Name: Yuan Zhou
ID: 10130623

READ ME:

The library “libsndfile” is used to read and write wav audio file which is approved by the professor. If you do not have this library on your system, please follow the steps below:

1. Intall on linux system by entering these

   ```
   sudo apt-get install libsndfile1
   sudo apt-get install libsndfile1-dev
   sudo apt-get install sndfile-programs
   ```

2. Please compile wih “gcc file.c -o file -lsndfile -lm” please

The regression testing will be shown through all steps of optimization throughout the report.

Used GuitarDry.wav and Mahal.wav for the whole report.

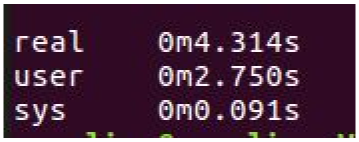
Base Program

Time measurement with and without different compiler optimization

optimization	Time Measurement
-o	25m 30.421s
-o1	27m 35.968s
-o2	22m 45.771s
-o3	20m 15.555s

Algorithm Base Program

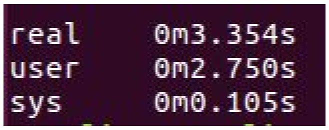
Time without code tuning and without compiler optimization



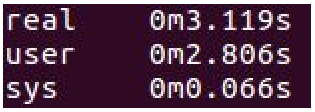
```
real	0m4.314s
user	0m2.750s
sys	0m0.091s
```

Time with compiler optimization

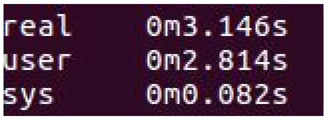
Compiled with	Time Measurement
-o1	3.354
-o2	3.119
-o3	3.146



```
real	0m3.354s
user	0m2.750s
sys	0m0.105s
```



```
real	0m3.119s
user	0m2.806s
sys	0m0.066s
```



```
real	0m3.146s
user	0m2.814s
sys	0m0.082s
```

It is obvious that with the compiler optimization, the program do have a better run time, however, compiling with -o3 did not show any improvement from -o2.

Code Tuning

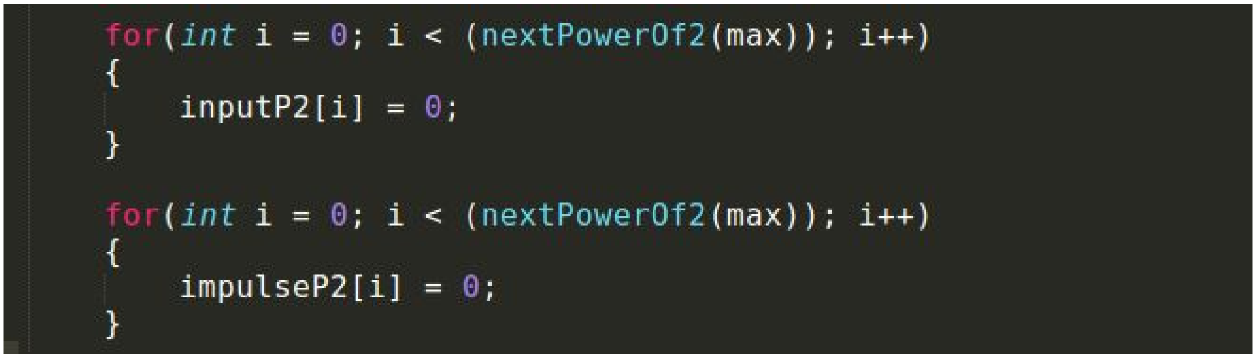
Result Table

Original time measurement before any code tuning: 4.313 seconds.

Code Tuning	Improvement(sec)	Time
Jamming	1.092	3.221
Minimize work of loop	0.297	2.924
Eliminate common subexpressions	0.064	2.860
Use proper data for constants	0.061	2.799
Logical: stop testing once you know the answer	-0.014	2.813

1. **Jamming**

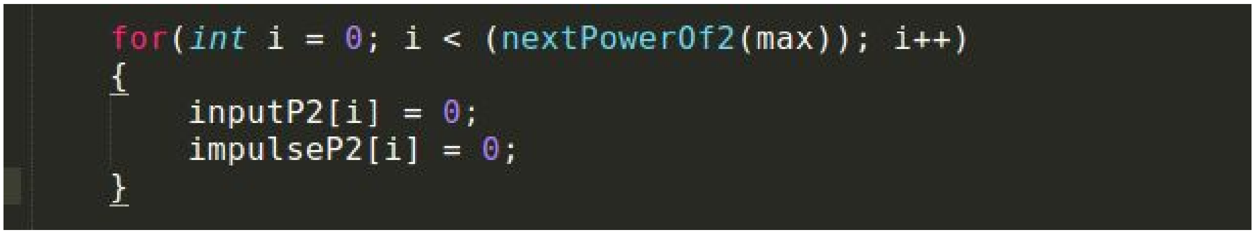
before



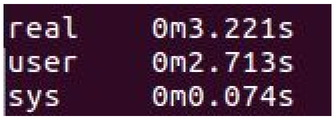
```
for(int i = 0; i < (nextPowerOf2(max)); i++)
{
    inputP2[i] = 0;
}

for(int i = 0; i < (nextPowerOf2(max)); i++)
{
    impulseP2[i] = 0;
}
```

after



```
for(int i = 0; i < (nextPowerOf2(max)); i++)
{
    inputP2[i] = 0;
    impulseP2[i] = 0;
}
```

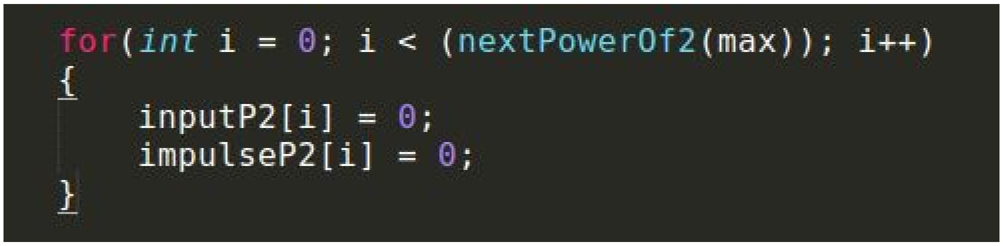


```
real    0m3.221s
user    0m2.713s
sys     0m0.074s
```

The program was originally 4.313 seconds, with an improvement of 1.092 seconds the program now is 3.221 seconds. We can see the extra loop takes a very long time to process.

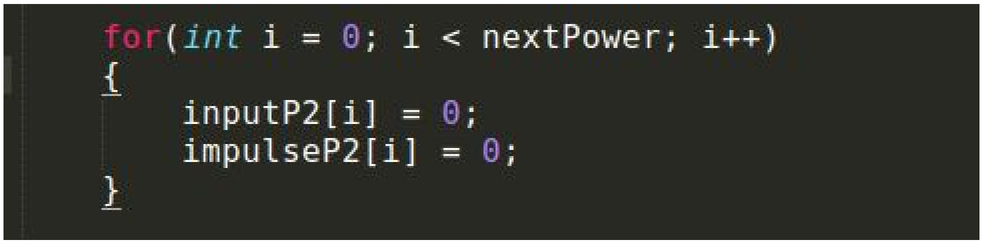
2. **Minimizing work of loop**

before

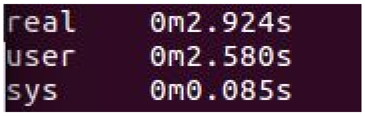


```
for(int i = 0; i < (nextPowerOf2(max)); i++)
{
    inputP2[i] = 0;
    impulseP2[i] = 0;
}
```

after



```
for(int i = 0; i < nextPower; i++)
{
    inputP2[i] = 0;
    impulseP2[i] = 0;
}
```

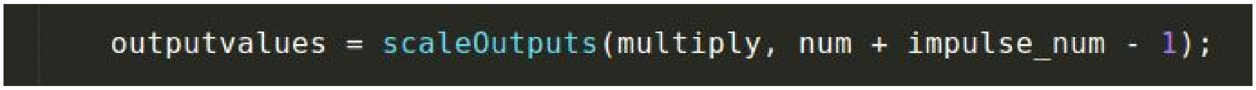


```
real    0m2.924s
user    0m2.580s
sys     0m0.085s
```

Rather than accessing calculating “nextPowerOf2(max)” every time throughout the loop. Calculate this unchange value ahead of time greately change the performance of the program.

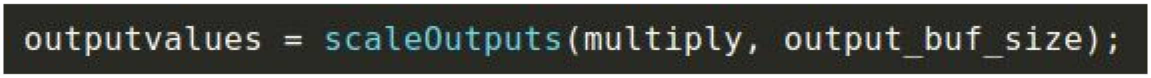
3. Eliminate common sub-expressionss

before

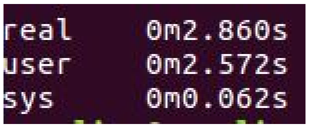


```
    outputvalues = scaleOutputs(multiply, num + impulse_num - 1);
```

after



```
outputvalues = scaleOutputs(multiply, output_buf_size);
```

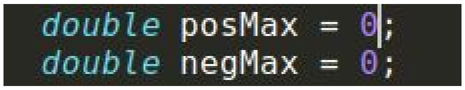


```
real    0m2.860s
user    0m2.572s
sys     0m0.062s
```

While there is no need to recalculate a value while we have a variable that store this value in memory already. It is obvious that doing this small step helps with the improve the speed of program.

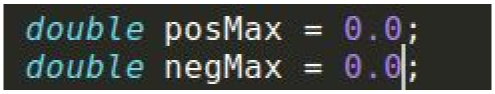
4. Use proper data

before

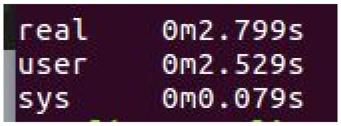


```
  double posMax = 0;
  double negMax = 0;
```

after



```
 double posMax = 0.0;
 double negMax = 0.0;
```

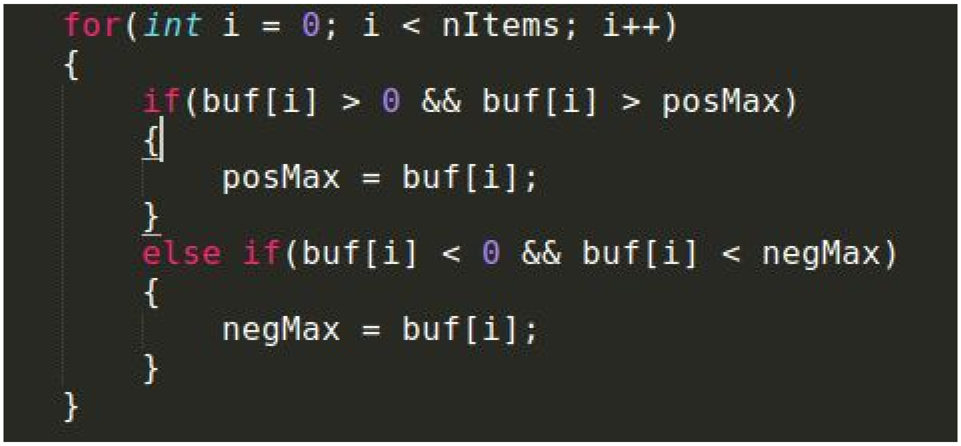


```
real    0m2.799s
user    0m2.529s
sys     0m0.079s
```

Changing the “0” to “0.0” ffor a double variable helps to avoid runtime type conversion. The difference in time is not a lot, but it is enough to keep this pratice in mind to ensure that we maximize the code efficiency.

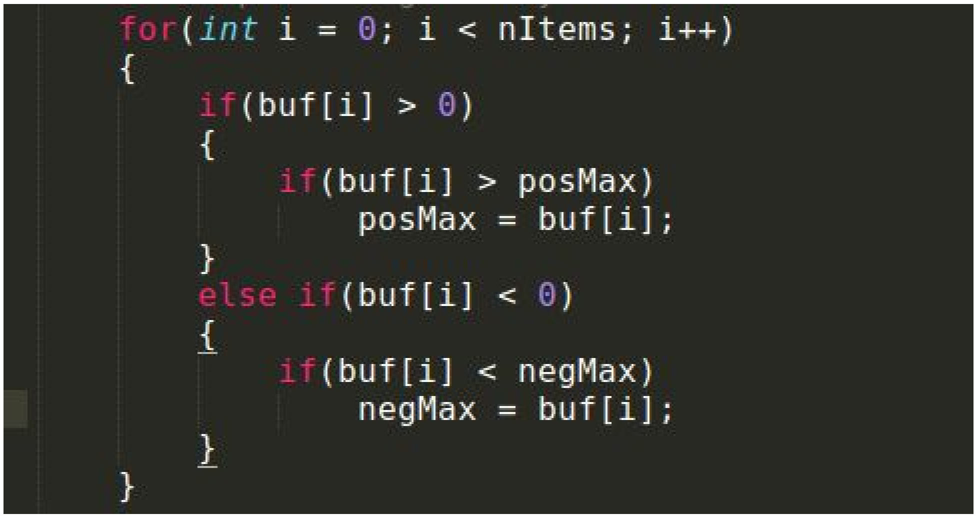
5. **Logical: stop testing once you know the answer(short circuit evaluation)**

before

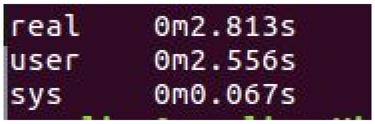


```
for(int i = 0; i < nItems; i++)
{
    if(buf[i] > 0 && buf[i] > posMax)
    {
        posMax = buf[i];
    }
    else if(buf[i] < 0 && buf[i] < negMax)
    {
        negMax = buf[i];
    }
}
```

after



```
for(int i = 0; i < nItems; i++)
{
    if(buf[i] > 0)
    {
        if(buf[i] > posMax)
            posMax = buf[i];
    }
    else if(buf[i] < 0)
    {
        if(buf[i] < negMax)
            negMax = buf[i];
    }
}
```



```
real	0m2.813s
user	0m2.556s
sys	0m0.067s
```

This code tuning did not show any improvement. So i change the code to the before to optimize performance.

Time with code tuning and compiler optimization

Compiled with	Time Measurement
-o	3.124
-o1	2.894
-o2	2.878
-o3	2.803

After all the code tuning, i tested all the compiler optimization again. We can see once we optimize the code manually, it also helps with the compiler optimization part.

Profiler with compiler optimization (-o3)
-- in file “profilerWithCompilerOpt.txt"

Profiler with code tuning and compiler optimization (-o3)
-- in file “profilerAfterTuning.txt”