### Getting Started

1) Configurations

- Open up Main.cpp. At the top of the file, there should be two #define directives, FFT and TESTS.

- To enable tests BEFORE executing the program, uncomment #define TESTS.

- Note that tests will delay the execution of the program by at least a minute

- To use the algorithm based optimization, FFT Convolution, uncomment #define FFT

2) Compilation

g++ \*.cpp –o a4 (or g++-4 \*.cpp –o a4 for 64-bit Windows Cygwin users)

3) Execution

./a4 (or a4 for Windows users)

### What Works

* Baseline Program (on .wav files)
* Algorithm-Based Optimization version of Baseline (on .wav files)
* Bonus #1: Stereo detection on Impulse Response (on .wav files)

### What Doesn’t Work

* Bonus #2: Handle aiff and snd file formats
* Using a snd file as any input or output file will produce a file that CAN be opened, but does not sound correct.
* Only a wave input / snd impulse / wav output, sounds close, but still not correct.
* I narrowed the reason down to reading the SND file data. Everything else works because I tried creating an SND file with WAVE data and it works. Therefore I’m reading the SND file data incorrectly, but I’m not sure why what I have is wrong.
* Using an aiff file as any input or output file will either produce (1) a file that CANNOT be opened (only tested with Windows Media Player) or (2) an openable file that has incorrect sounding output.
* I didn’t have much time to test this further, so I’m not sure why it doesn’t work.
* It may have to do with the fact that I just made a barebones AIFF file (all chunks except Common and Sound Data are optional, see References [5]). I tested opening the sound files with Windows Media Player, and it may have relied on these extra chunks.
* NOTE: I used sox to convert the dry recording and impulse responses from wave to snd and aiff. It is possible that I used sox incorrectly (maybe I didn’t set endian, b, c, or r flags correctly, or maybe more flags needed to be set) and that may have contributed to my strange outputs.

### Testing

* Unit tests are provided. four1 FFT and IFFT, and convolve/fftConvolve wrappers that take SoundFile\* are more difficult to test, and they are not tested directly.
* However, these convolve/fftConvolve wrappers are wrapping convolve/fftConvolve methods that take double\* signals. These lower level methods *are* tested.
* Four1 is indirectly tested through the regression test.
* The main regression test is found in RegressionTest.cpp and compares the output of the baseline program, with the algorithm-based optimization version of the baseline program (i.e. using FFT convolution).
* All tests were run after every optimization.

**Testing Constants and Inputs**

* Dry Recording: **testCase1.wav** – Mono 16-bit Sample Size
* Impulse Response: **Parking Garage.wav** – Mono 16-bit Sample Size

### Profiling

**Tool:** Microsoft Visual Studio Profiling Tools (see References [0])

**Measurement:** *Elapsed Inclusive* (for the whole program, see References [1]). This includes function stack set up and tear downs and system calls. Tests are turned off before profiling so tests are not included.

### Optimizations Performed

**Template**

**Code Before** **Code After**

<CODE BEFORE> <CODE AFTER>

**Steps**

1. A
2. B
3. C

**Total Time Before:**

**Total Time After:**

These times are measured as *Elapsed Inclusive* in the scope of the entire program (i.e. main()). This means that the total time that is spent executing the *program* from start to finish is measured. This includes function stack set up and tear downs, as well as system calls. Tests are turned off before profiling so tests are not measured.

**Speed Gain:**

**Results So Far:**

This shows the trending total execution time of the program thus far. Ideally, as this table grows, the execution time as it grows.

\*\* NOTE \*\*

The total time before any optimizations (i.e. the total time of the baseline program) is **359525.44 ms**

**Algorithm Based Optimization – FFT Convolution**

**Code Before** **Code After**

int main(int argc, char\* argv[]) {

// dry is input signal, ir is impulse

Convolver::fftConvolve(dryNormalized,

dry->getDataSize(),

irNormalized,

ir->getDataSize(),

result, P);

. . .

}

}

int main(int argc, char\* argv[]) {

// dry is input signal, ir is impulse

Convolver::convolve(dryNormalized,

dry->getDataSize(),

irNormalized,

ir->getDataSize(),

result, P);

. . .

}

}

\* See Convolver.cpp for convolve() \* See Convolver.cpp for fftConvolve()

**Steps**

1. Make fftConvolve() for the optimized algorithm with the same arguments/signatures as convolve()
2. Convert input signals to frequency domain before applying convolution
3. Convert results back to time domain

Step 2 is where the magic happens. Only one linear loop, rather than a nested loop, through the signal for complex multiplication is required. O(N) is better than O(N\*M).

**Total Time Before:** 359,525.44 ms

**Total Time After:** 8,637.24 ms

**Speed Gain:** 350888 ms

**Results So Far:**

|  |  |
| --- | --- |
| **Optimization Technique** | **Time (ms)** |
| None | 359,525.44 |
| FFT Convolution | 8,637.24 |

**Code Tuning -**

**Code Before** **Code After**

**Steps**

**Total Time Before:**

**Total Time After:**

**Speed Gain:**

**Results So Far:**

### References

Microsoft Visual Studio Profiling Tools: [0] <http://msdn.microsoft.com/en-us/library/z9z62c29.aspx>

Documentation: [1] <http://msdn.microsoft.com/en-us/library/dd264994.aspx>

Snd File Format:

[2] <http://sox.sourceforge.net/AudioFormats-11.html>

[3] <http://en.wikipedia.org/wiki/Au_file_format>

[4] <http://www-mmsp.ece.mcgill.ca/documents/audioformats/AU/AU.html>

Aiff File Format:

[5] <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/AIFF/Docs/AIFF-1.3.pdf>

[6] <http://muratnkonar.com/aiff/index.html>

Leonard Manzara & Abbas Sarraf : Amazing and helpful professor and TA.