Optimization of the C program summary

The general approach to optimize the code was done in a few steps.

Step 1:

- convert derivations of constants from other constants, e.g. a = 3.3 b= log(a), to a constant value. This is only noticeable when optimization is turned off.
- convert pow(x,integer) to a series of multiplies. This replaces a several dozen operations to only a few multiplies.

Step 2:

- convert pow(x,float y) to exp(y *log(x)). The exponential function is faster than the power function. In this code there are multiple calls to the power function of the same significand pow(fpt,blah). If we precompute the value of log(fpt), there are big savings.
- convert the loops over floating point values in main functions to ones over integers.
- Multithread the code with openmp so several points can be done at once. We can expect a speedup proportional to the number of cores.

I ran a series of timed tests (using bash time) on each version of the code with a linux desktop running Ubuntu 13 with the gcc-13 compiler. The CPU was an Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz with 4 cores and hyperthreading purchased in 2011. All times were collected using the bash time utility and the speedups are relative to the original code.

time(secs)			
version	no optimization	-O	-O2 -ffast-math
original	436.83	234.85	0.934
change 1	191.64	171.15	0.938
change 2	121.27	109.5	0.002
change 2 + mp	27.92	22.15	0.001

speed up			
original	1	1	1
change 1	2.279430182	1.372188139	0.9957356077
change 2	3.602127484	2.144748858	467
change 2 + mp	15.64577364	10.6027088	934

Profiles obtained with gprof utility are available in the github repo. Note that the openmp speedup is roughly 4 times that of the unthreaded code which is to be expected for a CPU with 4 cores. Overall, the optimizations were fairly effective in reducing the runtime.