IIT CS536: Science of Programming

Homework 7: Parallelism

My Dinh

April 26, 2022

1 A simple parallel program

Task 1.1 (Written, 4 points).

i	j	Vars(j)	Change(i)	i interferes w/ j?
s_1	s_2	$i2, n, r2$	$i1, r1$	No
s_2	s_1	$i1, r1, n$	$i2, r2$	No

From the table above, we know that s_1 and s_2 are disjoint. Thus, s is a disjoint parallel program.

Task 1.2 (Written, 7 points).

$\{n \geq 0\}$

$i1 := \overline{0};$

$r1 := \overline{0};$ $\{n \geq 0 \wedge i1 = 0 \wedge r1 = 0\}$

$\{\textbf{inv}\ i1 \leq n/2 \wedge r1 = sum(0, i1)\}$

while $(i1 < n/\overline{2})$ {

  $r1 := r1 + i1;$

  $i1 := i1 + \overline{1}$

}; $\{r1 = sum(0, n/2)\}$

$i2 := n/\overline{2};$

$r2 := \overline{0};$ $\{r1 = sum(0, n/2) \wedge i2 = n/2 \wedge r2 = 0\}$

$\{\textbf{inv}\ i2 \leq n \wedge r2 = sum(n/2, i2) \wedge r1 = sum(0, n/2)\}$

while $(i2 < n)$ {

  $r2 := r2 + i2;$

  $i2 := i2 + \overline{1};$

}; $\{r1 = sum(0, n/2) \wedge r2 = sum(n/2, n)\}$

$\Rightarrow \{r1 + r2 = sum(0, n)\}$

$r := r1 + r2$

$\{r = sum(0, n)\}$

Task 1.3 (Written, 11 points).

Before we can use **Par rule** to prove this program, we have to prove that thread s_1 and s_2 in this program have disjoint conditions.

i	j	Change(i)	Vars(j)	FV(j)	i interferes w/ j?	i interferes w/ cond j?
s_1	s_2	$r1, i1$	$r2, i2, n$	$r2, i2, n$	No	No
s_2	s_1	$r2, i2$	$i1, r1, n$	$i1, r1, n$	No	No

Thus, s_1 and s_2 have disjoint conditions.

	$\{n \geq 0\}$
[	$\{p_1 \equiv n \geq 0\}$
$i1 := \overline{0}$;	
$r1 := \overline{0}$;	$\{n \geq 0 \wedge i1 = 0 \wedge r1 = 0\}$
$\{\textbf{inv}\ i1 \leq n/2 \wedge r1 = sum(0, i1)\}$	
while $(i1 < n/\overline{2})$ {	
$r1 := r1 + i1$;	
$i1 := i1 + \overline{1}$	
}	$\{q_1 \equiv r1 = sum(0, n/2)\}$
||	$\{p_2 \equiv n \geq 0\}$
$i2 := n/\overline{2}$;	
$r2 := \overline{0}$;	$\{n \geq 0 \wedge i2 = n/2 \wedge r2 = 0\}$
$\{\textbf{inv}\ i2 \leq n \wedge r2 = sum(n/2, i2)\}$	
while $(i2 < n)$ {	
$r2 := r2 + i2$;	
$i2 := i2 + \overline{1}$;	
}	$\{q_2 \equiv r2 = sum(n/2, n)\}$
];	$\{r_1 = sum(0, n/2) \wedge r_2 = sum(n/2, n)\} \Rightarrow \{r1 + r2 = sum(0, n)\}$
$r := r1 + r2$	
	$\{r = sum(0, n)\}$

2 A More Realistic Parallel Program

Task 2.1 (Written, 10 points).

Proof outline for s_1:

	$\{n \geq 0 \wedge i1 = 0 \wedge r1 = 0\}$
$\{\textbf{inv}\ i1 \leq n/2 \wedge r1 = sum(0, i1)\}$	
while $(i1 < n/\overline{2})$ {	$\{i1 < n/2 \wedge i1 \leq n/2 \wedge r1 = sum(0, i1)\} \Rightarrow \{i1 < n/2 \wedge r1 = sum(0, i1)\}$
$< r1 := r1 + i1$;	$\{i1 < n/2 \wedge r1 = sum(0, i1) + i1\} \Rightarrow \{i1 + 1 < n/2 + 1 \wedge r1 = sum(0, i1 + 1)\}$
$i1 := i1 + \overline{1} >$	$\{i1 < n/2 + 1 \wedge r1 = sum(0, i1)\} \Rightarrow \{i1 \leq n/2 \wedge r1 = sum(0, i1)\}$
}	$\{i1 \geq n/2 \wedge i \leq n/2 \wedge r1 = sum(0, i1)\}$
	$\Rightarrow \{r1 = sum(0, n/2)\}$

Proof outline for s_2:

	$\{0 \leq i1 \leq n/2 \wedge i2 = n/2 \wedge r1 = sum(0, i1) \wedge r2 = 0\}$
$\{\textbf{inv}\ i1 \leq n/2 \wedge i2 \leq n \wedge r1 = sum(0, i1) \wedge$	
$r2 = sum(n/2, i2)\}$	
while $(i2 < n)$ {	$\{i2 < n \wedge i1 \leq n/2 \wedge i2 \leq n \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2)\}$
	$\Rightarrow \{i2 < n \wedge i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2)\}$
$r2 := r2 + i2$;	$\{i2 + 1 < n + 1 \wedge i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2 + 1)\}$
$i2 := i2 + \overline{1}$	$\{i2 < n + 1 \wedge i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2)\}$
	$\Rightarrow \{i2 \leq n \wedge i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2)\}$
};	$\{i2 \geq n \wedge i2 \leq n \wedge i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, i2)\}$
	$\Rightarrow \{i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, n)\}$
$\{\textbf{inv}\ i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, n)\}$	
while $(i1 < n/\overline{2})$ {skip};	$\{r1 = sum(0, n/2) \wedge r2 = sum(n/2, n)\}$
	$\Rightarrow \{r1 + r2 = sum(0, n)\}$
$r := r1 + r2$	$\{r1 + r2 = sum(0, n) \wedge r = r1 + r2\}$
	$\Rightarrow \{r = sum(0, n)\}$

Task 2.2 (Written, 6 points).

Because s_1 and s_2 are disjoint programs and have disjoint conditions so the proof s_1* and s_2* does not interfere with each other. We can see that proof s_1* reads $i1, r1$, n and writes $i1, r1$ that do not change the conditions and proof invariants in proof s_2* (because the changes are kept in a critical section). And proof s_2* reads $r1, i1, r2, i2, n$ and writes $s2, i2, r$ that do not interfere proof s_1*.

3 Even More Realistic with Await

Task 3.1 (Written, 3 points).

```
[
while (i1 < n/2) {
   ⟨r1 := r1 + i1;
   i1 := i1 + 1⟩
}
||
while (i2 < n) {
   r2 := r2 + i2;
   i2 := i2 + 1
};
await (i1 ≥ n/2) then {r := r1 + r2}
]
```

Task 3.2 (Written, 2 points).

Because a while loop might keep running in one thread that prevents other thread from running, which costs a lot of resources. It can also cause the program to diverge and make the total correctness of no longer hold. While await check the condition, if it false then it will let other threads run. It only runs the statement when the condition is true.

4 A Buggy (and therefore even more realistic) Parallel Program

Task 4.1 (Written, 7 points).

a) With the given precondition and initial state, program s can execute thread 2 until the end without having to wait for thread 1 to finish to compute the final sum r. When it reaches while $i1 < n/2$ {skip} in thread 2, because thread 1 hasn't run yet, the value of $i1 = 8285$ (from initial state σ) is larger than $n/2$ so the program in thread 2 exits the loops. At this point, if thread 2 computes $r = r1 + r2$ before thread 1 can end, the value of r is either 524999 ($r = 523752 + sum(13, 26)$ for $r1 = 523752$ from state σ) or from $sum(0, 0) + sum(13, 26)$ to $sum(0, 12) + sum(13, 26)$ (thread 2 interleaves to thread 1 then back to thread 2).

b) In hint 1 given in part a), there are 78 execution paths that will have this behavior (incorrect paths). Thus, the expected number of times we would have to run the program to find the bug in testing is

$$\frac{\#\text{ of possible execution paths}}{\#\text{ of incorrect execution paths}} = \frac{1.3 \times 10^{14}}{78} = 1.67 \times 10^{12} \text{ times}$$

The total time it would take to find one of the buggy execution described in part a) is

$$\frac{1.67 \times 10^{12}}{1000} = 1.67 \times 10^{9} \text{ seconds} \approx 52.85 \text{ years}$$

c) The total time an user will encounter the bug, in expectation, is

$$\frac{1.67 \times 10^{12}}{10^{11}} = 16.7 \text{ days}$$

d) Because program verification helps us make sure that our program will work correctly as we've intended it to do and avoid possible errors in the program. In the above example, without program verification, even though the error/bug does not affect us right away, it might appear in the future and cause the users and developers some troubles.

5 One more wrap-up question

Task 5.1 (Written, 0 points).

I spent about 3 hours on this homework, in total 2 hours of actual working time.