THE USE OF CONCURRENT CODES IN COMPUTER PROGRAMMING AND DIGITAL SIGNAL PROCESSING EDUCATION

William Bahn, Leemon Baird, and Michael Collins
Department of Computer Science
United States Air Force Academy
USAFA, CO 80840
719 333-8782
William.Bahn@usafa.af.mil

ABSTRACT

Introducing relevant and meaningful real-world applications in exercises intended to teach traditional undergraduate topics is difficult; limitations on depth and scope can result in only a passing relationship between application and exercise. However, the recent development of concurrent coding theory offers one opportunity to achieve this because the underlying problem is readily understood and the algorithms are tractable at the introductory level. Furthermore, no exotic or expensive hardware is needed to construct functioning implementations – a computer with a soundcard, microphone, and speaker is sufficient. An entire first semester programming course can be built around concurrent codes and significant lab exercises can be constructed for an introductory digital signal processing course. As a result, students gain an appreciation for a real-world problem of growing importance including insight into how that problem can be addressed while maintaining a focus on the primary material being taught.

INTRODUCTION

Historically, introductory material in science and engineering curricula is presented in a "bottom up" fashion. Students first develop a wide set of narrow skills and then combine those skills to solve progressively more complex problems. Unfortunately, the early disjointedness can interfere with a student's impression of the relevance of their coursework and lead to frustration and even departure from the program.

Alternatively, a "top down" approach often integrates real-world problems from the beginning that are intended to provide a context for presenting the educational topics. Unfortunately, depth and scope limitations often relegate the "real world" aspects to little more than buzzwords. For example, one popular computer programming text attempts to link course material to several "engineering challenges" facing society such as designing large aircraft to be more fuel efficient. A typical exercise emphasizes that designers develop computer programs to analyze wind tunnel test data relating lift to angle-of-attack. Students, reasonably expecting to gain insight into the analysis of such data, are disappointed when the actual exercise only involves sorting a list of numbers. Seen as "bait and switch," students can become annoyed and are likely to further lose sight of what relevance the exercises do have.

Real-world problems suitable for a top-down approach need substance that can be explored using only the tools and techniques available to lower division engineering students. This allows students to work exercises that require comprehension of the real-world problem, thereby enhancing relevance. Finding such problems in a world of ever

This work was sponsored in part by the Air Force Information Operations Center (AFOIC), Lackland AFB, TX, and was performed principally at the Academy Center for Information Security (ACIS) at the United States Air Force Academy.

increasing technological complexity can be difficult. Surprisingly, the recently developed concurrent coding theory provides a suitable real-world problem: Keyless omnidirectional jam resistant (KOJR) communication systems. Although real-world applications are in the radio-frequency domain, image and audio representations serve as suitable classroom analogs.

This paper outlines two top-down approaches, one suitable for a full semester programming course and one as a module in a digital signal processing course. Either can use, or even be centered on, an open source tool developed at the Air Force Academy.

CONCURRENT CODES AND THE BBC ALGORITHMS

Concurrent codes allow the extraction of all possible messages consistent with the received data. Unfortunately, space limitations prevent a meaningful discussion of omnidirectional jam resistance, concurrent codes, or how they address the growing need for KOJR systems; the interested reader is directed to the original technical report [1]. Only the mechanics of the BBC encoding and decoding algorithms are presented here.

To form a codeword from an m-bit message, first choose an expansion ratio, e, and a checksum length, k. Then start with an empty (all 0-bits) c-bit codeword where c = m*e. The original message is then padded by appending k zeros to it. Finally, the set of (m+k) prefixes of the padded message is mapped to locations within the codeword using a suitable hash function and marks (1-bits) placed at those locations. For example, say m=3, k=1, and e=6. To generate the 18-bit codeword for message 011, the padded message is 0110 and the set of prefixes is {0, 01, 011, 0110}. Using the example hash function provided below, these prefixes map to locations {11, 14, 2, 16} defining the codeword 001000000001001010 (assuming bit 0 is the far left-hand bit). Similarly, the codeword 000001010001000100 represents the message 001 (locations {11, 7, 15, 5}).

```
Example Hash Function, H(x)
H(0)    = 11 H(1)    = 02
H(00)   = 07 H(01)   = 14 H(10)   = 03 H(11)   = 15
H(000)  = 13 H(001)  = 15 H(010)  = 07 H(011)  = 02
H(100)  = 10 H(101)  = 08 H(110)  = 16 H(111)  = 05
H(0000) = 00 H(0001) = 01 H(0010) = 05 H(0011) = 12
H(0100) = 04 H(0101) = 13 H(0110) = 16 H(0111) = 17
H(1000) = 12 H(1001) = 10 H(1010) = 03 H(1011) = 16
H(1100) = 15 H(1101) = 03 H(1110) = 10 H(1111) = 12
```

Assume that the two codewords above are sent by two transmitters and arrive at the receiver at the same time. There they combine via a logical-OR (certain modulation schemes, such as On-Off Keying, can combine this way). The received packet would then be 001001010001001110 (locations {2, 5, 7, 11, 14, 15, 16}).

Decoding a packet also proceeds bit by bit. If any messages are in the packet, they must start with either a 0 or a 1. Hence if the packet contains marks at H(0) or H(1), then the corresponding message prefixes are added to a message list. In this case, both marks are present resulting in the initial message list {0, 1}. Since the next bit must also either be a 0 or a 1, the process is repeated for each prefix in the list. After doing this for the second bit, the list is {00, 01, 11}, while after the third bit it is {001, 010, 011, 110, 111}. Notice that both of the original messages, 011 and 001, are in the list but so are several others. False messages resulting from the combination of actual messages are called "hallucinations" and the appended 0-bits deal with them; only the case of an appended 0-bit must be checked (it is known that no 1-bits were appended). The final message list is then {001, 011, 110}, which includes the two actual messages and one hallucination (which a second checksum bit would probably have eliminated).

STUDENT EXERCISES

Concurrent codes lend themselves to student exercises in either computer programming or digital signal processing. The BBC application (see the "BBC Application" section) can be used to support the course either by performing those steps that are outside the scope of the course or by letting the student work with a completely functioning suite of tools that they incrementally replace with their own code and/or processing blocks.

Introductory Computer Programming (CS1) Course

In the context of a semester course in computer programming, an entire syllabus can be devised where nearly every example and exercise is an incremental step in the development of a significant final program that performs a narrow subset of the BBC application's capabilities. While most programming languages can be used, ANSI-C compliant code is the framework for this discussion. Following is one possible sequence of exercises, and the topics introduced with each one, that could be considered.

1) Problem introduction using pencil-and-paper examples.
   Boolean logic, flowcharts, algorithmic thinking.
2) Using the BBC application to work real-sized problems end-to-end.
   Running console applications and any others (e.g. MS Paint) that will be used.
3) Implement a very simple command-prompt environment.
   Console I/O, string operations and simple parsing, selection statements, loops.
4) Encode messages entered using a fixed (hard-coded) codec configuration.
   Simple bit-level operations, text file I/O, array operations, hash functions.
5) Configure the codec via command line, keyboard, and/or configuration file.
   Command line processing, simple script processing.
6) Decode message packets.
   Recursion, dynamic memory allocation, data structures,
   linked-lists, depth-first searches.
7) Represent message packets as Windows BMP files.
   Binary file output, standard file formats.
8) Extract packets from Windows BMP files.
   Binary file input.

Students do not perform any digital signal processing when working with image files. Instructors wishing to work with audio files can supply BBC scripts that perform the DSP tasks. In fact, the BBC application can perform any tasks that students have not yet implemented. As a result, from the very first exercise, students have a complete working system that they may continually use to verify their own code's functionality. In addition, since exercises are progressive, students are motivated to avoid the common mistake of forgetting an exercise the moment it is turned in.

Finally, some of the topics may be beyond the desired scope of the course or be introduced earlier than desired. The instructor can reorder or recraft the exercises eliminating or isolating troublesome topics (at least initially) inside instructor-supplied functions. For example, decoding message packets is the most sophisticated task and some instructors might want it last in the sequence. They can continue to use the BBC app for this task until then. Other instructors might forego having students write a hash function (although the needed hash function is quite tame) and simply supply the hash function code up front, with or without a discussion of how it works.

Digital Signal Processing (DSP) Course

While crafting an entire semester syllabus in digital signal processing around concurrent codes would be difficult, they can readily serve as the underlying topic for exercises constituting perhaps one quarter to one half of the course. Furthermore, a DSP course probably would not have students implement the coding and decoding algorithms, especially if these were being done in a separate programming course. Instead, the course would use the BBC application to perform these steps while the students focus on understanding, designing, and applying the various filters.

Initially, the students would design the filters and enter the coefficients into a BBC script. This allows them to focus on the math involved in the filter design and not get distracted by the housekeeping involved in actually implementing the filter. Then, with the math understood, they could implement the actual filters and filter the waveforms separate from the BBC application either by using a program they've written or a commercial application such as Excel or MATLAB. Since the BBC application can read and write the data at any point in the processing, either as an ASCII text file or as a Windows WAV file, the instructor can easily choose which parts of the processing the students are to perform separately.

Another aspect that makes concurrent codes a good test bed for an introductory DSP course is that students can tackle the problem in incremental steps. They can first encode the packet into a waveform file and then decode that same waveform file, allowing them to perform their signal processing on ideal waveforms. They can then play and record the file in order to see the effect that filtering has on helping with the inevitable distortion and noise. Finally, then can see how their processing is able to deal with actual jamming signals from their peers.

THE BBC APPLICATION

The open-source BBC application is an ANSI-C compliant, script-driven processing engine that can perform many tasks associated with concurrent codes. It can be used for the initial exploration of concurrent codes and to focus and guide students' efforts throughout the exercises. The following BMP.bbc script implements a simple end-to-end image-based system. In a full semester programming course, students would incrementally comment out commands as they complete the objectives (see the "Student Exercises" section) and their code becomes capable of performing the associated tasks.

```
// BMP.bbc - BBC Script Template for BMP files
CLEAR -a                                // Objective #5
CODEC -m 512 -e 1000 -k 50              // Objective #5
PACKET -e 1                             // Objective #5
```

```
MSG -r "pic_msgs.txt"                    // Objective #4
ENCODE -m                                // Objective #4
BMP -w "demo.bmp"                        // Objective #7
PAUSE -r                                 // Objective #8
BMP -r "demo.bmp"                        // Objective #8
MSG -e                                   // Objective #6
DECODE -s                                // Objective #6
MSG -d -w "received.txt"                 // Objective #6
```

The BMP.bbc script first clears any existing data and configuration information from the processing engine. It then configures the encoder/decoder (codec) for messages that are 512 bits long plus 50 checksum bits. The codewords are 1000 times as long as the base message (512,000 bits). The packet into which codewords are inserted has an expansion of one, making it the same length as a single codeword. The messages are then read from a text file in which each line is a separate message. They are then encoded into the packet (on top of each other via bitwise-OR). The packet is then formatted as a Windows bitmap file and written to disk. The next line pauses the engine until the user hits ENTER, thereby allowing the user to an image editing tool, such as Microsoft Paint, to add additional black marks to the image and save it back on top of itself. After hitting ENTER, the BBC program reads the file, erases all messages in the internal message buffer, and decodes the packet adding each message found to the buffer. Finally the message list is displayed on the screen and written to a text file.

While an image-based system is adequate for a programming course, an audio-based system, such as that implemented by WAV.bbc below, is more suitable for a digital signal processing course. As before, students begin with a complete, functioning system and progressively take responsibility for its functions. However, in this case, students would probably only perform the tasks indicated with an asterisk and continue to use the BBC application for the rest.

```
// WAV.bbc - BBC TX/RX Script Template for the WAV files
CLEAR -a
CODEC -m 128 -e 100 -k 50
PACKET -e 1
MSG -r "tx.txt"
ENCODE -m
WAV -c -b 1000 -s 11025 -o 0 -e 80 -f 5000 -g -w "tx.wav" // [*]
PAUSE -r

// Configure Filter
// http://www.dsptutor.freeuk.com/remez/RemezFIRFilterDesign.html
// High Pass; Fs  = 11.025 kHz; F_o =  4.4kHz (0.4*Fs)
// Transition = 1100Hz (0.1*Fs)
// Passband Ripple: 1 dB; Stopband Rejection: 40 dB
FILTER -x 0   0.026326745981419188                          // [*]
FILTER -x 1   2.3394244186887697E-4                         // [*]
FILTER -x 2  -0.062132178678304895                          // [*]
FILTER -x 3   0.1601061482350591                            // [*]
FILTER -x 4  -0.25254345256890826                           // [*]
FILTER -x 5   0.2905859513171658                            // [*]
FILTER -x 6  -0.25254345256890826                           // [*]
FILTER -x 7   0.1601061482350591                            // [*]
FILTER -x 8  -0.062132178678304895                          // [*]
FILTER -x 9   2.3394244186887697E-4                               // [*]
FILTER -x 10  0.026326745981419188                                // [*]

WAV -r "rx.wav"
FILTER -s                                                         // [*]
RAD -f 95.0 -s                                                    // [*]
WAV -l 2.0 -n 0 1                                                 // [*]
DSC -1 10.0 -0 10.0 -s                                            // [*]
WAV -p 1.00                                                       // [*]
MSG -e
DECODE -e 10000 -s
MSG -w "rx.txt"
```

The WAV.bbc script starts out similar to the BMP.bbc script by configuring the basic codec, reading messages from a text file, and encoding them into a packet. The next line configures the transmitter's modulator by setting the packet bit rate to 1 kbps, the sampling rate to 11.025 kHz, and the packet delay (origin) to 0 samples. It further defines a transmitted mark as consisting of a 5 kHz sinusoidal burst lasting 80% of a bit duration. Finally, it generates the waveform data and writes it to a Windows wave file. Instructors may elect to forego having students implement the transmitter in favor of focusing on the more DSP-intensive receiver functions.

After pausing to give the user the opportunity to play the resulting file (perhaps along with others that have been generated) while recording a new file, the script creates a finite impulse response filter by explicitly defining its coefficients and then, after reading the data from the input audio file, applies the filter. A simple infinite impulse response radiometer is then defined wherein 95% of the prior output is combined with 5% of the present input. The resulting waveform is then clipped at a maximum amplitude of 2 and the result is normalized to a range of 0 to 1. The next step implements a Schmitt trigger with, in this case, both the rising and falling thresholds set to 10% of the normalized range. Following this, a binary packet is produced from the waveform data by declaring a mark (HI) to exist if the discrimator output is HI anywhere within a 1.0*bit-period window centered on the nominal bit center. After this, the rest of the processing is similar to the BMP.bbc script except that the decoder is told to assume that the transmitting and receiving oscillators might be off by as much as 1% (10000 ppm).

CONCLUSIONS

Concurrent codes offer an opportunity to craft substantive exercises in either a computer programming or a digital signal processing course that have meaningful relationships to the growing real-world problem of keyless jam resistance. Furthermore, students can benefit by the availability of the BBC application program that allows them to work with a complete encoding/decoding system at each step of their work enabling them to verify successful implementation of the portion of the problem they have been tasked with. The end result should be greater student interest in the exercises with a correspondingly better mastery and retention of the course material.

REFERENCES

[1] Baird III, L.C., Bahn, W.L., Collins, M.D., Jam-resistant communication without shared secrets through the use of concurrent codes, *Technical Report USAFA-TR-2007-01*, United States Air Force Academy, 2007.