CS 378 – Homework #3

Due Thursday, Oct 2, Midnight

1 Easy Loop

Let's start with a simple loop.

```
void warmup(float* x, float* y, int size, float alpha) {
  for (int i = 0; i < size; ++i)
    y[i] = x[2*i] * x[2*i] + x[2*i+1]/alpha
}
```

Produce a simple (scalar) implementation and a vectorized implementation (using intrinsics). Use the highest available SSE versions on stampede. Time the two loops for x = 1600 and y = 800. The implementation should not assume x and y are aligned, but should only used aligned accesses in the main portion of the implementation. Compare the outputs of the vectorized and basic loop to ensure they produce the same value.

2 Finite Impulse Response

`https://en.wikipedia.org/wiki/Finite_impulse_response`

Let's try a real loop.

```
//h is length 4, y[i] with i < 4 should be 0
void FIR(float* y, float* x, float* h, int size) {
  for (int i = 4 ; i < size; i++)
    y[i] = h[3] * x[i-3] + h[2] * x[i-2] + h[1] * x[i-1] + h[0] * x[i];
}
```

Produce a simple implementation and a vectorized implementation (using intrinsics). Use the highest available SSE versions on stampede. Time the two loops for x = 1600 and y = 1600. The implementation should not assume x and y are aligned, but should only used aligned accesses in the main portion of the implementation. Compare the outputs of the vectorized and basic loop to ensure they produce the same value.

Turning in

Your code should use explicit intrinsics for vectorization. Measurements should be taken on stampede at tacc. These implementations should run fast enough that running on the login node should be fine. You may submit jobs to run the code if you want, but it isn't necessary. For the second problem, please explain what the key problem is which makes vectorization hard, your approach to vectorization, and the number of samples per second you compute (each element of y is a sample). Turn in by email an archive (.zip or .tgz) to the TA.

Extra Credit

Try applying the FIR to audio. Use:

```
h = {0.25, 0.25, 0.25, 0.25}
```

for the filter coefficients. Listen to the result. What happened?

16 bit stereo audio is 2 streams of 16 bit signed integers. Apply the filter to each stream separately.

I think it is easiest to convert a file to a raw file (just an array of samples) first rather than dealing with file formats for this question.

Decode a flac file to a stero raw file:

```
flac -d --force-raw-format --endian=little --sign=signed -c file.flac > foo.raw
```

The result is a binary file of int16_t data in L0,R0,L1,R1,L2,R2... format (left sample, right sample, left sample, right sample...).

Playing a raw file:

```
cat foo.raw |aplay -f S16_LE -c2 -r44100 -t raw
```

Both playing and converting existing files is possible from most any audio processing tool. audacity is quite capable for both these tasks too and is available on Linux and Windows.