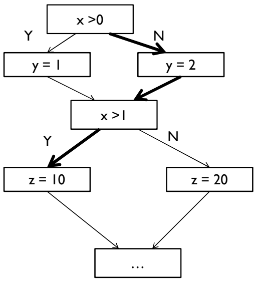
- Construct the CFG of the procedure given in the previous slide.
- How to construct an inter-procedural CFG for a program with many procedures?

```
main(){          f1(){         f2(){
…                  …              …
   f1();          f2();          f2();
   f2();           …              …
…                }              }
}
```

Copyright 2009 by Abhik Roychoudhury

Why all paths may not be feasible?

- if (x > 0{
-     y = 1;
- else
-     y = 2;
- }
- if (x > 1){
-     z = 10;
- else
-     z = 20;
- }
- …



Copyright 2009 by Abhik Roychoudhury

Restrictions of analysis – (1)

- Static analysis need not be on source program.
  - We can perform static analysis on assembly code of a given program.
  - The analysis is only for time taken, and not for the memory locations / values accessed.
  - No restriction on program data structures used for WCET analysis.
  - What about control flow ?

Restrictions of analysis – (2)

- Restrictions on control flow
  - 1. No unbounded loops
    - Common sense.
      - □ Otherwise how to guarantee time?
  - 2. No unbounded recursion
    - Similar issue.
  - 3. No dynamic function calls
    - Need to statically know the functions called, and the possible call sites of these functions.

Organization of WCET Analysis

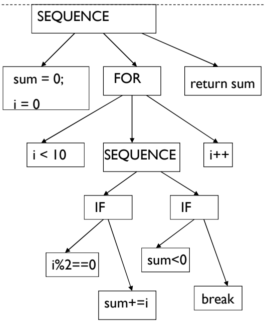
- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- Modeling Program Flows.
  - Primarily Control flow.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.

Timing Schema

- One of the first works on WCET analysis.
- Basically, perform control flow analysis to find the "longest" program path.
- The notion of "longest" is weighted
  - Take into account the cost of executing individual program elements.
  - Timing schema is a simple way of composing these costs.
- Does not work on Control Flow Graphs
  - Works on Abstract Syntax Tree

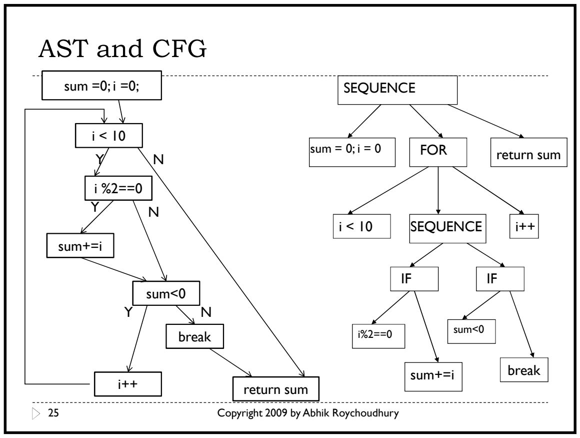
Example

```
sum = 0;
for (i=0; i< 10; i++){
    if (i % 2 == 0)
            sum += i;
    if (sum < 0)
            break;
}
return sum;
```



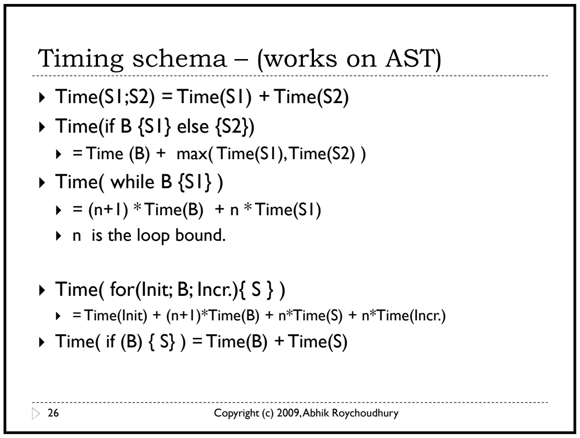
AST and CFG

- Hierarchy
  - AST shows the different scopes at different levels
  - CFG has no hierarchy.
- Loops
  - AST is tree, free from cycles
  - Any loop in the program is a cycle in the CFG.



AST and CFG

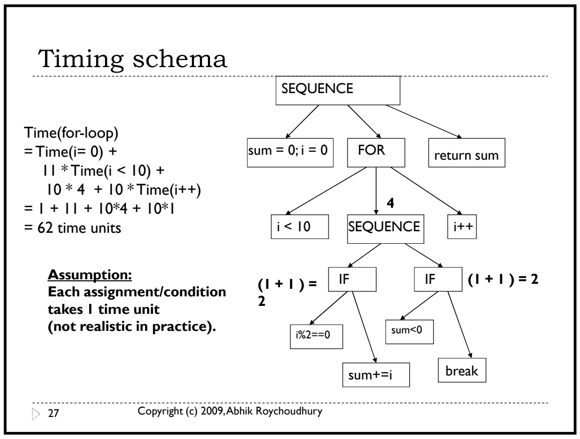
25 Copyright 2009 by Abhik Roychoudhury



Timing schema – (works on AST)

- Time(S1;S2) = Time(S1) + Time(S2)
- Time(if B {S1} else {S2})
  - = Time (B) + max(Time(S1),Time(S2))
- Time(while B {S1})
  - = (n+1) * Time(B) + n * Time(S1)
  - n is the loop bound.
- Time(for(Init; B; Incr.){ S })
  - = Time(Init) + (n+1)*Time(B) + n*Time(S) + n*Time(Incr.)
- Time(if (B) { S}) = Time(B) + Time(S)

26 Copyright (c) 2009, Abhik Roychoudhury

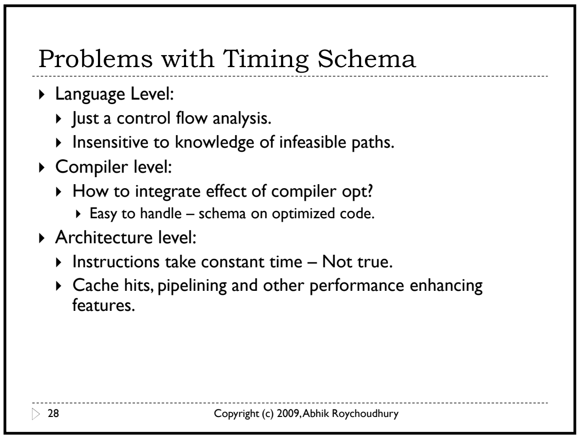


Timing schema

Time(for-loop)
= Time(i= 0) +
11 * Time(i < 10) +
10 * 4 + 10 * Time(i++)
= 1 + 11 + 10*4 + 10*1
= 62 time units

<u>Assumption:</u>
Each assignment/condition takes 1 time unit (not realistic in practice).

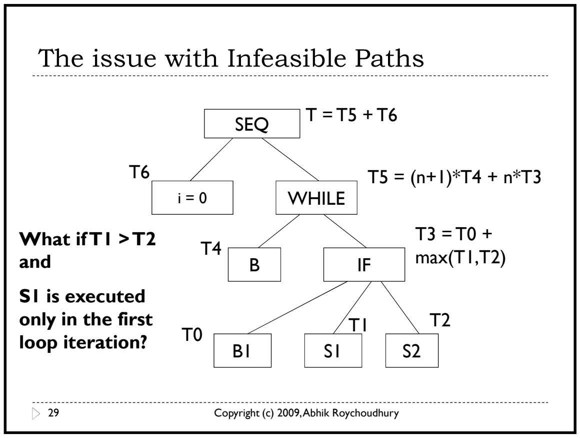
27 Copyright (c) 2009, Abhik Roychoudhury



Problems with Timing Schema

- Language Level:
  - Just a control flow analysis.
  - Insensitive to knowledge of infeasible paths.
- Compiler level:
  - How to integrate effect of compiler opt?
    - Easy to handle – schema on optimized code.
- Architecture level:
  - Instructions take constant time – Not true.
  - Cache hits, pipelining and other performance enhancing features.

28 Copyright (c) 2009, Abhik Roychoudhury

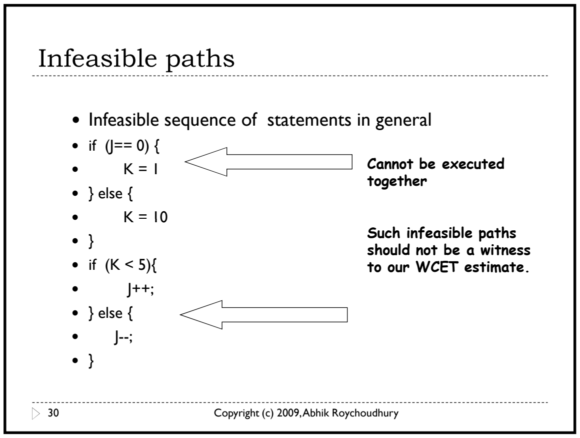


The issue with Infeasible Paths

What if T1 > T2 and

S1 is executed only in the first loop iteration?

29 Copyright (c) 2009, Abhik Roychoudhury



Infeasible paths

- Infeasible sequence of statements in general
- if (J== 0) {
-     K = 1
- } else {
-     K = 10
- }
- if (K < 5){
-     J++;
- } else {
-     J--;
- }

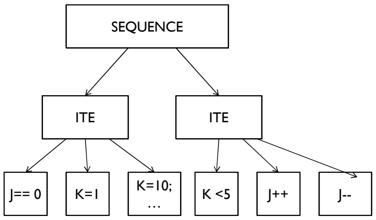
Cannot be executed together

Such infeasible paths should not be a witness to our WCET estimate.

30 Copyright (c) 2009, Abhik Roychoudhury

Infeasible Path handling in Timing Schema

- if (J== 0) {
-         K = 1
- } else {
-     K = 10; …
- }
- if (K < 5){
-         J++;…
- } else {
-     J--;
- }


SEQUENCE

ITE ITE

J== 0 K=1 K=10; … K <5 J++ J--

How will timing schema work on this example?

Working of timing schema

Time(first-if-statement)
= 1 + max(1,5) = 6

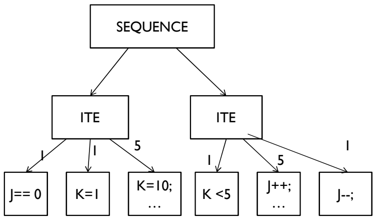
Time(second-if-statement)
= 1 + max(5,1) = 6

Estimated worst case
time = 6 + 6 = 12

Actual worst-case time
= 2 +6 = 8

Why?

Where is the overestimate from?

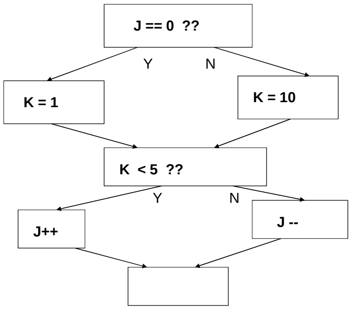

SEQUENCE

ITE ITE

1 1 5 1 5 1

J== 0 K=1 K=10; … K <5 J++; … J--;

Control flow graph (CFG)


J == 0 ??

Y N

K = 1

K = 10

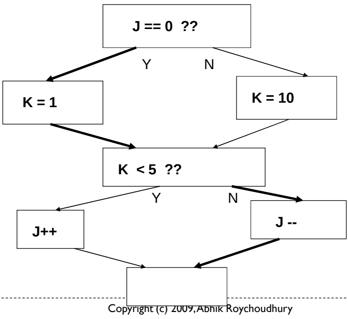
K < 5 ??

Y N

J++

J --

Infeasible path in CFG


J == 0 ??

Y N

K = 1

K = 10

K < 5 ??

Y N

J++

J --

Modeling of control flow

- Path-based
  - Enumerate paths and find longest path
    - Expensive !
    - Need to remove longest path if it is infeasible.
- Tree-based
  - Bottom-up pass of Syntax Tree
    - Timing Schema
  - Difficult to integrate infeasible path info
- Integer Linear Programming
  - Can take into account certain infeasible path information if available.
  - Efficient solvers available e.g. CPLEX
    - Forms the back-end of most state-of-the-art timing analyzers.

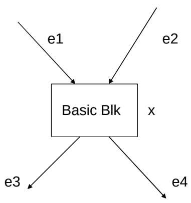
Integer Linear Programming

- ILP: Integer Linear Programming
  - Variables and linear constraints on them.
  - Cost function (linear) to optimize.

```
f = 3x + 5y + z
0 <= x, y, z <= 100
x + y + z = 200
x + 2y <= 160
```

```
Optimal:      f = 520; x = 40; y = 60; z = 100
Non-Optimal:  f = 480; x = 80; y = 30; z = 90
```

ILP Modeling

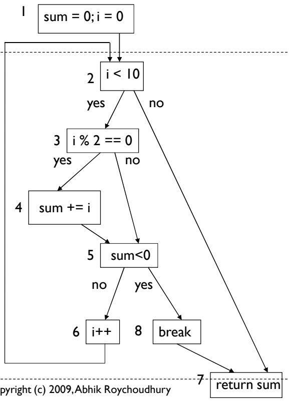


x = e1 + e2

= e3 + e4

We are dealing with aggregated execution counts of nodes/edges of CFG.

```
sum = 0;
for (i=0; i< 10; i++){
    if (i % 2 == 0)
            sum += i;
    if (sum < 0)
            break;
}
return sum;
```



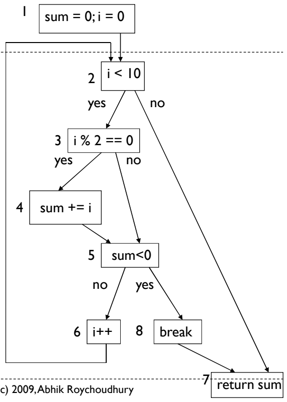
Maximize

Time =

$c_1N_1 + c_2N_2 + c_3N_3 + c_4N_4 +$
$c_5N_5 + c_6N_6 + c_7N_7 + c_8N_8$

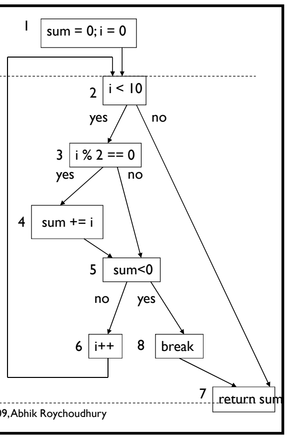
$1 = N_1 = E_{1,2}$
$E_{6,2} + E_{1,2} = N_2 = E_{2,3} + E_{2,7}$
$E_{2,3} = N_3 = E_{3,4} + E_{3,5}$
$E_{3,4} = N_4 = E_{4,5}$
$E_{3,5} + E_{4,5} = N_5 = E_{5,6} + E_{5,8}$
$E_{5,6} = N_6 = E_{6,7}$
$E_{5,8} = N_8 = E_{8,7}$
$E_{8,7} + E_{2,7} = N_7 = 1$

$E_{6,2} \leq 10$



Infeasible path

- The break statement is executed at most once.
  - $N_8 \leq 1$



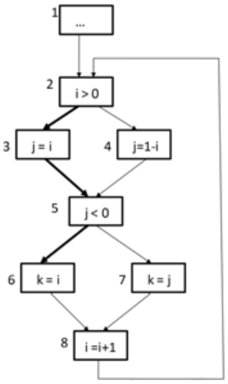
Blocks 3 and 6 are never executed in same loop iteration

$N_3 + N_6 \leq$ loopbound

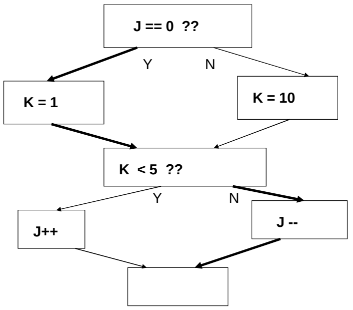
Edges (2,3) and (5,6) are not executed together.

$E_{2,3} = E_{5,7}$

```
while (...){
        if ( i > 0 ){
                j = i;
        } else{
                j = 1-i;
        }
        if (j < 0){
                k = i;
        } else{
                k = j;
        }
        i = i+1;
}
```



How to express this inf. path constraint?



Exercise: What are the infeasible paths?

```
procedure Check_data()
{       int  i = 0, morecheck = 1, wrongone = -1, datasize = 10;
        while (morecheck)
        {
             if  (data[i] < 0)
                { wrongone = i;   morecheck = 0; }
             else
                  if  (++i >= datasize) morecheck = 0;
          }
          if (wrongone >= 0)
                { handle_exception(wrongone); return 0; }
          else  return i;
 }
```

Organization

- What is Timing Analysis ?
- An Early solution – Timing schema
- Modeling Program Flows.
  - Primarily Control flow.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.

The two phases

- WCET analysis involves
  - Program path analysis – ILP
  - Micro-architectural modeling.
- How do the two analyses interact?
  - Time = c1*N1 + c2*N2 +c3*N3 + …
  - c1,c2, … : exec time of basic blocks 1,2, …
  - N1,N2, … : exec count of basic blocks 1,2, …
  - c1, c2, … are estimated by μ-arch. modeling
  - N1,N2, … are fixed by control flow analysis via Integer Linear Programming (ILP).

Instruction Cache Modeling

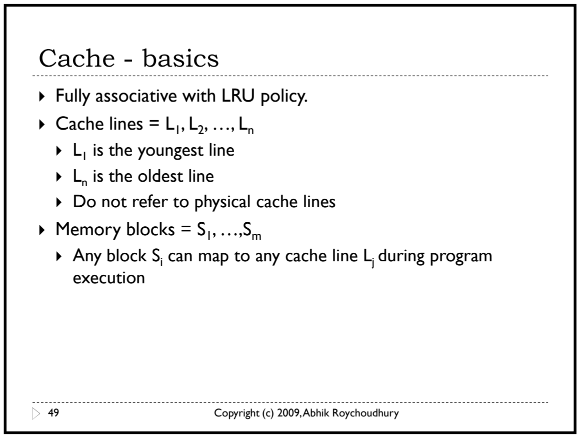
- One concrete hardware data structure.
- With no hardware modeling, all instructions should be taken as misses.
- Instead we can categorize some instructions as "always hit"
  - Coarse modeling.
  - For certain instructions, even the "worst case" may not be a miss !

Categorization

- … of instructions
  - AH (always hit)
  - AM (always miss)
  - PS (Persistent: second and all further executions are guaranteed to produce a hit)
    - Effect of cold misses
  - NC (not AH, AM, PS)

Cache - basics

- Redundant storage to reduce memory access time.
- Many memory blocks map to a single cache line
- F: Memory Block → Cache lines
  - Given a memory block m, F(m) returns the set of cache lines it can map to.
  - If F(m) is always a singleton set, then we have a direct mapped cache.
  - If |F(m)| is n, we have n-way set associative cache.
  - If F(m) = Set of all cache lines, then we have a fully associative cache (any memory block can map to any cache line).

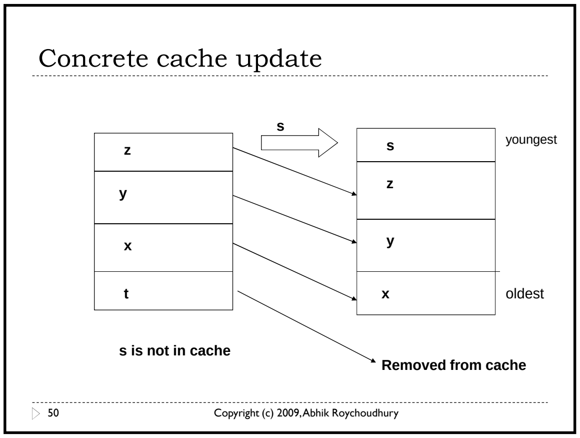


Cache - basics

- Fully associative with LRU policy.
- Cache lines = $L_1, L_2, \ldots, L_n$
  - L_1 is the youngest line
  - L_n is the oldest line
  - Do not refer to physical cache lines
- Memory blocks = $S_1, \ldots, S_m$
  - Any block S_i can map to any cache line L_j during program execution

49

Copyright (c) 2009, Abhik Roychoudhury



Concrete cache update

z
y
x
t

s

s
z
y
x

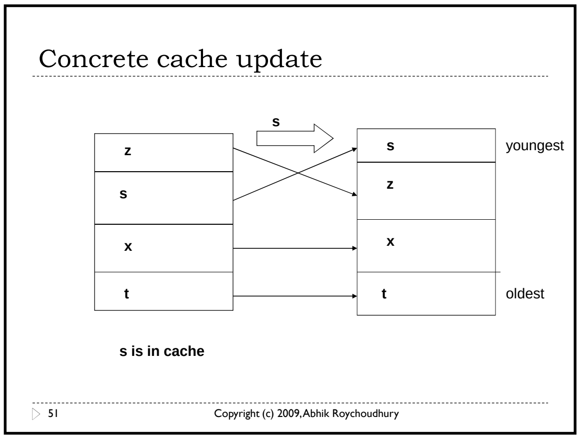
youngest
oldest

s is not in cache

Removed from cache

50

Copyright (c) 2009, Abhik Roychoudhury



Concrete cache update

z
s
x
t

s

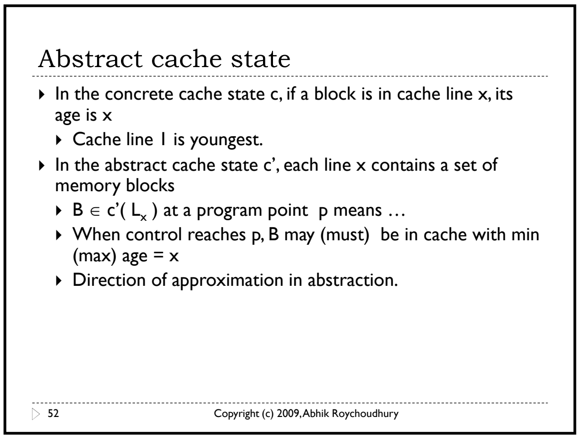
s
z
x
t

youngest
oldest

s is in cache

51

Copyright (c) 2009, Abhik Roychoudhury

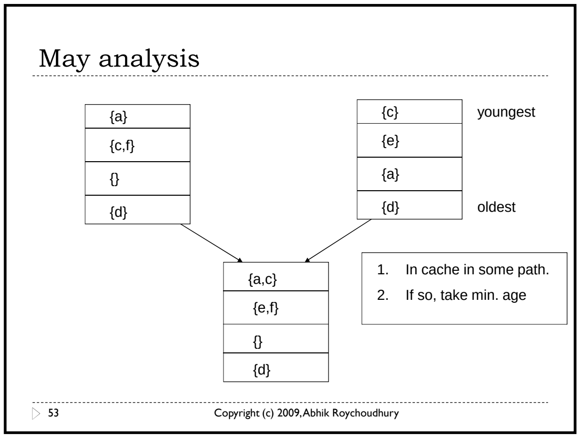


Abstract cache state

- In the concrete cache state c, if a block is in cache line x, its age is x
  - Cache line 1 is youngest.
- In the abstract cache state c', each line x contains a set of memory blocks
  - B ∈ c'(L_x) at a program point p means …
  - When control reaches p, B may (must) be in cache with min (max) age = x
  - Direction of approximation in abstraction.

52

Copyright (c) 2009, Abhik Roychoudhury



May analysis

{a}
{c,f}
{}
{d}

{c}
{e}
{a}
{d}

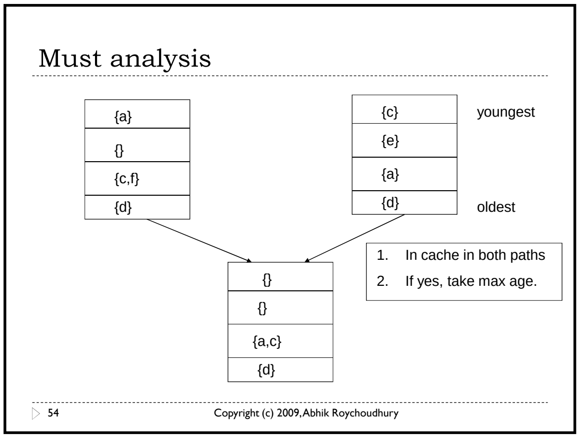
youngest
oldest

{a,c}
{e,f}
{}
{d}

1. In cache in some path.
2. If so, take min. age

53

Copyright (c) 2009, Abhik Roychoudhury



Must analysis

{a}
{}
{c,f}
{d}

{c}
{e}
{a}
{d}

youngest
oldest

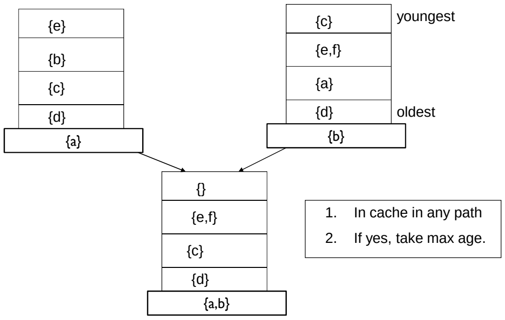
{}
{}
{a,c}
{d}

1. In cache in both paths
2. If yes, take max age.

54

Copyright (c) 2009, Abhik Roychoudhury

Persistence analysis



1. In cache in any path
2. If yes, take max age.

How to use such analyses?

- Let I be an instruction at control loc. CL
- Let M be the memory block containing I.
  - Consider cache state at CL obtained via "must analysis".
    - If M is in some cache line within this abstract cache state, then I is Always Hit.
  - For cache state at CL obtained via "may analysis"
    - If M is not in any cache line within this abstract cache state, then I is Always Miss.
  - For abstract cache state at CL obtained from persistence analysis
    - If M is not in the evicted line, then I is Persistent.

Use of may-must analyses

- Let hit_time = t1, miss_time = t2
- Number of accesses of I == #I (ILP variable)
  - I is AH
    - #I * t1 = contribution of I to WCET
  - I is AM
    - #I * t2 = contribution of I to WCET
  - I is PS
    - (#I -1)*t1 + t2 = contribution of I to WCET
- Formulation is still linear, solve via ILP.

Improving precision

- If we can bound the number of misses of instr. I (via constraints)
  - No need to reduce exec. Time of I to constant
  - Contribution of I to WCET
    - #miss(I)*t2 + (#I – #miss(I))*t1
  - Need constraints to bound #miss(I)
  - How to develop such constraints ?
    - ***ILP, Expensive !!***
    - See cache conflict graph approach in textbook
      - □ Pages 147 – 149.

Micro-arch. modeling so far

- **Modeling timing effects of I-cache**
  - Abstract Interpretation to categorize instr
  - ILP based modeling is more expensive.
- **I-cache does not have timing anomalies**
  - Can assume all accesses are misses.
  - Very pessimistic, but estimate still safe !
- For certain processors, even this is not true !
  - Adding worst-case of each instruction may produce an estimate lower than the global worst-case !

Pipelined execution

Divide the execution of an instruction into stages

Instruction I+1 can proceed before I completes

Increased throughput, lower overall execution time

SIMPLIFIED VIEW !!

	IF	ID	EX	WB	CM
0	I				
1	I+1	I			
2	I+2	I+1	I		
3	I+3	I+2	I+1	I	
4	I+4	I+3	I+2	I+1	I

Out-of-order pipeline

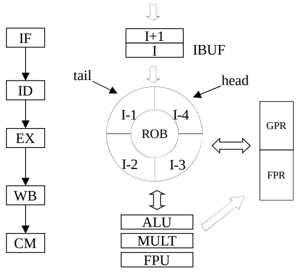
Mem => I-buffer (inorder)

IBUF => ROB (in-order)

ROB => FU (out-of-order), (Instr still in ROB)

FU => ROB (out-of-order) (forward data)

Update register file, free ROB entry (in-order)



O-o-o execution (1)

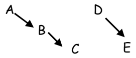
#	Ready Cycle	Instruction
A	0	mult r3 r1 r2
B	1	add r3 r3 8
C	2	and r3 r3 0xff
D	3	addu r5 r4 8
E	4	mult r5 r5 r6

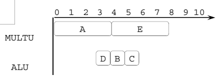
Instruction sequence

MULTU	1 ~ 4	cycles
ALU	1	cycle

Latencies

Partial order of dependences





Instruction A executes 4 cycles

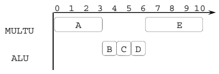
O-o-O execution (2)

#	Ready Cycle	Instruction
A	0	mult r3 r1 r2
B	1	add r3 r3 8
C	2	and r3 r3 0xff
D	3	addu r5 r4 8
E	4	mult r5 r5 r6

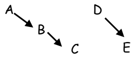
Instruction sequence

MULTU	1 ~ 4	cycles
ALU	1	cycle

Latencies



Instruction A executes 3 cycles



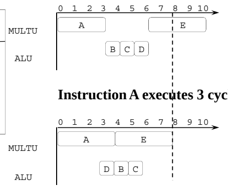
Difficulty in modeling

#	Ready Cycle	Instruction
A	0	mult r3 r1 r2
B	1	add r3 r3 8
C	2	and r3 r3 0xff
D	3	addu r5 r4 8
E	4	mult r5 r5 r6

Instruction sequence

MULTU	1 ~ 4	cycles
ALU	1	cycle

Latencies



Instruction A executes 3 cycles

Instruction A executes 4 cycles

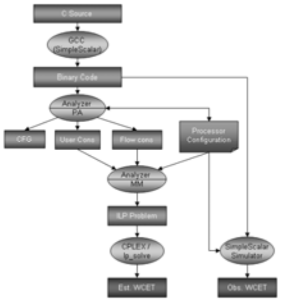
Timing Anomaly

- Overall WCET of an instruction sequence cannot be obtained from WCET of each instruction
- Need to consider all possible execution times of each instruction to safely estimate WCET !
  - Expensive enumeration
- Very different from cache modeling
  - Worst-case cache behavior of an instruction sequence can be safely estimated by considering all cache accesses as misses

Summing up …

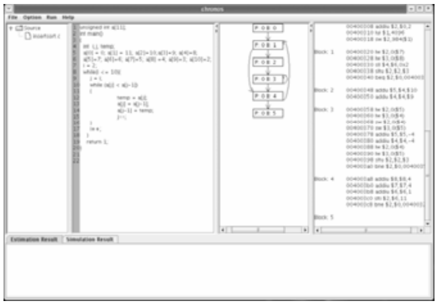
- WCET Analysis
  - Program flow modeling (typically by ILP)
    - Combine reasoning about timing of program fragments.
    - Exploiting Infeasible path information.
    - Difficult to use model checking for this purpose.
  - Micro-architectural modeling (customized analysis)
    - Exec. time of each instruction is not constant.
    - Worst-case not found by adding up worst-cases of code fragments – non compositional.
      - □ Efficient analysis developed to overcome this (not discussed).

Chronos Tool for WCET analysis



Copyright (c) 2009, Abhik Roychoudhury

Chronos – views of program

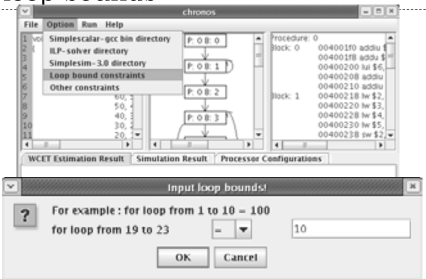


```
unsigned int a[11];
int main()
{
  int i,j, temp;
  a[0] = 0; a[1] = 11; a[2]=10;a[3]=9; a[4]=8;
  a[5]=7; a[6]=6; a[7]=5; a[8] =4; a[9]=3; a[10]=2;
  i = 2;
  while(i <= 10){
    j = i;
    while (a[j] < a[j-1])
    {
          temp = a[j];
          a[j] = a[j-1];
          a[j-1] = temp;
          j--;
    }
    i++;
  }
  return 1;
}
```

```
            00400108 addiu $2,$0,2
            00400110 lui $1,4096
            00400118 sw $2,984($1)

Block: 1    00400120 lw $2,0($7)
            00400128 lw $3,0($8)
            00400130 sll $4,$6,0x2
            00400138 sltu $2,$2,$3
            00400140 beq $2,$0,004003

Block: 2    00400148 addu $5,$4,$10
            00400150 addu $4,$4,$9

Block: 3    00400158 lw $2,0($5)
            00400160 lw $3,0($4)
            00400168 sw $2,0($4)
            00400170 sw $3,0($5)
            00400178 addiu $5,$5,-4
            00400180 addiu $4,$4,-4
            00400188 lw $2,0($4)
            00400190 lw $3,0($5)
            00400198 sltu $2,$2,$3
            004001a0 bne $2,$0,004003

Block: 4    004001a8 addiu $8,$8,4
            004001b0 addiu $7,$7,4
            004001b8 addiu $6,$6,1
            004001c0 slti $2,$6,11
            004001c8 bne $2,$0,00400

Block: 5
```

Copyright (c) 2009, Abhik Roychoudhury

Set loop bounds



```
Procedure: 0
Block: 0    004001f0 addiu $
            004001f8 addu $
            00400200 lui $6,
            00400208 addiu
            00400210 addiu
Block: 1    00400218 lw $2,
            00400220 lw $3,
            00400228 lw $4,
            00400230 lw $5,
            00400238 sw $2,
```

For example : for loop from 1 to 10 = 100

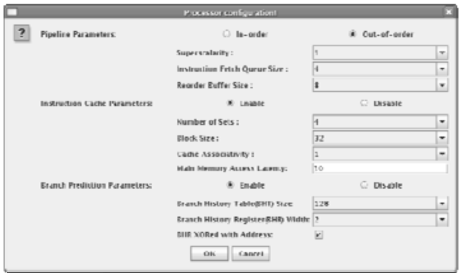
for loop from 19 to 23 = 10

Constraints specified by user at source code level.

These are automatically translated to node counts of assembly level control flow graph.

Copyright (c) 2009, Abhik Roychoudhury

Flexible micro-arch modeling



Pipeline Parameters: ○ In-order ● Out-of-order

Superscalarity : 1

Instruction Fetch Queue Size : 4

Reorder Buffer Size : 8

Instruction Cache Parameters: ● Enable ○ Disable

Number of Sets : 4

Block Size : 32

Cache Associativity : 1

Main Memory Access Latency: 10

Branch Prediction Parameters: ● Enable ○ Disable

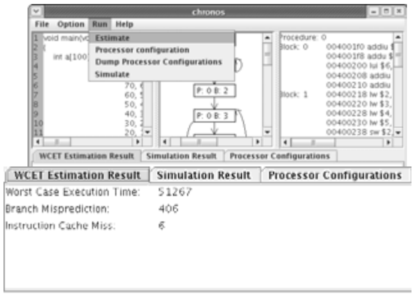
Branch History Table(BHT) Size: 128

Branch History Register(BHR) Width: 2

BHR XORed with Address: ☑

Copyright (c) 2009, Abhik Roychoudhury

WCET Estimation



```
void main(vo
{
   int a[100
```

```
Procedure: 0
Block: 0    004001f0 addiu $
            004001f8 addu $
            00400200 lui $6,
            00400208 addiu
            00400210 addiu
Block: 1    00400218 lw $2,
            00400220 lw $3,
            00400228 lw $4,
            00400230 lw $5,
            00400238 sw $2,
```

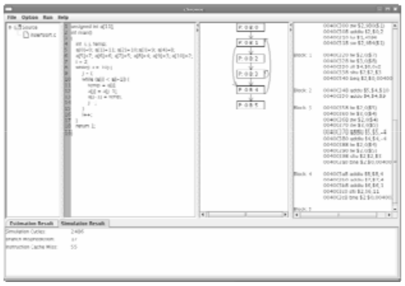
Worst Case Execution Time: 51267

Branch Misprediction: 406

Instruction Cache Miss: 6

Copyright (c) 2009, Abhik Roychoudhury

Simulation vs Estimation



Simulation Cycles: 2486

Branch Misprediction: [illegible]

Instruction Cache Miss: 55

Simulation produces Observed WCET.

Estimation produces Estimated WCET

Observed WCET ≤ Actual WCET ≤ Estimated WCET

Copyright (c) 2009, Abhik Roychoudhury