CSC2002S Tutorial 1

Digital Signal Processing with the Java Fork/Join framework

Radar (**RA**dio **D**etection **A**nd **R**anging) is an echo location system that uses a short pulse of energy to reflected off (and hence detect) objects in the surroundings. The simplest form of Radar consists of a transmitter and a receiver: the transmitter emits radio impulses at regular intervals and the receiver scans a narrow band of the sky (Figure 1). Radio waves sent by the transmitter propagate at the speed of light and reflect off objects of adequate size. Reflected waves propagate back to the receiver, at which point the round trip time and the distance to the reflecting object can be estimated. Unfortunately, it is not quite this simple, as the received signal is both attenuated (made fainter) by the atmosphere it passes through and polluted with noise from Electromagnetic Interference (EMI).

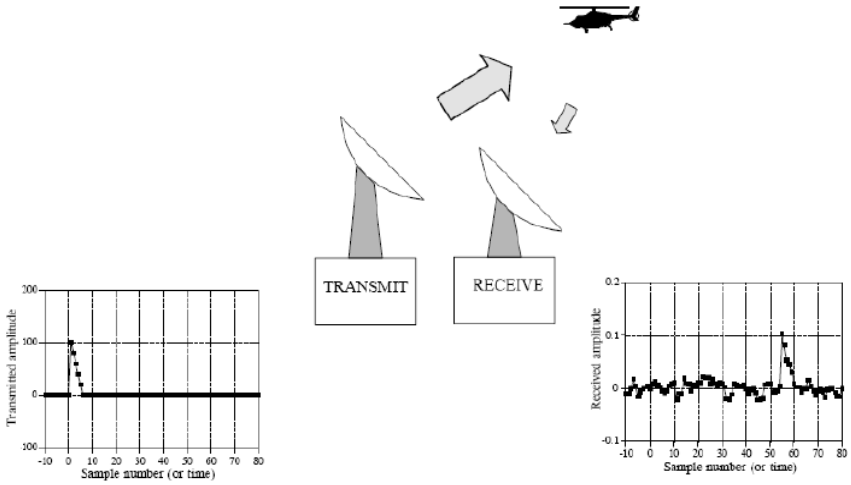

Figure 1: Radar transmission and reception. The transmitted signal and received signal differ in both phase (time delay) and amplitude (due to attenuation, absorption and noise pollution).

The detection problem is this: given a transmitted signal (t) of known shape, how do we find that (shifted and scaled) signal in the received signal (x)? The answer is found using a fairly simple mathematical operation called *correlation*, where the transmitted and received signals are combined to produce a third signal, called the *cross-correlation* (Figure 2). In this process, the received signal, x, is shifted in steps along the time axis. For each shift, the transmitted signal, t, and the shifted, noisy and attenuated received signal, x ,are point-wise multiplied and then added, to obtain the cross-correlation at the point $y[n]$. The serial algorithm is as follows.

```
for (int i = 0; i < length of x; ++i){
   float sum = 0;
   for (int j = 0; j < length of t; ++j){
     if (i+j < length of x)  // check that we do not index out of the boundary
        sum += x[i+j] * t[j];
   }
   y[i] = sum;}
```

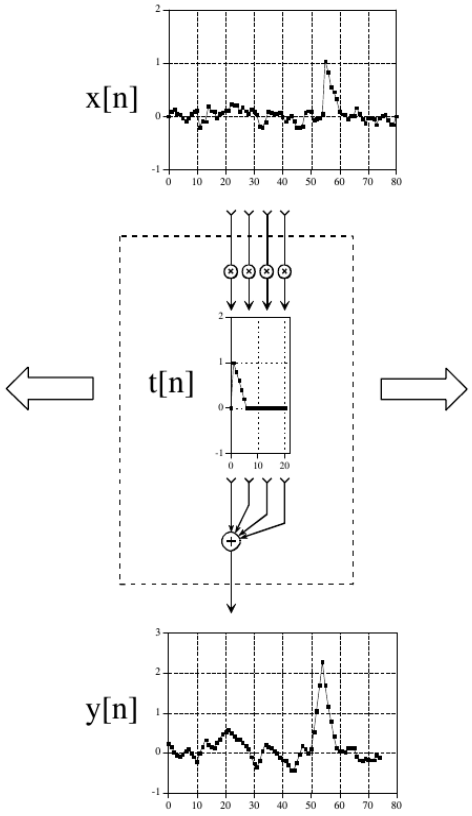

Figure 2: The correlation machine. The cross-correlation signal, y, is calculated from the transmitted signal, t, and the received signal, x . In each step, the dashed box is shifted along. The samples from x indicated by the dashed box are multiplied by the corresponding samples in t, and the products added to produce the indicated sample, y[n]. [Image taken from page 139 of *The Scientist and Engineer's Guide to Digital Signal Processing, Steven W. Smith.* Freely available online at: `http://www.dspguide.com/`]

There is then one more step to the algorithm. The value of y[n] indicates how much the signal x resembles t *at the point n.* Therefore, we want to find the point at which y is maximum, to locate the signal t in the signal n and thus compute the shift.

1 The assignment

The cross-correlation computation can be quite expensive, especially if the two input signals are of equal length (at which point it becomes quadratic). However, both the loops and the finding the maximum components of this algorithm are easily parallelized, which can bring the computation time down by a constant factor!

Your job in this assignment is to:

- Use the Java pooled Fork/Join framework to parallelize both the cross correlation and finding the maximum operations.

- Use high precision timing to evaluate the run times of the two methods an then optimise your approach.
- Experiment to find suitable limits at which sequential processing should begin.

1.1 Input and Output

We will provide you with[1] both a crisp simulated transmission signal and a noisy received signal (both lists of floating-point values separated by spaces). In each case, the first number in the file is an integer indicating the number of floating-point numbers in the file. We will also give you a Python script to generate longer signals yourself (you will need to install the SciPy[2] suite of packages on your machine to do so).

Your program should accept the filenames of the transmission and received signal files as command-line arguments. In terms of output, you will obviously need to calculate the correlation and the maximum point for the correlation, as well as timings for the sections of the code, but how you output/display these is up to you (and should be reported on). You will also need to consider validation - how do you know that your program is doing the correct thing? How can you show that it is correct? You may consider using an external package like JFreeChart to plot these signals and so visualize what your program is doing.

1.2 Report

Your clear and concise report on the project should contain the following:

- An Introduction, comprising a short description of the problem.
- A"Methods" section, giving a description of your approach to the solution, with details on the parallelization, the output, how you validated your algorithm, how your profiled your algorithm, the machine architectures you tested the code on and any problems encountered.
- A "Results and Discussion" section, demonstrating the parallel speedup, including graphs and and a discussion of the sequential limits.
- A "Conclusions" section.

Please do NOT ask for recommended numbers of pages. Say what you need to say, no more, no less.

[1] check "Resources" on Vula

[2] See `http://www.scipy.org/` for more details

1.3 Handin and marking

Your submission must consist of BOTH a technical report and your solution code (including a Makefile for compilation). You make will be based primarily on your analysis and investigation. The usual late penalties of 10% a day (or part thereof) apply.

Due date: Monday 11 August 2014, 8am
