

Counter Intuitive Results: Optimising an FFT routine for Speed

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2003-10/1144.html

From: Paul Brown (postmaster_at_pwi.mailshell.com)

Date: 10/07/03

Date: 7 Oct 2003 07:08:33 -0700

OK – I feel somewhat exonerated.

I tried various combinations of options with the high iteration code segment. Indeed the "pointerized" version DID RUN SIGNIFICANTLY FASTER (as in 21%) when I revert to 32 bit floating point here instead of double (for those entering the thread here I repeat the salient code).

This is strange but credible, I understood that all internal arithmetic was carried out with doubles, and casts to and from float performed as necessary to match the variables data types. This does not appear to be the case with Turbo-C

Now I can put my present trials to rest and start hunting through FFTW.

BTW – I could not get CYCLE.h to work, Turbo-C does not support `__INLINE__` or `__ASM__`. Good thing I spent the time to get my `****PORTABLE****` timer to work.

```
/*
  _1 curReal = wr *data[ ixj ] -wi *data[ ixj
+1];
  _1 curImag = wr *data[ ixj+1 ] +wi *data[ ixj
];
  _1 data[ ixj ] = data[ ixData ] -curReal;
  _1 data[ ixj+1 ] = data[ ixData+1 ] -curImag;
  _1 data[ ixData ] += curReal;
  _1 data[ ixData+1 ] += curImag;
*/

#if defined(FFT_FLOAT)
# define srcPtr0_ floatPtr_1__
# define srcPtr1_ floatPtr_2__
# define tgtPtr0_ floatPtr_3__
# define tgtPtr1_ floatPtr_4__
#endif
```

comp.lang.c: Counter Intuitive Results: Optimising an FFT routine for Speed

```
srcPtr1_ = srcPtr0_ = &( data[ixj]); ++srcPtr1_;  
tgtPtr1_ = tgtPtr0_ = &( data[ixData]); ++tgtPtr1_;  
curReal = wr **srcPtr0_ -wi **srcPtr1_;  
curImag= wr **srcPtr1_ +wi **srcPtr0_;  
*srcPtr0_ = *tgtPtr0_ -curReal;  
*srcPtr1_ = *tgtPtr1_ -curImag;  
*tgtPtr0_ += curReal;  
*tgtPtr1_ += curImag;
```

```
# undef srcPtr0_  
# undef srcPtr1_  
# undef tgtPtr0_  
# undef tgtPtr1_
```

This is a typical execution time with full double variables

FFT_NRC starting to execute 5000 loops
FFT_NRC time = 923647 ±88(3.5s) ns

Here is the comparison between 32 bit floats, FFT_NRC_prev is the _1
code commented out above and FFT_NRC is the pointerized code :

FFT_NRC starting to execute 5000 loops
FFT_NRC time = 519175 ±88(3.5s) ns

FFT_NRC_prev starting to execute 5000 loops
FFT_NRC_prev time = 658008 ±88(3.5s) ns

FFT_NRC time change :FFT_NRC_Prev = -138.833 ± 0.124(3.5s) µs
(-21.10%)

FFT_NRC_prev starting to execute 1000 loops
FFT_NRC_prev time = 654509 ±451(3.5s) ns

FFT_NRC starting to execute 1000 loops
FFT_NRC time = 515313 ±451(3.5s) ns