Gorman Law

10053193

Code Tuning

First tuning: Jamming (Fusion)

Original Code:

for(int i = 0; i < nn\_2; i++)  
{

x[i] = 0;

}

for(int i = 0; i < nn\_2; i++)

{

h[i] = 0;

}

for(int i = 0; i < nn\_2; i++)

{

y[i] = 0;

}

After jamming:

for(int i = 0; i < nn\_2; i++)

{

x[i] = 0;

h[i] = 0;

y[i] = 0;

}

We move the padding inside one loop

Code tuning 2: Minimizing work inside arrays

Original code:

double MAX\_VAL = 32767;

// Complex multiplication i think (?)

for(int i = 0 ; i < nn\_2; i+=2)

{

y[i] = (x[i]/MAX\_VAL \* h[i]/MAX\_VAL) - (x[i + 1]/MAX\_VAL \* h[i + 1]/MAX\_VAL);

y[i + 1] = (x[i + 1]/MAX\_VAL \* h[i]/MAX\_VAL) + (x[i]/MAX\_VAL \* h[i + 1]/MAX\_VAL);

}

Tuned code: divide by a constant value beforehand

for(int i = 0 ; i < nn\_2; i+=2)

{

y[i] = (x[i] \* h[i]) - (x[i + 1] \* h[i + 1]);

y[i + 1] = (x[i + 1] \* h[i]) + (x[i] \* h[i + 1]);

}

//done while reading data

data[i++] = (double)sample/MAX\_VAL;

Code tuning 3: Precompute values at compile time

Original Code

h = (double\*) malloc(sizeof(double) \* nn\_2);

x = (double\*) malloc(sizeof(double) \* nn\_2);

y = (double\*) malloc(sizeof(double) \* nn\_2);

outdata = (double\*)malloc(sizeof(double) \* nn);

Tuned code

#define SIZE\_OF\_DOUBLE sizeof(double)

h = (double\*) malloc(SIZE\_OF\_DOUBLE \* nn\_2);

x = (double\*) malloc(SIZE\_OF\_DOUBLE \* nn\_2);

y = (double\*) malloc(SIZE\_OF\_DOUBLE \* nn\_2);

outdata = (double\*)malloc(SIZE\_OF\_DOUBLE \* nn);

Precalculate a constant, so we don’t have to do it again later.

Code tune 4: Strength reduction

Original Code

for(int i = 0; i < dryNumSamples; i++)

{

x[2 \* i] = data[i];

}

for(int i = 0; i < irNumSamples; i++)

{

h[2 \* i] = irdata[i];

}

for(int i = 0; i < nn\_2; i++)

{

outdata[i] = y[i\*2]/(nn\_2 \* 2);

}

Tuned Code: Do a shift instead of using multiplication

x[i << 1] = data[i];

h[i << 1] = irdata[i];

outdata[i] = y[i << 1]/(nn\_2 << 1);

Shift to the left instead of multiplying by 2. It’s faster!

Code Tune 5: Minimize array references

Original Code:

y[i] = (x[i] \* h[i]) - (x[i + 1] \* h[i + 1]);

y[i + 1] = (x[i + 1] \* h[i]) + (x[i] \* h[i + 1]);

Tuned Code:

xi = x[i];

xi1 = x[i+1];

hi = h[i];

hi1 = h[i+1];

y[i] = (xi \* hi) - (xi1 \* hi1);

y[i + 1] = (xi1 \* hi) + (xi \* hi1);

Access the array less. In the original code, the array is accessed twice. After tuning, they are only accessed once!

Code tuning 6: Unrolling

Original Code:

for(int i = 0; i < nn\_2; i++)

{

x[i] = 0;

h[i] = 0;

y[i] = 0;

}

Tuned Code:

for(int i = 0; i < nn\_2; i+=2)

{

x[i] = 0;

x[i+1] = 0;

h[i] = 0;

h[i+1] = 0;

y[i] = 0;

y[i+1] = 0;

}

Self explanatory