Concurrent Programming
CS:-431 Programming Languages Lab

Nikit Begwani(130101055) Rohan Gupta (130101066)
Group - 8

September 11, 2016

1 Benefits of Having Concurrency

Sequential programs have one thread of control while on the other hand concurrent programs can run many threads for having logically simultaneous processing. With concurrency , modeling of the problem becomes easier by assigning specific tasks to specific threads which needs to be done by them. Concurrency in the given problem helps in isolating and simplifying the task. If we imagine a sequential solution to the given problem, it will be like a single thread(or process) running each step one by one. Like, first it will generate 10 data (one for each sensor), then it converts the data of each sensor into integer value (again 10 steps) , then it does the function of add, avg, multiplication which takes a lot of time and by this time, other work remains at halt.

Now, if we imagine a concurrent solution with the machine having more than one processor then, we can have 10 threads (representing the 10 sensors) whose job is to generate binary valued 8-bit data and store it in space assigned to them.As the work of these threads are just to generate data so, they keep on doing independently and we can give them access to only those objects which are required by them while others can be set free and used for other tasks. Now comes the converters , the benefit of using individual converters is that , none of them have to wait for other converter while doing their work because each of them have access to their individual queue. The benefit of using concurrency for functions is that all the three functions work independently and suppose if the data is not ready for one of the function, it simply leaves the lock, puts itself to halt and the other function may perform its duties.

The program becomes more responsive by keeping everything in an organized manner and giving them access to just the specifics required by them.

2 Concurrent Solution For The Problem

We have tried to use as much concurrency as possible for the problem, by keeping everything separate and independent. The proposed architecture contains:-

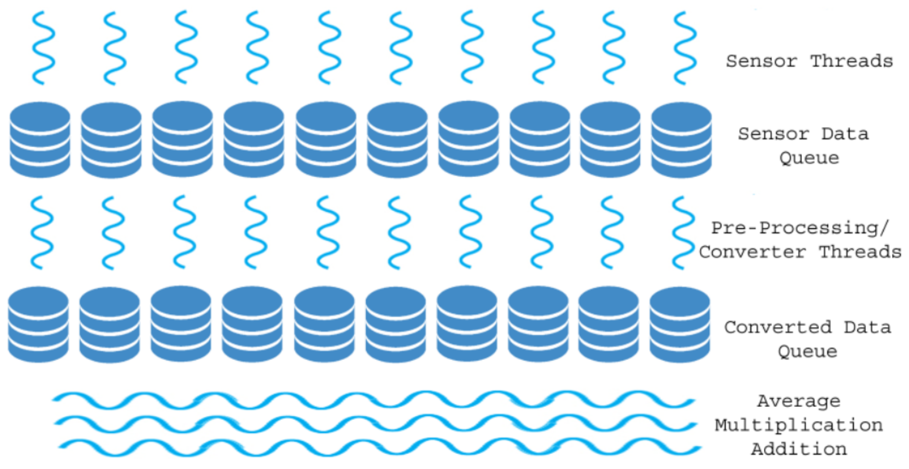

Figure 1: Concurrent Architecture

- **Sensors-** These are the 10 threads responsible for generating the binary valued data for each of the 10 sensors independently and storing them in respective Sensor Data Queue.
- **Converters-** These are the 10 threads responsible for converting the binary valued data to integer from each of the 10 Sensor Data Queue and storing them into Converted Data Queue.
- **Functions-** We create 3 threads one for each function, the collect data from each Converted Data Queue and perform addition, average and multiplication depending upon the task assigned.
- **Sensor Data Queue-** We create 10 queue one for each of the sensor to store their data so that many threads can run simultaneously.
- **Converted Data Queue-** We create 10 queue one for each of the converter to store their converted data so that many threads can run simultaneously.

Figure 1 shows the architecture in detail. We use synchronization or locks in respective implementation to control the data access. We represent each object of the above diagram using new instance of classes. Due to this independence, we get a very good performance response and high concurrency.

3 Fork-Join Framework

For the second question of the assignment, we include the fork-join framework in second implementation of first part. In the pre-processing stage, before each function uses the data we sort it by using this framework. In fork-join, we define a class named divide task which takes the array to sort as an input and using invoke method of fork-joinpool on task we begin the sorting. For each task, we further divide it into two tasks and wait for them to finish after that using the mergedArrays method we merge both parts of parent array. Fork-join employs worker-stealing algorithm where in idle threads can pick subtasks created by other active tasks and execute them. In this way there is efficient execution of tasks spawned by other tasks.

4 Calculator

Packages used:

1. **Swing** for implementation of UI
2. **SwingWoker** for concurrency
3. **Java Keylistner** for keyboard events

Implementation:

1. The interface is created using a **JFrame object** of Swing. JButton and Jtextfield classes are used for buttons and text area.
2. Periodic highlighting of numbers and functions are done in seperate threads from the main Event Dispatch thread. Multithreading is done using the **SwingWorker** Class.
3. Standard **Java KeyListner iterface** is implemented for key press events on the main class. This keylistner is added to the JFrame object and hence runs on the main Event Dispatch Thread.
4. Logic is implemented using different **states of input**, program moves from one state to another as the user provides the input.
5. **Java BigDecimal** objects are used to handle input numbers.

Multi Threading:

- Buttons need to be highlighted after specific time intevals, the timer cannot run on the Event Dispatch Thread as the UI will become unresponsive in between, therefore the timer for function and number updates are run on seperate threads.
- SwingWorker is a multi threading utility class for the Swing library. It enables proper use of the event dispatching thread. Therefore it is used to for the above mentioned task.
- NOTE: Instead of running the timer, thread is temorarily suspended for specified time which is more computationally efficient than a timer.