Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 4: Parallel Models
Spring Semester 2020

Assigned on: **11.03.2020**

Due by: **(Wednesday Exercise) 16.03.2020**
(Friday Exercise) 18.03.2020

Task 1 – Pipelining

Bob, Mary, John and Alice share a flat. In this flat they share a washing machine, a dryer and an ironing board. The washing machine takes 50 minutes for one wash cycle. The dryer takes 90 minutes. Everyone of them takes roughly 15 minutes to iron their laundry.

a) Assuming they would do their laundry in strictly sequential order (one person starts only after the other finished ironing), calculate how long would it take to finish the laundry.

Answer: They would take $(50 + 90 + 15) * 4 = 620$ minutes as illustrated in Figure 1.

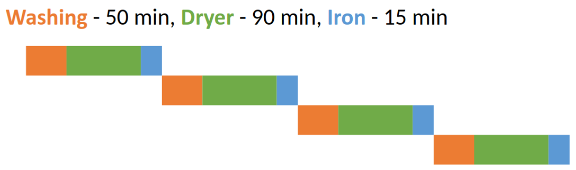


Figure 1: Pipeline executed strictly in sequential order.

b) Are there any better options? If yes, can you describe them and calculate the improved laundry time? Further, determine whether this pipeline is balanced or unbalanced.

Answer: `(50 + (90 * 4) + 15) minutes = 425 Minutes`
Assuming that three tasks are going in parallel, the speedup is: $S_3 = T_1/T_3 = 620/425 = 1.46$ as illustrated in Figure 2. The pipeline is unbalanced as after washing we have to wait until the dryer is available.

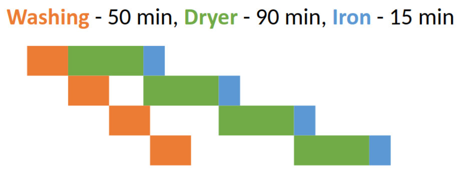


Figure 2: Pipeline executed in parallel.

c) Can you devise a better strategy assuming that the four roommates bought another dryer? If yes, calculate the new laundry time. Further, determine whether such an improved pipeline is balanced or unbalanced and calculate the pipeline throughput as well as latency.

Answer: With two dryers, the third person in order can start drying as soon as their washing is done as shown in Figure 3. This brings the total time to $(50) * 4 + 90 + 15 = 305$ minutes. The pipeline is balanced. The latency of the pipeline is the sum of the execution times of each stage, that is, $50 + 90 + 15 = 155$ min. The throughput of the pipeline is the amount of work that can be done by a system in a given period of time. In our case, the 1st person finished the laundry after 155 min, the 2nd person after 205 min, the 3rd after 255 min and 4th after 305 min. Therefore, the throughput is 1 person per 50 min.

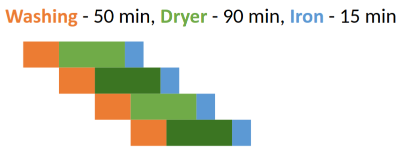
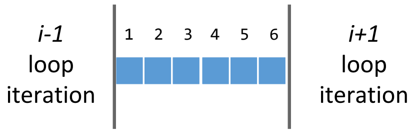


Figure 3: Pipeline with two dryers.

Task 2 – Pipelining II

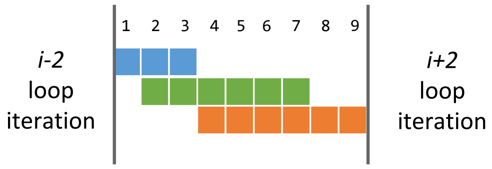
a) Each loop body has to perform one multiplication instruction before we can move to the next loop iteration. Since the multiplication instruction has a latency of 6 cycles and we need to wait until it fully finishes, the whole loop will require `data.length * 6` cycles. In the following, we assume `int size = data.length`.

```
   for (int i = 0; i < size; i++) {
🟦   data[i] = data[i] * data[i];
   }
```



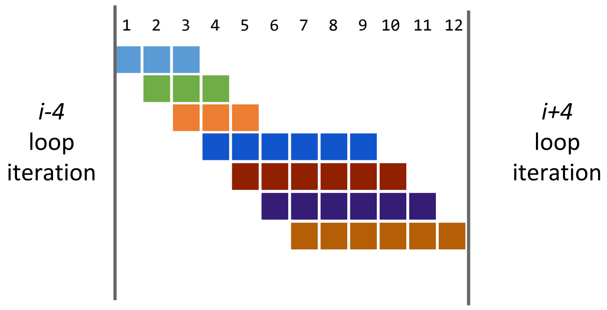
b) In the first cycle the computation of the addition `i + 1` is issued. In the next cycle, the multiplication `data[i] * data[i]` can be issued as there is no dependency on the value of `j` that is still being computed. However, the last multiplication needs to wait until the execution of the addition is finished as it uses the value `j` to index into the array. The total execution time is `data.length * 9 / 2` cycles as illustrated below:

```
   for (int i = 0; i < size; i += 2) {
🟦   j = i + 1;
🟩   data[i] = data[i] * data[i];
🟧   data[j] = data[j] * data[j];
   }
```



c) In the last example we simply execute all instructions, issuing one each cycle. This is possible as unrolling the loop four times provides enough work to hide the dependency between the index computation (via addition) and the corresponding multiplication. The total execution time is therefore `data.length * 12 / 4` cycles as illustrated below:

```
   for (int i = 0; i < size; i += 4) {
🟦   j = i + 1;
🟩   k = i + 2;
🟧   l = i + 3;
🟦   data[i] = data[i] * data[i];
🟥   data[j] = data[j] * data[j];
🟪   data[k] = data[k] * data[k];
🟫   data[l] = data[l] * data[l];
   }
```



Task 3 – Identify potential parallelism

a) Inspect the code snippets for the two loops below. For each loop explain if it is possible to parallelize the loop, and how would you do it.

(a) Loop-1

```
for (int i=1; i<size; i++) {     // for loop: i from 1 to (size-1)
    if (data[i-1] > 0)           // If the previous value is positive
        data[i] = (-1)*data[i];  // change the sign of this value
}                                // end for loop
```

Answer: This loop can not be parallelized as each iteration depends on the answer of the previous iteration:

```
if (data[i-1] > 0)
```

Here the computation of current `data[i]` depends on the value of `data[i-1]` which will be available only after the previous iteration has finished. Due to this dependency, we can't parallelize the loop.

(b) Loop-2

```
for (int i=0; i<size; i++) {       // for loop: i from 0 to (size-1)
    data[i] = Math.sin(data[i]);   // calculate sin() of the value
}                                  // end for loop
```

Answer: In this loop, there is no dependency on any previous loop computations, so we can compute each iteration of the loop in parallel.