Static Race Detection for Periodic Programs

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

Joint work with Varsha Suresh (IIITB),
Rekha Pai (IISc/Oxford), Sujit Chakrabarti (IIITB) and
Meenakshi D'Souza (IIITB)

05 July 2022

Overview

A way to report data-races in real-time periodic programs, in a sound and precise manner.

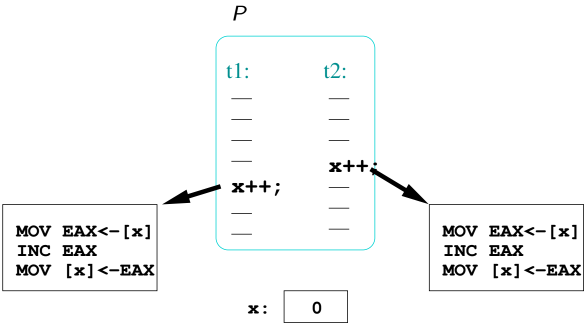
Contributions

- A way to estimate worst-case response time (WCRT) for programs with non-nested locks
- Disjoint-Block patterns that are useful in eliminating races.

Outline

Data Races

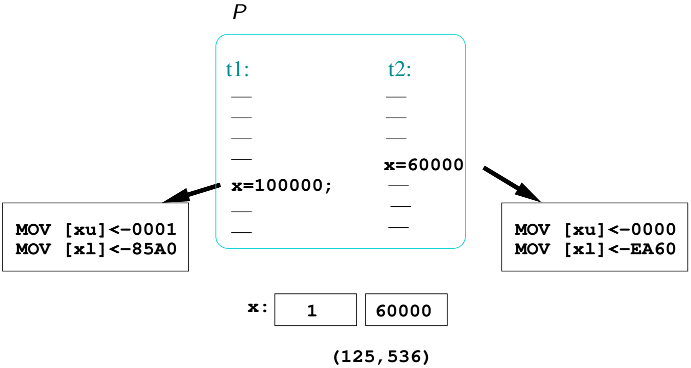
Statements in two different threads that are conflicting accesses and can happen-in-parallel.



One increment may get lost.

Data Race: Another example

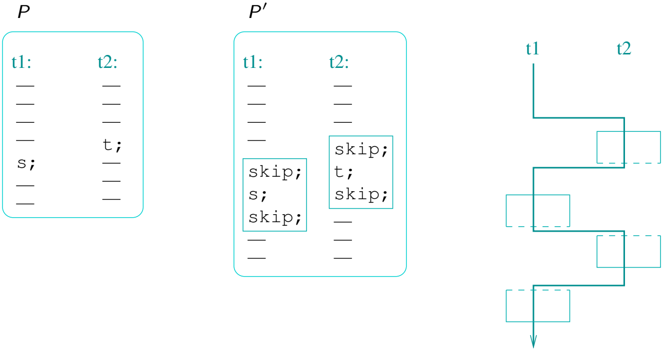
Statements in two different threads that are conflicting accesses and can happen-in-parallel.



x may get the value 125,536.

Data-Race Definition [Chopra, Pai, D. ESOP 2018]

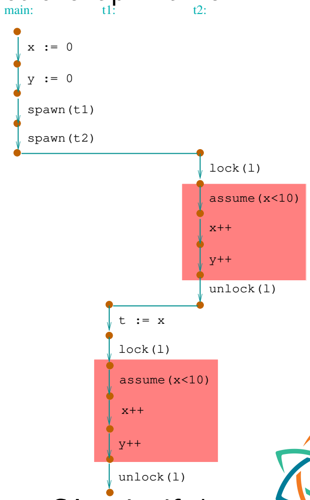
Notion of may-happen-in-parallel (MHP): Statements s and t in program P MHP if there is an execution of P' in which the notional blocks around s and t overlap in time.



Data-Race = Conflicting Access + MHP

Disjoint blocks with locks

Disjoint blocks: blocks of code which cannot overlap in time.



main:

```
1. x :=  y := 0;
2. spawn(t1);
3. spawn(t2);
```

t1:

```
0. t := 0;
1. lock(l);
2. if (x < 10)
3.   x++;
4. y++;
5. unlock(l);
```

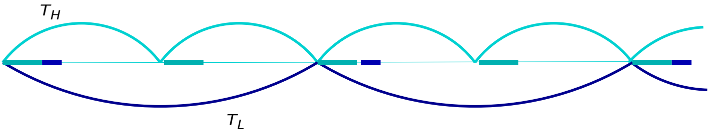
t2:

```
1. lock(l);
2. if (x < 10)
3.   x++;
4. y++;
5. unlock(l);
```

Technique for static race detection: Eliminate CA pairs if they are covered by pairs of disjoint blocks.

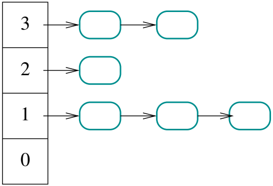
Periodic Program: Example

```
init:
1.  obstacle := 0;
2.  forward := 0;
3.  sIn := 0;
4.  start;

ObsDect: // Period = 100, Prio = 2
10.  obstacle := 0;
11.  if (sIn <= 10) {
12.    obstacle := 1;
13.    forward := -100;
14.  }

MoveForward: // Period = 200, Prio = 1
20.  if (!obstacle)
21.    forward := 100;
```



Execution Semantics

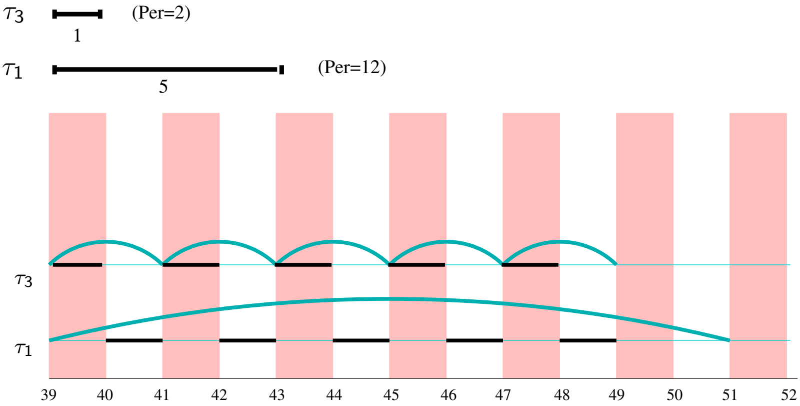
- Scheduler maintains a priority-wise FIFO Ready Queue
- Interrupted tasks are put back at the head of the queue.
- Tasks are moved from Delayed Queue to Ready Queue at multiples of their period.
- Program runs on a single processor.



Ready Queue

Response Time

Time that a task instance takes to complete after being made ready.



Worst Case Response Time (WCRT) is the largest response time over all instances of the task.

Computing WCRT without locks

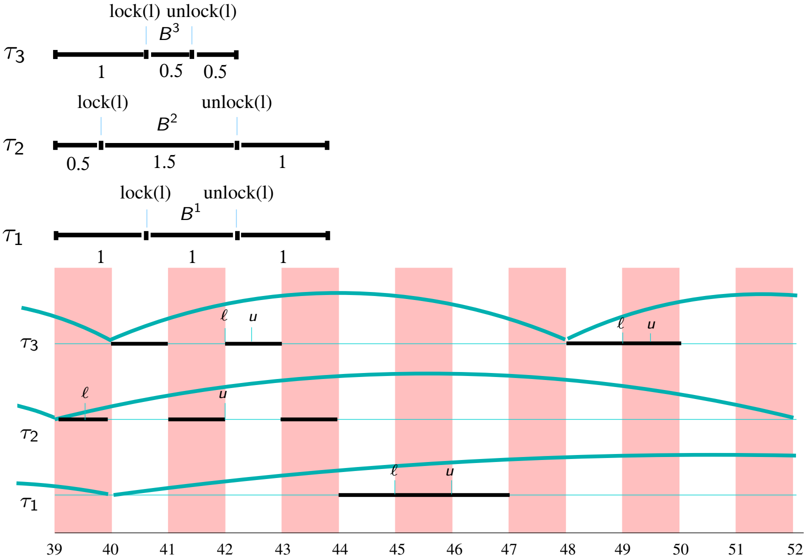
$$R_i = C_i + \sum_{j>i} (\lceil R_i / T_j \rceil \cdot C_j). \qquad (1)$$

Theorem (Joseph-Pandya-1982, Liu-Layland-1973)

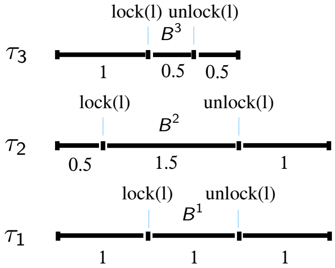
The least solution to Eq (1), whenever it exists, is an upper bound on the WCRT of task τ_i.

Simple iterative algo to compute WCRT.

Response Time with Non-Nested Locks



Computing WCRT with Non-Nested Locks



$$R_3 = C_3 + \max(U_l^2, U_l^1) \quad (2)$$

$$R_2 = C_2 + U_l^1 + \lceil R_2 / T_3 \rceil \cdot C_3 \quad (3)$$

$$R_1 = C_1 + \lceil R_1 / T_3 \rceil \cdot C_3 + \lceil R_1 / T_2 \rceil \cdot C_2 \quad (4)$$

$$U_l^2 = C_l^2 + \lceil U_l^2 / T_3 \rceil \cdot C_3 \quad (5)$$

$$U_l^1 = C_l^1 + \lceil U_l^1 / T_3 \rceil \cdot C_3 + \lceil U_l^1 / T_2 \rceil \cdot C_2 \quad (6)$$

Computing WCRT with Non-Nested Locks

$$R_i = C_i + \sum_{l \in L} (N_l^i \cdot \max_{j<i} U_{l,k}^j) + \sum_{j>i} (\lceil R_i / T_j \rceil \cdot C_j) \qquad (7)$$

$$U_{l,k}^i = C_{l,k}^i + \sum_{j>i} (\lceil U_{l,k}^i / T_j \rceil \cdot C_j) \qquad (8)$$

Theorem

The least solution to the system of Eqs (7,8), whenever it exists, is an upper bound on the corresponding WCRT of tasks τ_i and the blocks $B_{l,k}^i$.

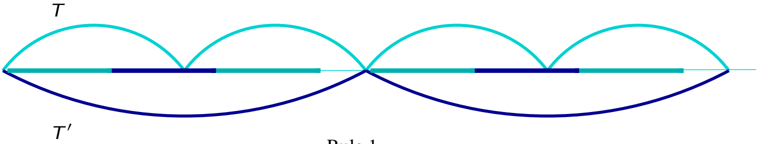
Simple iterative algo to compute WCRT.

Disjoint-Block Rules: Rule 1 (Same Priority)

Let τ and τ' be two distinct tasks in $\mathcal{T}$ such that:

- τ and τ' have the same priority (i.e. $p_\tau = p_{\tau'}$); and
- Neither τ nor τ' shares a lock with a lower priority task.

Then τ and τ' are disjoint.



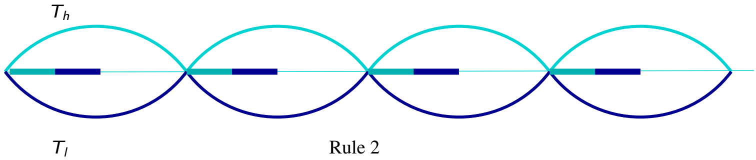
Rule 1

Disjoint-Block Rules: Rule 2 (Same Period)

Let τ and τ' be two distinct tasks in $\mathcal{T}$ such that:

- τ and τ' have the same period (i.e. $T_\tau = T_{\tau'}$); and
- Neither τ nor τ' shares a lock with a lower priority task.

Then τ and τ' are disjoint.

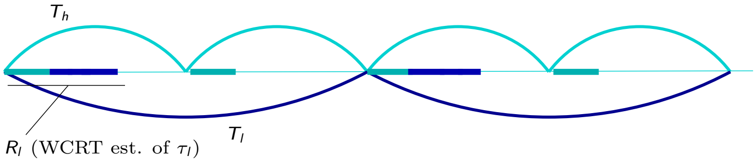


Disjoint-Block Rules: Rule 3 (Low-Multiple-of-High)

Let τ_l and τ_h be two tasks in $\mathcal{T}$ such that:

- τ_l has a lower priority than τ_h;
- The period of τ_l is a multiple of the period of τ_h;
- τ_h does not share a lock with a task of lower priority than τ_l; and
- The WCRT estimate R_{τ_l} of τ_l is at most the period of τ_h (i.e. $R_{\tau_l} \leq T_{\tau_h}$).

Then τ_l and τ_h are disjoint.



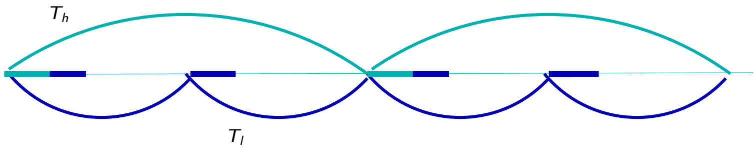
Rule 3

Disjoint-Block Rules: Rule 4 (High-Multiple-of-Low)

Let τ_l and τ_h be two tasks in $\mathcal{T}$ such that:

- τ_l has a lower priority than τ_h;
- The period of τ_h is a multiple of the period of τ_l; and
- τ_h does not share a lock with a task of lower priority than τ_l.

Then τ_l and τ_h are disjoint.



Rule 4

Disjoint-Block Rules: Rule 5 (Low-WCRT)

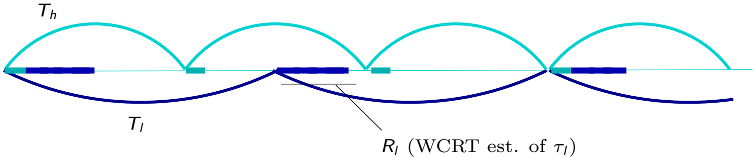
Let τ_l and τ_h be two tasks in $\mathcal{T}$ such that:

- τ_l has a lower priority than τ_h;
- Periods of τ_l and τ_h are not multiples of the other.
- τ_h does not share a lock with a task of lower priority than τ_l.
- Let m be the minimum *strictly positive* value in the set

$$\{(k \cdot T_{\tau_h}) \mod T_{\tau_l} \mid k \in \mathbb{N}\}.$$

The WCRT estimate R_{τ_l} of τ_l is at most m (i.e. $R_{\tau_l} \leq m$).

Then τ_l and τ_h are disjoint.



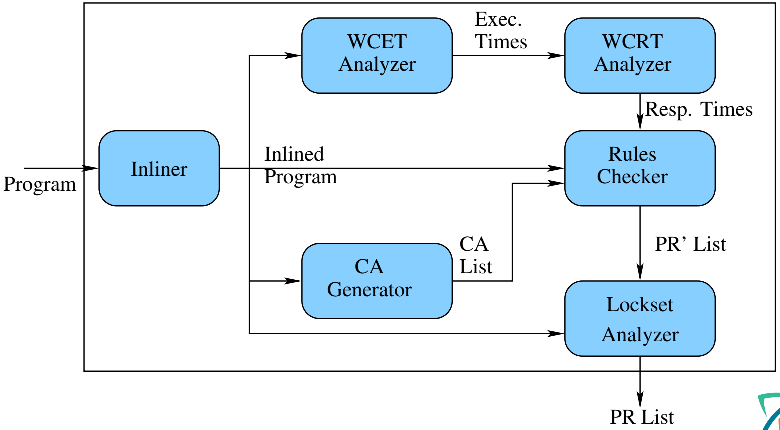
Rule 5

Race-Detection Algorithm

Given a periodic program P:

1. Compute list of conflicting accesses CA in P.
2. Set potential races $PR := CA$.
3. For each CA pair (s_1, s_2) in PR, if (s_1, s_2) is covered by a pair of disjoint blocks according to Rules 1–5, or the Lock-Rule; Remove (s_1, s_2) from PR.
4. Report PR as list of potential races in P.

Implementation



Experiments

Program	LoC	Tasks	Sched.	CA	PR	% Elim.	Time (sec)
fse_obstacle.c	24	2	Y	3	0	100	0.12
avionics.c	588	15	N	51	42	18	0.13
biped_robot.c	340	3	Y	1	0	100	0.22
sumo.c	5287	4	Y	146	0	100	0.32
nxtgt.c	209	4	Y	3	0	100	0.21
lego_osek.c	2036	2	Y	1320	0	100	0.12
objectfollower.c	1878	3	Y	14	0	100	0.31
nxtway_gs.c	2263	3	Y	4	0	100	0.37
car.c	1329	4	Y	670	0	100	0.28
ardupilot.c	1392	4	Y	17	0	100	0.24
follower.c	2769	7	Y	1179	0	100	0.30
sumoR.c	5287	4	Y	146	77	47	0.31
carR.c	1329	4	Y	670	125	81	0.28

Conclusion

- Given a way to compute WCRT for periodic programs with non-nested locks.
- Disjoint-Block (Not-MHP) patterns for periodic programs.
- Effective race-detection technique for these programs.

Future work:

- Handle immediate ceiling priority locks used in OSEK and other RTOSs.
- Data-Flow analysis for periodic programs.

Thanks for listening!